Foundations of Temporal Constraint Databases

Manolis Koubarakis

Foundations of Temporal Constraint Databases

Manolis Koubarakis

Doctoral Dissertation

February 1994

Computer Science Division
Dept. of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece

Abstract

An important requirement of advanced temporal applications is the ability to deal with *definite, indefinite, finite* and *infinite* temporal information. There is currently *no* database model which offers this functionality in a single unified framework. We argue that the combination of *relational databases* and *temporal constraints* offers a powerful framework which addresses these needs. In this thesis we develop the foundations of a theory of *temporal constraint databases* and *indefinite temporal constraint databases*. At first, we study a hierarchy of parameterized database models: $\mathbf{M}_{\mathcal{L}}$-relational databases, $\mathcal{L}$-constraint databases and indefinite $\mathcal{L}$-constraint databases. The language $\mathcal{L}$, the parameter, defines the constraint vocabulary and $\mathbf{M}_{\mathcal{L}}$ is the structure over which $\mathcal{L}$-constraints will be interpreted. The models of temporal constraint databases and indefinite temporal constraint databases can then be studied as *instances* of the last two of the above parameterized models. In the course of our study we develop quantifier elimination and decision algorithms for several theories of temporal constraints. The most important theories we consider are subtheories of Presburger arithmetic (in the case of discrete time) or real addition with order (in the case of dense time). Then we use these results to analyze the complexity of query evaluation in temporal constraint databases and indefinite temporal constraint databases. Our analysis shows that there is *no change* in data complexity or combined complexity when we go from relational databases to temporal constraint databases (with or without indefinite information). Finally, we study the properties of a new tractable class of temporal constraints involving disjunctions of inequations.

Acknowledgements

I would like to thank John Mylopoulos and Alberto Mendelzon who were my advisors while I was a student at the University of Toronto. John introduced me to the area of knowledge representation, offered me the privilege to work in the Telos project and had a profound influence on my research throughout my studies at Toronto. Alberto kindly accepted to be my co-supervisor when my interests started becoming more theoretical. He was always very helpful and improved my understanding of various database theory problems.

When I decided to move to Greece, Timos Sellis welcomed me at the National Technical University of Athens and agreed to supervise my work on the topic of temporal constraint databases. Since then he has been a good friend and a valuable source of support and new ideas.

Foto Afrati, Yannis Vassiliou, Stathis Zachos, Nick Roussopoulos, Manolis Skordalakis and Andreas Stafylopatis served on my thesis committee. I would like to thank all of them for their comments and suggestions. Stathis in particular was always there when I urgently needed to discuss some aspect of complexity theory!

I am grateful to Prof. Barry Richards of Imperial College who gave me time to work on the final draft of this dissertation after I joined his group.

I also want to thank my friends and colleagues Thodoros Topaloglou, Dimitris Plexousakis, Brian Nixon, Lawrence Chung, Martin Stanley, David Wilkes and Murray Mazer (University of Toronto) and Dimitris Papadias, Yannis Theodoridis and Kyriakos Diakonikolaou (National Techinical University of Athens). Thodoros and Dimitris read this dissertation (as most of my previous papers!) and offered valuable comments and advice.

Throughout my studies, I received financial support from the University of Toronto, the National Techical University of Athens, the University of Crete and the Institute of Computer Science, FORTH. I would like to thank all the people at these institutions who supported me through their research grants.

Finally, I would like to thank my family and my fiancee Vicky for their love and encouragement throughout this long but rewarding journey.

Dedication

This dissertation is dedicated to my family.

Contents

Chapter 1

Introduction

The study of time has attracted the attention of philosophers ever since the ancient times. More recently plilosophers, logicians and linguists initiated the study of temporal logic, motivated by temporal phenomena as they manifest themeselves in natural language [Pri57, Pri67, RU71, DWP81, vB83]. In the last fifteen years, reasoning about time and change has also received considerable attention in a number of areas of computer science. These areas include artificial intelligence, database and information systems and computer-aided verification. This trend is quite natural since many problems studied by computer scientists require temporal reasoning. For example, a planning program often needs to decide whether a task can be executed before another task and meet a given deadline [AKP91]. A medical expert system has to reason about the course of a disease over time [Dut88]. A company database might need to keep a record of past salaries of employees [TCG$^+$93]. A geographical information system has to store the changes in ownership of a parcel of land over time [BFAT91]. Finally, in computer-aided verification, the designer of a system might be interested in verifying the property "channel A delivers message B 3 to 5 seconds after its receipt" [MP91, Cou93]. As it has been pointed out in [vB89b], computer science has not just been a customer of existing temporal logic; it has also enriched traditional temporal logic in various ways.

1.1 Overview

In this dissertation we will be interested in reasoning about time as it arises in databases and artificial intelligence. We will make a conscious attempt to integrate concepts from both of these research programs. Our proposal, originally put forward in [Kou93a], can be summarized as follows. We believe that applications involving temporal reasoning can benefit from the development of a database model which allows the representation of *definite, indefinite, finite* and *infinite* temporal information in a single unifying framework. This can be achieved with the graceful integration of concepts and techniques from three areas:

- Database models for the representation of *indefinite information* (most prominently [IL84, Gra89]).
- The *constraint database models* of [KKR90] and [KSW90].
- Formalisms for reasoning with temporal constraints [All83, VK86, VKvB89, vBC90, vB92, DMP91, KL91, Mei91a, Lad88].

Independently a similar proposal has been made in [Cho93] where the temporal database model of [Kou93a] is discussed in some length.

This dissertation puts the above proposal on a strong formal basis. We lay the foundations of a theory of *indefinite temporal constraint databases* based on the relational model [Mai83]. As a starting point of our investigation, we take the model of constraint databases proposed in [KKR90]. This model is useful for the representation of *unrestricted* (i.e., finite or infinite) *definite* information. However, indefinite information is also important in many temporal applications e.g., planning and scheduling [AKP91, SFG85], medical expert systems [Dut88], geographical information systems [BFAT91], office information systems [BP85] and natural language processing systems [MS88b]. Motivated by these practical considerations, we develop the model of *indefinite constraint databases* by integrating the constraint database model of [KKR90] and the *conditional table* model of [IL84, Gra89].

Our contributions to the theory of constraint databases is the following:

- Following the approach of [JL87, Mah93], we develop three database schemes: $\mathbf{M}_{\mathcal{L}}$-relational databases, $\mathcal{L}$-constraint databases and indefinite $\mathcal{L}$-constraint databases (chapters 4 and 5). The language $\mathcal{L}$ defines the constraint vocabulary and $\mathbf{M}_{\mathcal{L}}$ is the structure over which $\mathcal{L}$-constraints will be interpreted. The last two schemes are interesting from a practical point of view. $\mathbf{M}_{\mathcal{L}}$-relational databases simply serve as a theoretical tool for providing semantics for the other two schemes.

- We provide algebraic and calculus query languages for the above schemes and study the semantics of query evaluation.

We obtain temporal relational databases, temporal constraint databases, and indefinite temporal constraint databases as instances of the above schemes. The developments in [KSW90, Kou93a] can also be seen as instances of these schemes.

Then we proceed to develop the theory of indefinite temporal constraint databases. In this respect our main contributions can be summarized as follows:

- We develop four first-order theories of time: (i) a first-order theory of constraints on time points over a discrete model of time, (ii) a first-order theory of constraints on time points over a dense model of time, (iii) a first-order theory of constraints on time points and intervals over a discrete model of time and (iv) a first-order theory of constraints on time points and intervals over a discrete model of time (chapter 3).

- We study the problems of quantifier elimination and decision for the above theories (chapter 6). Our results are original; with the exception of [Lad88] research on temporal constraints has so far concentrated on special classes of quantifier-free formulas [All83, VK86, VKvB89, vBC90, vB92, DMP91, KL91, Mei91a]. Because the first of the above theories is a subtheory of Presburger arithmetic and the second is a subtheory of real addition with order, our results must be of independent interest to the theoretical computer science community. Previous results on the same line of research include [FR74, FR75, FR79, FG77, RL78].

- Using the results on quantifier elimination and decision, we study the complexity of query evaluation in temporal constraint databases and indefinite temporal constraint databases with constraints from the above theories (chapter 7). We show that the complexity of query evaluation *remains the same* when we go from queries in relational calculus over relational databases, to queries in relational calculus with temporal constraints over temporal constraint databases. The same result is true when we consider indefinite information.

Among other things, we extend the survey of [Cho93] and answer all the posed questions which are relevant to our context.

Finally, we turn our attention to special classes of temporal constraints. Under the assumption that time is dense, we introduce a new tractable class of temporal constraints involving inequations and disjunctions of inequations. Traditionally inequations have received less attention than inequalities; perhaps because inequalities arise more often in applications. We were the first to point out that inequations deserve to be studied in their full generality [Kou92]. This dissertation deals with the problems of deciding *consistency*, computing the *minimal network*, performing *variable elimination* and enforcing *global consistency* for constraints in this class (chapters 8 and 9).

1.2 Organization of the Thesis

This dissertation is organized as follows. In chapter 2 we survey related work. In chapter 3 we present some background material on first-order logic, temporal constraint languages, and reasoning with temporal constraints. In chapters 4 and 5 we present the three database schemes: indefinite $\mathcal{L}$-constraint databases, $\mathcal{L}$-constraint databases and $\mathbf{M}_{\mathcal{L}}$-relational databases. In chapter 6 we study quantifier elimination and decision for theories of data and time. In chapter 7, we exploit the results obtained in chapter 6 for calculating precise bounds for query evaluation in temporal constraint databases and indefinite temporal constraint databases. In chapters 8 and 9 we study temporal constraints with disjunctions of inequations under the assumption that time is dense. Finally, in chapter 10 we present our conclusions and propose directions for future research.

Chapter 2

Related Work

In this chapter we present a survey of research which is related to the main part of this dissertation. We concentrate on the following three areas: temporal formalisms and systems developed in artificial intelligence and databases, constraint logic programming and constraint databases and relational database models for indefinite information. Our survey is by no means complete. For each of the above areas we discuss in detail only the proposals which we consider to be most relevant to this dissertation; the rest are simply mentioned briefly. The interested reader is referred to the published papers for more details.

This chapter is organized as follows. In section 2.1 we discuss several ontological questions which arise in any formal temporal representation. In section 2.2 we discuss representation and reasoning with temporal constraints and in section 2.3 we present some representative temporal logics. In section 2.4 we discuss the frame-problem and the related issues of persistence. In section 2.5 we discuss temporal database models. In sections 2.6 and 2.8 we discuss the problem of representing infinite and indefinite information in a database. Finally, in sections 2.7 and 2.9 we study several database models which can deal successfully with infinite and indefinite temporal information.

2.1 Ontology for Time and Change

In everyday life, most people are able to communicate their knowledge and understanding of temporal phenomena without any major difficulties. However, quite different intuitions surface as soon as people undertake to construct a formal temporal representation.

2.1.1 Change-Based vs. Time-Based Representations

There have been two approaches for representing temporal phenomena: *change-based* and *time-based* [SG88]. The change-based approach is based on the intuition that time is important *only* because the world around us changes constantly. If the world was static, there would probably be no reason for having a notion of time. In this approach, the entities which bring about change, called *change-indicators* in [SG88], are given primary consideration. For example, in situation calculus [MH69] the primitives are actions; states come into being only as a result of executing actions. In a similar way, events are the primitives in event calculus [KS86]; they start or end time periods during which relationships can be true in the world.

In the time-based approach, time is introduced explicitly. There is a set of times (the *time structure*) and change is manifested when propositions become true or false at different elements of this set. In this case, all the above types of actions and their effects can be represented. Once one adopts this approach, the time structure must be precisely defined. The relevant issues here are:

- What are the elements of the time structure? Points or intervals?
- Is time *linear, branching* or *cyclic*?
- Is time *discrete* or *dense*? The issue here is whether there exists a unit of time which cannot be decomposed. Discrete time is usually considered to be isomorphic to the integers. Proponents of dense time have a choice between rationals and reals.
- Is time *bounded* or *unbounded*? Time is unbounded when for every element of the time structure there is a "previous" and a "next" element.

The last three questions have not created much discussion in the published literature. Each researcher usually assumes whatever is appropriate for the intended application of his (or her) theory. However, the first of the above questions has generated much lively debate. In the remainder of this section, we review some of the relevant proposals.

2.1.2 Interval-Based Theories of Time

In [All81, All83], Allen presents a theory of time based on intervals. Intervals are proposed for capturing the intuitive notion of when some event occurs. They are things with duration contained in an unbounded single time line. Allen introduced thirteen mutually exclusive binary relations which can capture all the possible ways two intervals can be related. These relations are

$$Before, Meets, Overlaps, During, Starts, Finishes, Equals$$

and their inverses ($Equals$ is its own inverse). Whenever the temporal relation between two events is not known with certainty, a disjunction of some of the thirteen temporal relations can be used to represent what is known. For example, if T_1 is the time that John reads his morning newspaper and $T2$ is the time that he has breakfast, and we know that John never reads a newspaper while he is eating, then, in Allen's algebra, the formula

$$T_1 \ Before \ T_2 \ \vee \ T_1 \ Meets \ T_2 \ \vee \ T_1 \ After \ T_2 \ \vee \ T_1 \ MetBy \ T_2$$

characterizes T_1 and T_2 according to the information available.

Originally, points were not present in Allen's ontology. In subsequent papers [AH85, HA87, AH89a], Allen and Hayes presented a new theory of time which is a superset of the theory in [All83]. Their primitive notion is again that of a time interval, but now points are recognized as useful entities. Formally, time intervals are viewed as ordered pairs of points over an arbitrary unbounded linearly ordered set S. In other words $\langle a, b \rangle$ is a time interval iff $a, b \in S$ and $a < b$. No other commitment is made with respect to the structure of time.[1]

Two kinds of point-like entities are defined: *points* and *moments*. Points are introduced as the "meeting places" of intervals and are axiomatized appropriately. They are totally ordered

[1] Therefore, several different models of the theory are possible (e.g., time intervals can be interpreted as pairs of integers, or pairs of rationals or pairs of reals).

and have no duration. In sentences of natural language, they can be associated with the so-called accomplishment events like “I shot the gun”. Points can also be explicitly constructed from intervals: the beginning of an interval is defined as the non-empty set of all intervals which touch or include the beginning in any way; the ending of an interval is defined similarly.[2]

In contrast with points, moments are “very small time intervals”. They have duration but are not decomposable. Assume that an incandescent bulb burns out with a flush and during that time one is able to see a face on the window. This very short time interval which corresponds to a single very short event is an example of a moment. Moments can meet other intervals but they cannot overlap or contain other intervals.

[AH85] also showed that from the thirteen relations of [All83] only *Meets* should be introduced as a primitive relation; all the other relations can be defined from it.[3]

2.1.3 Point-Based Theories of Time

The principal intuitions guiding the work of Allen and Hayes come from time as reflected in sentences of natural language [AH89a]. A different tradition of work, influenced by elementary physics [McD82, Sho87b, Gal90], suggests that points should be the primitives in a temporal ontology. Researchers in temporal databases seem to have similar opinions [TCG$^+$93].

The arguments in this tradition are particularly simple and powerful. Points are important when one wants to model continuous change [McD82, Gal90]. For example, it is hard to imagine that the time spent by a ball, thrown vertically into the air, at the top of its trajectory can be represented by anything but a time point. If in some other cases intervals are more natural, they can be easily defined as ordered pairs of points [Sho87a, Lad87b]. In this way, it is irrelevant whether an interval includes its endpoints or not and the “truth at the endpoints” problem presented by Allen simply disappears. If intervals are introduced in this way, Allen's thirteen temporal relations can be defined in terms of the natural point relations $<$, $>$ and $=$.

Once one adopts an ontology and a structure for time, one usually turns to other, equally important, considerations. What are the kinds of temporal knowledge that must be represented? What is a good organization of temporal knowledge? What are efficient algorithms for answering queries in a temporal knowledge base? These questions are the topic of the next section.

2.2 Reasoning About Interval and Point Relations

Allen proposed to represent *qualitative* temporal information by qualitative interval constraint networks. A *qualitative interval constraint network* is a directed graph where nodes represents intervals and edges are labeled with vectors (i.e., disjunctions) of the thirteen binary qualitative interval relations presented in [All83].

Traditionally the following basic reasoning problems have been associated with constraint networks [Mon74]:

- Deciding whether a constraint network is *consistent* or *satisfiable*.
- Computing the minimal network equivalent to a given one. A network is called *minimal* if it subsumes all the networks having the same set of solutions.[4]In other words, in a minimal

[2] For a different construction see [Lad87d, Lad87a].

[3] Ladkin and Maddux showed that all the relations in this theory can also be defined using only one of Before, After, Overlaps, OverlappedBy, MetBy [LM88b].

[4] A network of interval relations is said to subsume another if they have the same number of nodes and every relation of the first subsumes the corresponding relation of the second.

network the label of each arc consists only of relations which can be part of a solution.

- Finding a *solution of the network*. If a network is consistent then we can label each arc with one of the possible relations and obtain a network which is also consistent. Each such labeling is a solution of the network.

[All83] presented a constraint propagation algorithm for computing the minimal network equivalent to a given network of interval relations. If any inconsistencies are discovered during constraint propagation, his algorithm reports unsatisfiability. This algorithm is similar to the *path consistency* algorithms for traditional constraint satisfaction problems [Mon74, Mac77, MF85]. It runs in $O(n^3)$ time (n is the number of intervals in the network) but unfortunately it is incomplete (i.e., it neither discovers all unsatisfiable networks nor it computes the minimal network corresponding to the network given). Vilain and Kautz demonstrated that this incompleteness should not surprise anyone. The problem of deciding whether a network of interval relations is satisfiable is NP-complete (as is computing the minimal network) [VK86].

Vilain and Kautz introduced the *point algebra* PA which allows one to relate two time points using the binary qualitative relations $<$, $>$ and $=$ and their disjunctive combinations [VK86]. Assertions in PA can also be represented in a graphical form by *qualitative point constraint networks*. Vilain and Kautz also claimed that Allen's constraint propagation algorithm compute the minimal point network for PA. Van Beek pointed out that this result is true only for the subset of PA which does not include the $\neq$ relation [vB89a]. He also gave a complete algorithm for the full PA [vB89a, VKvB89, vBC90, vB90b]. A large subset of Allen's interval relations can be represented in this point algebra by translating a relation between two intervals into a conjunction of point relations between the intervals' endpoints. Therefore one can get significant computational savings by restricting oneself to this subset of Allen's interval calculus.

Van Beek also discussed efficient algorithms for deciding consistency of PA networks (with or without the $\neq$ relation), computing the minimal PA network and finding a solution of a PA network or its corresponding interval network [vB89a, vBC90, vB90b, vB92]. These (and other) problems have also been discussed in by Ladkin and Maddux in [LM88a, LM88b, Lad87b, Lad87c, Lad87d, Lad87a, Lad86a, Lad86b]. Finally, Len Schubert and his colleagues have implemented really practical algorithms for reasoning in PA networks as part of their natural language understanding system [MS88a, MS88b, MS90, GS93, GSS93]. The work of [GA89] has also addressed similar implementation issues.

Dechter, Meiri and Pearl studied *quantitative* or *metric* temporal constraints of the form

$$a_1 \leq T_i - T_j \leq b_1 \ \vee \ \cdots \ \vee \ a_n \leq T_i - T_j \leq b_n$$

where T_i, T_j are real variables representing time points and $a_1, \ldots, a_n, b_1, \ldots, b_n$ are real numbers [DMP89, DMP91]. To deal with these constraints, [DMP89] introduced *quantitative point constraint networks* where nodes represent the given variables and arcs represent the given inequalities. Deciding consistency of of quantitative point constraint networks is also NP-complete. An important tractable subcase occurs when all constraints have only one disjunct i.e., they are of the form $a \leq T_i - T_j \leq b$. For this case deciding consistency and computing the minimal network can be done in $O(n^3)$ time (where n is the number of variables-points) by running any all-pairs-shortest path algorithm on an appropriate graph representation of the constraints [DMP89].

The results of [DMP89] have been extended by [KL91] and [Mei91a] to include qualitative constraints. [Mei91a] has also identified various classes of constraint networks with qualitative and quantitative constraints for which the above reasoning problems are tractable.

The above flurry of research activity resulted in several implemented systems [DM87, Koo89, GA89, KL91, Bod93, GSS93, SAD93]. [YA93] gives a comparative performance analysis for most of these systems.

Up to this point, we studied time and temporal inference in isolation. We will now turn our attention to formalisms and systems which combine a temporal dimension with a general framework for representing knowledge about a changing world.

2.3 Logics of Time and Change

In this section, we review a number of logics proposed by artificial intelligence researchers for reasoning about time and change. Initially we present a very influential change-based formalism: situation calculus. Then we turn our attention to various kinds of temporal logics.

2.3.1 Situation Calculus

In *situation calculus* the world is formalized in terms of states or situations, actions and fluents [MH69]. A *state* is a snapshot of the world at a given point in time. In each state, relationships in the world are represented by functions called *fluents*. The 2-place predicate $True$ or $Holds$ is introduced for stating what fluents are true in a given state.[5] The following is a situation calculus axiom for a typical blocks world:[6]

$$True(On(x,y),s) \implies \neg True(On(y,x),s)$$

The world changes when an *action* is performed. In situation calculus, actions transform a state into a new one. To formalize this, one introduces a special function *Result* for mapping a state and an action into the state that results from the performance of the action in the original state. For example, in the blocks world the effect of a $Move$ action can be captured by the following axiom:[7]

$$\begin{array}{l} True(On(x,y),s) \wedge True(Clear(x),s) \wedge True(Clear(z),s) \wedge x \neq z \implies \\ True(On(x,z), Result(Move(x,y,z),s)) \wedge True(Clear(y), Result(Move(x,y,z),s)) \end{array}$$

In its initial form, which appears in [MH69], situation calculus is inadequate for the formalization of a number of interesting cases: actions which take time, actions with delayed effects, concurrent actions and overlapping actions [SG88]. For extensions which address these inadequacies see [Web90, Sch90, GLR91, LS92, PR93a, JR93].

2.3.2 Temporal Logics

Depending on how time is introduced, [Rei89] proposes to distinguish three different kinds of temporal logics: first-order logics with temporal arguments [Hau87, Rei89, BTA89], reified temporal logics [All84, McD82, Sho87a] and modal temporal logics [Rei89, HS86]. Depending on whether formulas are interpreted over intervals or points, one can also distinguish between interval-based and point-based logics. Let us now consider some of the representative logics of each kind.

[5] It is customary to present situation calculus as a reified logic although [MH69] did not follow that route.

[6] In the rest of this section, I will write logical formulas according to the following syntactic conventions. The first letter of a predicate, function or constant symbol will always be an uppercase alphabetic character. The first letter of a variable symbol will always be a lowercase alphabetic character. Variables not explicitly quantified are assumed to be universally quantified at the beginning of the formula in which they appear.

[7] This is not the whole story. The above axiom says nothing about what does *not* change as a result of the $Move$ action. More on this, in section 2.4.

First Order Logics with Temporal Arguments

The simplest way of introducing time into a first order logic is the method of temporal arguments [Hau87, BTA89, Rei89]. In this approach, every function or predicate symbol in a first-order logic is augmented with one (or more) temporal arguments.[8] In this way, the interpretation of these symbols depends on time explicitly. For example, to express the fact that "the President of the U.S.A. in 1962 died in 1963" in the logic BTK of [BTA89], one can write the following formula:

$$Died(1963, President(1962, Usa))$$

Logics in this category are simple, easy to understand and can be implemented using standard theorem proving technology. It is also helpful to cast these logics as 2-sorted logical systems[9] in order to get the advantages associated with many-sorted logic [BTA89].

Allen's Interval-Based Temporal Logic

In [All84], Allen complements his theory of time with a reified interval-based temporal logic. He divides temporal propositions into *properties, processes* and *events*. Properties are used to capture static aspects of the world (e.g., "The house is painted red" or "The car is in the garage") while processes and events are used to capture dynamic aspects (e.g., "I am walking" or "I shot the gun"). More precisely, processes are used to model activities not involving a culmination or anticipated result like "I am walking". On the other hand, events denote activities involving a product or outcome like "I shot the gun".

Temporal propositions are denoted by terms. The 2-place predicate $Holds$ is used for talking about properties. $Holds(p,t)$ means that property p is true throughout time interval t. What distinguishes properties from other temporal propositions is *homogeneity*: whenever a property holds over an interval, it must hold throughout all its proper subintervals. In other words, properties are constrained by the following axiom:[10]

$$Holds(p,t) \iff (\forall t')(In(t',t) \implies Holds(p,t'))$$

Similarly, the 2-place predicates $Occur$ and $Occuring$ are introduced for talking about events and processes respectively. The difference between properties and events is illustrated by the following axioms:

$$Occur(e,t) \wedge In(t',t) \implies \neg Occur(e,t')$$
$$Occuring(p,t) \implies (\exists t')(In(t,t') \wedge Occuring(p,t'))$$

Informally, what distinguishes events from processes is that whenever an event occurs over an interval it does not occur over any of its proper subintervals.[11] On the other hand, processes occuring over an interval must also be occuring over at least one of its subintervals. For a critique of Allen's logic from the perspective of reasoning about continuous change see [Gal90].

McDermott's Temporal Logic

McDermott considers points (or states) as primitives in his logic and assumes that time is dense and branching into the future [McD82]. He introduces two kinds of temporal propositions: *facts*

[8] These arguments can be interpreted as points or intervals. Similarly, the time structure can have any of the properties presented in section 2.1.

[9] With one temporal and one non-temporal sort.

[10] $In(x,y)$ is equivalent to $During(x,y) \vee Starts(x,y) \vee Finishes(x,y)$.

[11] Therefore, we can count the times an event occured.

and *events*. The denotation of a fact is a set of time points while the denotation of an event is a set of maximal intervals. An interval is defined as a totally ordered, convex set of points. McDermott's notion of fact is the same as Allen's; however, McDermott does not insist that if an event happens over an interval, it does not happen over any of its subintervals.

The predicate T is introduced for talking about facts. $T(t,p)$ means that the fact p is true at time t.[12] The predicate TT (true throughout) is also introduced and defined in terms of T:

$$TT(t_1, t_2, p) \iff (\forall t)(t1 \leq t \leq t2 \implies T(t,p))$$

Similarly, a predicate OCC is introduced for stating that an event occurs.

The most important point in McDermott's work is his discussion of persistence. Most of his ideas were later on implemented in the TMM system [DM87] and will be presented in section 2.4.2.

Shoham's Interval-Based Temporal Logic

Shoham pointed out that the logics of Allen and McDermott lacked precise semantics. In response, he presented a first order temporal logic with a clean model theory [Sho87b]. In his logic the primitives are points; intervals are defined as pairs of points. Non-logical symbols are interpreted over time intervals. The interpretation of relation and function symbols is time-dependent but constants denote the same thing all the time. The following is a statement in Shoham's logic:

$$1867 \leq t \leq 1992 \implies True(t, t, Gender(PrimeMinister(Canada), Male))$$

Shoham rejects the categorizations of temporal propositions proposed by Allen and McDermott because they seem unnecessary at some times and insufficient at others. For example, if $Causes(A, B)$ means "A causes B", A and B will be facts or events so one actually needs a term for both kinds of propositions. Moreover, the proposed categorizations are insufficient to capture other properties of propositions. For example, some propositions such as "I ran more than two miles" have the property that if they hold throughout two overlapping intervals, they hold throughout the union of these intervals, but not necessarily vice versa [Sho87a]. Shoham allows a single class of propositions in his logic: *temporal propositions*. He also presents axioms which allow one to define categorizations which are finer-grained than these allowed by the logics of Allen and McDermott.

Modal Logics of Time

Modal logics of time, the so called *tense logics*, have been studied by philosophers for a long time [Pri57, Pri67, RU71]. Several artificial intelligence researchers [HS86, Rei89, RB89b] and most of the researchers in program verification [MP91] have also followed this approach. Here I will only discuss the logic of [Rei89].

In Reichgelt's modal temporal logic formulas are interpreted over time points. In addition to the classical tense logic operators P (past) and F (future), Reichgelt introduces an infinite number of operators $AT(t)$ where t is an element of the temporal universe. The AT operators can be used to express precise temporal information about a point in time. As an example, the sentence "On Tuesday, the letter had not still arrived" can be formally represented in this logic as follows:

$$AT(Tuesday)(F(Arrive(Letter)))$$

[12] Since I present a logic, I will avoid McDermott's "Cambridge Polish" notation.

Reichgelt's logic can cope with changing ontologies. This is achieved by augmenting each structure with a function which assigns to each point in time a subset of the domain of individuals (the set of individuals which exist at that time). In addition, the interpretation function is constrained to use only existent individuals for the denotations of the non-logical symbol at each point in time.[13]

Recently several modal temporal logics have been extended with the ability to formalize not only with qualitative but also with quantitative temporal information [AH89b, AH90, ACD93, RJC91]. This has been achieved by introducing new temporal operators like $\Box_{\leq t}$ (always within t time points) and $\Diamond_{\leq t}$ (eventually within t time points). These logics are particularly useful for the verification of real-time systems.

Discussion

If we were to use any of the logics of the previous section to represent a changing world, we would face the following problem. All these formalisms can only be used to explicitly specify the times events take place or properties hold in the world. From this specification one can infer when other events take place or when other properties hold. Although this kind of temporal capability is fundamental, it is not the only one needed in most situations. When dealing with a changing world, one is usually informed *only* about the events that take place and the relationships that become true or false as a result of these events. It is not usually specified what happens to the rest of the relationships in the world or for how long the new state of affairs will persist. From this limited amount of knowledge, people usually *assume* that all the relationships irrelevant to the announced event will not change their status in the new state of affairs, and that this new state will persist for some time. This amount of time can vary. For example, if I learn that someone was promoted to the rank of professor, I am ready to assume that this state of affairs will persist for a very long time. On the other hand, if I learn that someone was hired as an assistant professor, I will most likely assume that he or she will be an assistant professor for at most five years (then he or she will either be fired or promoted). This problem has been widely known as the frame problem and is the topic of the next section.

2.4 Reasoning About Persistence

The *frame problem* was originally introduced in the context of situation calculus [MH69]. This problem amounts to characterizing formally the aspects of a state that persist after an action is executed. With the same motivation, McDermott and Shoham [SM88] defined the more general *persistence problem*: reasoning, on the basis of information about the past, that a fact remains unchanged for an extended period of time.[14] Since these problems are so important in reasoning about a changing world, a temporal formalism or system must offer ways for dealing with them effectively.

An obvious solution to the frame problem in situation calculus is to augment our representation of the world with *frame axioms* which specify what does *not* change as a result of an event [MH69]. But if we model a world with a possible actions and r possible relationships then $O(ar)$ frame axioms are needed. Moreover, if we later on realize that something went wrong with this action

[13] One can deal with changing ontologies in any logic by introducing a 2-place predicate $Exists$ with the obvious denotation [Hau87]. Reichgelt disagrees with this option and argues that one should use the meta-language and not the object language to deal with ontological considerations [Rei89].

[14] Therefore, according to Shoham and McDermott, the frame problem is the persistence problem as it arises in situation calculus.

and we want to revise our beliefs, we have to *modify* some of our axioms. In conclusion, this solution does not seem satisfying.

How can we do better? Because the frame problem seems to involve the reasoning pattern "In the absence of information to the contrary, assume ...", one feels inclined to turn to non-monotonic reasoning [Rei87]. However, Hanks and McDermott demonstrated that a very simple example where the frame problem arises, the "Yale shooting problem" could actually give non-monotonic logics a lot of trouble [HM86]. The Yale shooting problem generated a lot of discussion in the artificial intelligence community. Many formalisms accounting for persistence were developed as a direct response to this problem; others were developed independently. Some of the the theoretical work resulted in implemented systems [DM87]. Due to the lack of space, we will only present here the event calculus [KS86] and the temporal reasoning system TMM [DM87]. The interested reader can find more on persistence and related problems in [Kau86, Sho88, Lif87, MS88c, GS88a, GS88b, BG89, Sha89, Sch90, Rei91, PR93a, JR93, PR93b].

2.4.1 The Event Calculus

Kowalski and Sergot proposed event calculus as a formalism for reasoning about events and persistence in a logic programming framework [KS86, Kow86]. Event calculus is a change-based formalism: change comes only in the form of an *event*. Whenever an event occurs, it initiates or terminates a time period throughout which an atomic fact is true. Events can be partially ordered and need not be associated with numerical times. Concurrent events and events with duration are allowed.[15] Whenever a fact is initiated, it is assumed to persist as much as possible into the future. When its exact termination time is later on determined, previous assumptions are withdrawn if they are inconsistent. *Negation as failure* is used for formalizing this non-monotonic behaviour.

In event calculus, every event is assigned a unique name and is represented by a collection of binary relationships similar to logic based representations of case semantics [CM85]. This notation allows for incomplete event descriptions and easy atomic updates. For example, the event "John is promoted to the rank of associate professor" is represented in the following way:

$$\begin{array}{l} Act(E1, Promote) \\ Object(E1, John) \\ Source(E1, Assistant) \\ Destination(E1, Associate) \end{array}$$

The functions $After(e, r)$ and $Before(e, r)$ are used to name the time periods which are started or ended by the event e and throughout which fact r is true. The 1-place predicate $Holds$ is used to assert that a certain fact is true for a given time period. For example, after event $E1$ the following fact is true:[16]

$$Holds(After(E1, Rank(John, Associate)))$$

In general,

$$\begin{array}{l} Holds(After(e, r)) \longleftarrow Initiates(e, r) \\ Holds(Before(e, r)) \longleftarrow Terminates(e, r) \end{array}$$

Domain specific facts can now be infered to hold by writing rules for $Initiates$ and $Terminates$. For the above example, the following rules are appropriate:

[15] However, no constructs are provided for relating these events.

[16] More clearly but less economically $Holds(After(E1, Rank(John, Associate)), Rank(John, Associate))$.

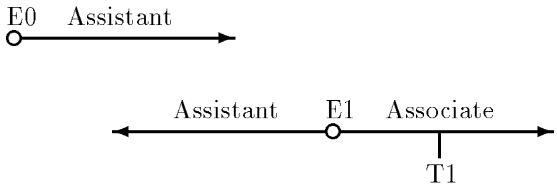

Figure 2.1: Events and time periods

$$Initiates(e, Rank(x, y)) \longleftarrow Act(e, Promote) \wedge Object(e, x) \wedge Destination(e, y)$$
$$Terminates(e, Rank(x, y)) \longleftarrow Act(e, Promote) \wedge Object(e, x) \wedge Source(e, y)$$

Facts are assumed to persist into the past and future as much as possible. As an example, assume that the events in Figure 2.1 have been observed. At first, John is hired as an assistant professor as a result of event $E0$. Later on, as a result of event $E1$, John is promoted to the rank of associate professor. Since we *cannot prove* that an event terminates the period throughout which John is an associate professor, we can assume by default that $Rank(John, Associate)$ is true for every time point following $E1$'s occurence. Using the negation as failure operator NOT the general rule can be posed as follows:

$$\begin{aligned}&HoldsAt(r, t_0) \longleftarrow \\ &\qquad Holds(After(e, r)) \wedge Time(e, t) \wedge t < t_0 \wedge \\ &\qquad NOT\ (\exists e', t')(Terminates(e', r) \wedge Time(e', t') \wedge t < t' \wedge t' \leq t_0)\end{aligned}$$

In the situation of Figure 2.1 this rule can be used to conclude $HoldsAt(Rank(John, Associate), T1)$.

Backward persistence is treated in a symmetric way. In the above situation, since we cannot prove that an event affecting John's rank has happened before $E1$ and after $E0$, we can conclude by default that $after(E0, Rank(John, Assistant)) = Before(E1, Rank(John, Associate))$ or that $after(E0, Rank(John, Assistant))$ is terminated before $Before(E1, Rank(John, Associate))$ is initiated.

[Sri88, Sri91, Sri89] extend event calculus to accomodate reasoning not only about the history of an aplication domain, but also about the history of the system's *beliefs* about of this domain. The functionality of the resulting calculus is that of a *bitemporal deductive database* (see section 2.5).

2.4.2 The Time Map Management System

The Time Map Management system (TMM) is an implementation of a substantial part of McDermott's temporal logic [DM87]. The TMM manages a collection of time points related by metric constraints on their distance. Points correspond to the beginnings and endings of events and persistences. *Events* have already been defined in section 2.3.2: they are occurences whose duration can be predicted with precision. *Persistences* correspond to facts: one can assert that they become true without specifying explicitly for how long they will remain true. By default, persistences remain true until they are *clipped* by a contradictory persistence.[17] In the TMM, the

[17] In [McD82], upper bounds on the lifetimes of persistences can be specified. This feature has not been implemented in the TMM.

term *time token* is used to refer to an interval together with an event or persistence happening throughout that interval.

The most significant feature of the TMM is its *temporal reason maintenance facilities*. The TMM detects apparent contradictions between time tokens according to the following rule of persistence: "If one token is determined to be in contradiction with a second, and the first is constrained to begin before the second begins, then the first is constrained to end before the second" [DM87].[18] A simple reason maintenance facility [Doy79] can be used to implement this rule. Whenever a token T_1 is asserted, the TMM finds all the tokens T_2 that could possibly contradict it and sets up "clipping constraints" (e.g., $(< \ (end\ T_1)\ (begin\ T_2))$) with appropriate justifications (e.g., $(< \ (begin\ T_1)\ (begin\ T_2))$) for them. The underlying reason maintenance system makes sure that as soon as the justifications are believed, the clipping constraints are activated.

The TMM also supports three kinds of temporal forward chaining: temporal forward chaining rules, auto-projection rules and controlled forward inference. Temporal forward chaining rules and auto-projection rules can be used to model the "physics" of the situation represented in the TMM. Temporal forward chaining rules are of the following form:

$$(\rightarrow_t \ (and\ P_1 \dots P_n)\ Q)$$

Their meaning is obvious: whenever $P_1, \dots, P_n$ are true simultaneously, Q is also true. Auto-projection rules are somewhat different:

$$(project\ AntecedentCondition\ TriggerEvent\ Delay\ ConsequentEffect\ Duration)$$

A rule of this form asserts that whenever an event of type $TriggerEvent$ occurs, and the condition $AntecedentCondition$ is true throughout the interval associated with the triggering event, then a token of type $ConsequentEffect$ is created. This token follows the triggering event by an interval $Delay$ and lasts for a length of time determined by $Duration$. When events are partially ordered computing the consequences of rules like the above is very hard. More formally, when events are partially ordered, determining whether a fact P holds immediately after an event E in some (or in every) totally ordered extension is NP-hard [DB88].

The last temporal reason maintenance facility offered by the TMM is *controlled forward inference*. The need for this facility arises when an application program (e.g., a planner [Dea88, DFM88]) poses a query to the TMM and then, depending on the answer, it asserts the existence of some new time tokens. In these cases, the TMM keeps track of the answers to the query and the assertions made by the program. When an answer to such a query is no longer warranted (remember, temporal information in the TMM is defeasible), the TMM informs the application program about the situation. For more recent work on the TMM implementation see [Bod93, SBC92].

This section finishes our survey of work on temporal reasoning carried out by artificial intelligence researchers. We will now turn to related developments in the artificial intelligence community.

2.5 Data Models with a Temporal Dimension

The database community has reacted positively to the challenge of extending existing data models with temporal capabilities. A lot of work has already been done in this area. Due to space

[18] This kind of reasoning is the same as the simple case of event calculus studied in [Kow86].

considerations, this and the following sections summarize only a few of the important questions tackled by database researchers. For a complete up-to-date anthology of the area see [TCG+93].

[SA85] proposed the following classification of DBMS depending on their support for time:

- *Snapshot databases.* All conventional databases e.g., INGRES [SWKH76] belong to this category. The information in these databases corresponds to a snapshot of the world at a particular point in time. Updates are destructive and old information is not retained. The only kind of temporal information supported by such systems is *user-defined time* (e.g., the date of birth of a person can be stored as a string).

- *Rollback databases.* Rollback databases maintain a complete record of the evolution of the database. Intuitively, they can be understood as sequences of snapshot databases indexed by *transaction time* [SA85] or *belief time* [MBJK90, Sri88] i.e., the time a transaction changing the current database state committed (or the time the system starts believing the new information about the world). In rollback databases, updates are performed with respect to the current state of the database while queries can inquire about past states as well. POSTGRES [SRH90] is an example of a rollback database system.

- *Historical databases.* Rollback databases keep a record of the changes as registered in the database. Historical databases, on the other hand, keep a record of the changes as they take place in the real world. Intuitively, they can be understood as sequences of snapshot databases indexed by *history time* [MBJK90] or *valid time* [SA85] i.e., the time a piece of information is true in the real world. In historical databases, the users can pose queries to find out about the world at any point in time. Any errors about the past can be corrected but no record of past database states is kept.

- *Bitemporal databases.* Bitemporal databases combine the advantages of rollback and historical databases. Intuitively, they are sequences of historical databases indexed by transaction time. The user is allowed to inquire about the world at some point in time according to the information present in the database at some other point time. The user is also allowed to modify the current state of the database in order to add, delete, or modify historical information about the world. The prototype presented in [AS86] is an example of a bitemporal database management system.

Adding time to a data model is a challenging job and it raises many interesting questions. The following is a list of the most important ones:

- What is the static model which will be extended? So far, most of the work has concentrated on the relational data model [CW83, CT85, CC87, Gad86, Gad88, GV85, Sno87, MS87, GY88, GB89, LJ88a, LJ88b, SRH90, KSW90]; some researchers have considered extending object-oriented database models [Klo81, DBG+85, MBJK90, EWK93] or deductive database models [CI88, CI89, Cho90b, Cho90a, MPB92, Sri91]. [Wuu92, Wuu93] have chosen to use the available facilities of an object-oriented model for modeling time. The advantage of this work is that no special constructs are needed in the query language.

- What are the dimensions of time that will be supported? [CW83, CT85, CC87, Gad86, Gad88, GV85, GY88, LJ88a, LJ88b, KSW90, EWK93, Cho90a] have considered only valid time while [SRH90, LS93b, LS93a] consider only transaction time. [Sno87, MS87, GB89, MBJK90, MPB92, Sri91] considered both valid time and transaction time.

- What is an appropriate ontology and structure for time? In contrast with artificial intelligence formalisms temporal database models adopt very similar notions of time. Most researchers seem to agree that a linearly ordered discrete temporal universe is enough for their applications.[19] The elements of the discrete universe are *points* or line segments called *chronons* [CR88]. Whenever intervals are needed they are usually defined by pairs of points or chronons.

- Is the integration of time into the static model tight or loose? Some temporal models are *consistent extensions* of static models i.e., every construct or operation of the static model has a corresponding construct or operation in the temporal model [Gad88, CC87]. Other models simply introduce concepts with temporal semantics which can be manipulated using constructs and operations of the static model [DBG+85].

- What are efficient ways for storing and manipulating information in the new model? Enough work has already been done on this area but clearly much remains to be done. The interested reader is referred to [TCG+93, part III] for a discussion of the solved problems and the open issues.

- Is *infinite* or *indefinite* temporal information allowed in the new model? Until very recently these kinds of temporal data had been ignored by database researchers. [CI88, CI89, Cho90b, Cho90a, KSW90, BNW91, BCW93] have studied *infinitely periodic* temporal data while [Cha88, MBJK90, GNP92, DS93, vdM92, Kou93a] have concentrated on indefinite temporal data.

- What is the expressive power of the new model and its query language? What is the complexity of query processing in the new model? Up to this point these questions have not received the attention they deserve from the database community. For historical data models the first question has been studied in [CCT93]. In the context of historical data models with periodic data these questions have been addressed in [BCW93].

In the rest of this section we will discuss the work of Snodgrass [Sno87, Sno93] which is probably the most infuential attempt towards the development of a bitemporal database system.[20]

2.5.1 A Data Model and Query Language For Bitemporal Databases

[Sno87, Sno93] present a bitemporal relational model and its query language TQUEL. TQUEL is an extension of QUEL [HSW75] for bitemporal databases. In addition to standard (static) relations, the data model of [Sno87] offers event and interval relations. *Event relations* are intended for the representation of *instantaneous* events i.e., events whose time of occurence is specified by a point or a chronon. *Interval relations* can be used to represent facts that are true for arbitrary periods of time. In an event relation tuples are time-stamped with a *single valid time* denoting when the modeled event occured. In an interval relation tuples are time-stamped with a *pair of valid times* denoting the endpoints of the interval for which the modeled fact is true. In both kinds of relations tuples are optionally time-stamped with two *transaction times* denoting when the tuple was logically inserted or deleted in the relation.

The following relation EMP is an example of a bitemporal interval relation.

[19] There are exceptions to the rule: [CW83] uses continuous time; [Gad88] can handle both discrete and continuous time.

[20] [MBJK90] and [Sri88] (see section 2.4.1) are other proposals with similar goals.

NAME	DEPT	Valid Time		Transaction Time	
		From	To	Start	Stop
John	Toys	1-1-81	∞	1-2-81	∞
Tom	Shoes	1-6-82	∞	1-6-82	∞
Mary	Toys	1-3-81	∞	1-3-81	∞
Tina	Toys	1-5-81	∞	1-5-81	∞

The notation *day-month-year* is used for representing times in the Gregorian Calendar. The granularity of a chronon is one day. The special value ∞ is used to achieve a simple form of persistence. The same method has been used by virtually *all* database models.[21]

The following are some queries and updates which demonstrate the capabilities of TQUEL.

Example 2.5.1 The query "Who were the employees in Toys when Tina joined the department" can be expressed as follows:

```
range of E is EMP
range of E1 is EMP
retrieve into NEW (NAME=E1.NAME)
   where E.NAME=Tina and E1.DEPT=Toys
   when E1 overlap begin of E
```

The `when` clause is similar to the QUEL `where` clause but refers to the valid time of the tuples. Since no `as of` clause is present, this query is executed against the current state of the database. Its answer is a relation with a single column and tuples `(John)` and `(Mary)`.

Example 2.5.2 Assume that John joined Shoes on March 1st, 1985 and the following update is executed a month later.

```
range of E is EMP
delete E
   where E.NAME=John and E.DEPT=Toys
   valid from ''1-3-85''

append to EMP (NAME=John, DEPT=Shoes)
   valid from ''1-3-85''
```

The relation EMP would then look as follows:

NAME	DEPT	Valid Time		Transaction Time	
		From	To	Start	Stop
John	Toys	1-1-81	∞	1-2-81	1-4-85
Tom	Shoes	1-6-82	∞	1-6-82	∞
Mary	Toys	1-3-81	∞	1-3-81	∞
Tina	Toys	1-5-81	∞	1-5-81	∞
John	Toys	1-1-81	1-3-85	1-4-85	∞
John	Shoes	1-3-85	∞	1-4-85	∞

Let *now* stand for the commit time of the transaction executing the delete statement. In general, a `delete` statement has the following effect:

[21] For a non-monotonic logic approach in a bitemporal deductive database context see [Sri88].

- It changes some transaction stop times from ∞ to *now*.
- It adds some tuples with a transaction start time of *now* whenever the information to be deleted is a restriction of some existing tuple.

In other words, although no tuples are deleted from the database some transaction times are modified.

The `append` statement simply records John's new assignment. The `valid` clause is used to specify the valid time of the deleted or appended tuples. Alternatively, the update could have been specified using the command `replace`:

```
range of E is EMP
replace E (DEPT=Shoes)
    where E.NAME=John and E.DEPT=Toys
    valid from ''1-3-85''
```

Example 2.5.3 The query "Output the names of the employess in Shoes together with the time they worked there, according to the state of the database at the end of 1985" can be expressed as follows:

```
range of E is MGR
retrieve into NEW1(NAME=E.NAME)
    valid from begin of E to end of E
    where E.DEPT=Shoes
    as of end of ''1985''
```

The `as of` clause intuitively rollbacks the database to the state it was at the end of 1985 and then answers the query against that state. The answer to this query contains tuples for John and Tom. The answer to this query should be compared with the answer to the following query (which contains only a tuple for Tom).

```
range of E is MGR
retrieve into NEW2(NAME=E.NAME)
    valid from begin of E to end of E
    where E.DEPT=Toys
    as of begin of ''1985''
```

Example 2.5.4 Now assume that we just realized that Mary actually joined the company a month later than what we originally thought, and actually works for Shoes. The following `replace` statement will correct the error.

```
range of E is EMP
replace E (DEPT=Shoes)
    where E.NAME=Mary and E.DEPT=Toys
    valid from ''1-4-81''
```

If the corresponding transaction committed at "1-5-85", then the updated relation is as follows.

NAME	DEPT	Valid Time		Transaction Time	
		From	To	Start	Stop
John	Toys	1-1-81	∞	1-2-81	1-4-85
Tom	Shoes	1-6-82	∞	1-6-82	∞
Mary	Toys	1-3-81	∞	1-3-81	1-5-85
Tina	Toys	1-5-81	∞	1-5-81	∞
John	Toys	1-1-81	1-3-85	1-4-85	∞
John	Shoes	1-3-85	∞	1-4-85	∞
Mary	Shoes	1-4-81	∞	1-5-85	∞

For related issues concerning bitemporal data models see [MS87, GB89].

Most of the data models discussed in this section can only represent definite and finite temporal information. In sections 2.7 and 2.9 we discuss several proposals for representing infinite or indefinite temporal information. To set the scene for this discussion, we first discuss the problem of representing infinite or indefinite information in a more general context.

2.6 Representing Infinite Information Using Constraints

A very natural way of representing infinite information by a finite expression is by using *constraints*. For example the equation $x^2 = y$, where x and y are variables ranging over the real numbers, is a *finite representation* of the infinite set of pairs of real numbers (x, y) such that the square of x is y. Recently this idea has been considered very seriously in the area of constraint logic programming.

Constraint logic programming is an extension of logic programming with the goal of performing computation over an arbitrary *domain* (not necessarily the Herbrand universe) and replacing unification with the more powerful operation of *constraint solving* over this domain. Under this paradigm many constraint logic programming languages have been developed over the last few years e.g., CLP($\mathcal{R}$) [JMSY92], Prolog III [Col90], CHIP [vH89] and others. The following is a CLP($\mathcal{R}$) program dealing with mortgages [JMSY92]:

```
mortgage(P, Time, Interest, MP, B):-
        Time > 0,
        Time <= 1,
        Interest = Time * (P * I/100),
        B = P + Interest - (Time * MP).
mortgage(P, Time, Interest, MP, B):-
        Time > 1,
        Interest = P * I/1200,
        mortgage(P + Interest - MP, Time - 1, I, MP, B).
```

The details of the program are not difficult to understand given that `P` is the principal sum, `Time` is the life of the mortgage in months, `Interest` is the annual interest rate (compounded monthly), `MP` is the monthly payment and `B` is the outstanding balance. If we want to know how much it would cost to finance a \$100,000 mortgage with interest rate 12% for 30 years, we can ask the query `?- mortgage(100000, 360, 12, MP, 0)`. The answer to this query will be `MP=1028.61`. But the interesting feature of CLP($\mathcal{R}$) is that it allows us to ask queries that can have *constraints* as answers. For example, let us consider the query `?- mortgage(P, 120, 12, MP, B)` which asks for the *relation* between `P`, `MP` and `B` given the values for the rest of the parameters. This answer to this query is

$$\mathtt{B} = 0.302995 * \mathtt{P} - 69.700522 * \mathtt{MP}$$

which is a finite representation of the infinite set of triples which are solutions of the above equation.

Recently there have been attempts to apply the same ideas in the context of relational and deductive databases [KKR90, KSW90, Rev90, Sri92, JBM93, Kou93a]. The more general discussion of the issues involved in this exercise appears in [KKR90] where Kanellakis, Kuper and Revesz propose a *constraint database* scheme based on relational databases. In this scheme a tuple is a quantifier-free conjunction of constraints over a structure $\mathcal{M}$. A relation is a finite set of tuples i.e., a disjunction of conjunctions of constraints. The query language for a constraint database is relational calculus with constraints over $\mathcal{M}$. Finally, the semantics of query evaluation can be defined using quantifier elimination in the theory of the structure $\mathcal{M}$.

As an example, let us consider the following constraint relation RECTANGLE which stores data about rectangles. The first tuple of this relation means that R_1 is a rectangle which consists of the infinite set of points (x, y) such that $1 \leq x \leq 5 \ \wedge \ 1 \leq y \leq 8$ is true. In this example we have two kinds of constraints: *equality constraints* and *rational order constraints.*

RECTANGLE

ID	X_C	Y_C	CON
n	x	y	$n = R_1 \ \wedge \ 1 \leq x \leq 5 \ \wedge \ 1 \leq y \leq 8$
n	x	y	$n = R_2 \ \wedge \ 3 \leq x \leq 8 \ \wedge \ 3 \leq y \leq 10$
n	x	y	$n = R_3 \ \wedge \ 20 \leq x \leq 25 \ \wedge \ 15 \leq y \leq 18$

In this example the query language is relational calculus with equality and rational order constraints. The following query finds all pairs of intersecting rectangles:

$$\{(n_1, n_2) | \ n_1 \neq n_2 \ \wedge \ (\exists x, y)(RECTANGLE(n_1, x, y) \ \wedge \ RECTANGLE(n_2, x, y))\}.$$

The following important questions arise in this constraint database scheme:

1. Are queries evaluable *bottom-up in closed form?* This is the basic property of query evaluation in relational databases. It follows from Codd's theorem on the equivalence of relational calculus and algebra.

2. What is the complexity of query evaluation?[22] The data complexity of relational and Datalog queries is low (i.e., LOGSPACE and PTIME respectively).

Both of these questions have been answered positively for certain classes of constraints and queries considered in [KKR90]. For example, the above query can be evaluated bottom-up in closed form by substituting the definitions of relation RECTANGLE in the query and then performing *quantifier elimination.* The result of the elimination will be the quantifier-free formula

$$(n_1 = R_1 \ \wedge \ n_2 = R_2) \ \vee \ (n_1 = R_2 \ \wedge \ n_2 = R_1)$$

which gives the answer to the query.

[KKR90] considered several classes of constraints and obtained the following results:

- *Polynomial inequality constraints over the reals.* Relational calculus with polynomial inequality constraints can be evaluated bottom-up in closed form with NC data complexity.

[22] The complexity of query evaluation is usually measured using the notions of data complexity and combined complexity introduced in [Var82, CH82]. When we consider *combined complexity*, we measure the complexity of query evaluation as a function of the size of the database, the size of the schema and the size of the query. When we consider *data complexity*, we measure the complexity of query evaluation as a function of the database size only; the query and the schema are considered *fixed*.

- *Dense linear order constraints.* Relational calculus with dense linear order constraints can be evaluated bottom-up in closed form with LOGSPACE data complexity. For the same class of constraints, Datalog and inflationary Datalog with negation can be evaluated bottom-up in closed form with PTIME data complexity.

- *Equality constraints over an infinite domain.* Relational calculus with equality constraints over an infinite domain can be evaluated bottom-up in closed form with LOGSPACE data complexity. For the same class of constraints, Datalog and inflationary Datalog with negation can be evaluated bottom-up in closed form with PTIME data complexity.

- *Boolean equality constraints.* Datalog with boolean equality constraints can be evaluated bottom-up in closed form with Π_2^p-hard data complexity (i.e., the data complexity in this case is rather high).

In the same spirit [Rev90] considered Datalog with integer gap-order constraints (denoted by $Datalog^{<_{\mathbb{Z}}}$). A *gap-order constraint* is a constraint of any of the forms

$$x = c,\ x = y,\ x < c,\ c < x,\ x - y < g$$

where x, y are variables ranging over $\mathbb{Z}$, $c \in \mathbb{Z}$, $g \in \mathbb{Z}$ and $g \geq 0$. Revesz showed that $Datalog^{<_{\mathbb{Z}}}$ queries over integer gap-order databases can be evaluated bottom-up in closed form. In addition recognizing whether a certain tuple is in the answer to a query can be done with PTIME data complexity. Independently this problem has also been solved in [CM93]. [CM93] have also considered gap-order constraints over a dense domain and study *combined complexity* of query evaluation. The only difference with [Rev90] is that [CM93] consider combined complexity of query evaluation which is EXPTIME.

The previous papers adopt a rather theoretical perspective. On a more practical note [JBM93] study query evaluation in relational databases with linear inequality constraints over the reals. The authors discuss the similarities and differences from query evaluation in a standard relational environment and propose a new methodology for query optimization.

One of the problems of query evaluation in constraint databases is determining whether a newly derived tuple is subsumed by the tuples already in the database. This problem is essentially the *duplicate elimination* problem for constraint databases and it is harder than the corresponding problem in standard relational databases. In [Sri92] Srivastava showed that this problem is coNP-complete even for very simple inequality constraints over the reals.[23] He also presented a divide-and-conquer algorithm for this problem in the case of linear inequality constraints over the reals, and discussed indexing of constraint relations.

Let us now turn to the special case of infinite temporal information.

2.7 Infinite Temporal Information

We have *infinite temporal information* when an infinite number of times is associated with an event or fact. The following are two typical examples of infinite temporal information:

- Events that happen periodically and there is no a priori bound on the number of times they will occur. As an example, consider the sentence “The committee will meet every Monday starting today”.

[23] This result also follows from [Son85].

- Facts that are true over an infinite number of points or intervals. Even the ubiquitous fact "John's salary was \$50,000 from January 1, 1990 to December 31, 1992" falls under this category since we have a dense model of time!

The models studied so far represent temporal data by *explicitly* listing all the different values of attributes over time. As a result, the above kinds of infinite temporal data either cannot be represented at all in these models or can be represented in a very poor way.

Let us now consider the following piece of *finite* temporal information: "For the next 24-hour shift Shaky the robot will work at the second lathe. He will start work at hour 2 and have one hour break every two hours". The bitemporal model of the previous sections needs seven tuples for representing this information. Clearly a more economical representation would be preferable in this case. This section presents two data models which deal explicitly with these problems.

Linear Repeating Points and Constraints

[KSW90] proposed a *generalized relational model* for representing temporal data. The distiguishing feature of this approach is that finite and infinite temporal data can be represented in a very natural way. The basic construct in this model is the linear repeating point. Given $c, p \in \mathbb{Z}$, a *linear repeating point* is a set $\{c+pn \mid n \in \mathbb{Z}\}$. If T is a set of linear repeating points and D a set of non-temporal data values then a *generalized tuple* is an element of $D^k \times T^k$ possibly constrained by a conjunction of inequalities of the form

$$x_i \leq x_j + a,\ x_i = x_j + a,\ x_i \leq a,\ x_i \geq a \text{ or } x_i = a$$

where $1 \leq i,\ j \leq k$ and $a \in \mathbb{Z}$. A *generalized relation* is a finite set of generalized tuples of the same schema. An example of a generalized relation, representing the assignments of robots to machines from time T_1 to T_2, is given below [KSW90].

ROBOT	MACHINE	T_1	T_2	CON
Shaky	Lathe1	0	1	—
Shaky	Lathe2	$2+3n_1$	$4+3n_2$	$T_1 = T_2 - 2 \wedge T_1 \geq 2 \wedge T_2 \leq 24$
Breaky	Lathe1	$6+10n_1$	$7+10n_2$	$T_1 = T_2 - 1 \wedge T_1 \geq 10$
Breaky	Lathe2	$10n_1$	$3+10n_2$	$T_1 = T_2 - 3$

[KSW90] define *generalized relational algebra* operations and show that they can be computed with polynomial time *data complexity*. However, a significant problem lurks in the background. The database has to be "normalized" before the operation of projection can be computed. This normalization process can increase the size of a relation considerably. [NS92] discuss some practical alternatives for implementing generalized relational databases. Finally, in [BNW91] the generalized relational model has been extended with a deductive layer.

The Language $Datalog_{1S}$

Chomicki and Imielinski proposed the language $Datalog_{1S}$ for modeling infinite temporal phenomena [CI88, CI89, Cho90b, Cho90a]. The following is an example of a $Datalog_{1S}$ program which can be used to keep track of the meetings of some professor with his (or her) students:

$meeting(0, john)$
$meeting(1, mary)$

$next(john, mary)$
$next(mary, john)$

$meeting(t+1, x) :-$
$\quad meeting(t, y),$
$\quad next(y, x)$

$Datalog_{1S}$ programs allow two kinds of predicates: temporal (e.g., *meeting*) and non-temporal (e.g., *next*). The first argument of a temporal predicate is always a temporal term. A temporal term can be any term constructed from the temporal constant 0, temporal variables and the temporal function symbol +1.[24] In other words, temporal terms are in a one to one correspondence with the natural numbers. The rest of the arguments in temporal and non-temporal predicates must be constants or variables.

Although least models of $Datalog_{1S}$ programs can be infinite, they admit equivalent finite representations because they exhibit a repetitive periodic structure. Let us assume that c is the largest temporal term in the base predicates of a $Datalog_{1S}$ program. A model M of this program is called *periodic* with period (b, p) if and only if for all t such that $t \geq c + b$ it is true that $M[t] = M[t + p]$.[25]

In [Cho90a], *relational specifications* are introduced for finitely representing periodic models. Relational specifications are triples $(\mathcal{T}, \mathcal{B}, \mathcal{R})$ where $\mathcal{T}$ is the set of representative ground terms, $\mathcal{B}$ is the primary database and $\mathcal{R}$ is a rewrite rule of the form $c + b + p \rightarrow c + b$ (where (b, p) is the period of the model). A relational specification for the above $Datalog_{1S}$ program is as follows:

$\mathcal{T} = \{0, 1\}$
$\mathcal{B} = \{\ meeting(0, john),$
$\quad meeting(1, mary),$
$\quad next(john, mary),$
$\quad next(mary, john)\ \}$
$\mathcal{R} :\ 2 \rightarrow 0$

Relational specifications can be used to provide finite representations of infinite answers to queries. If the query is ground (e.g., $meeting(4, john)$?), the temporal term is at first rewritten using the rule R (e.g., $meeting(0, john)$?) and then the result is checked for membership in B. For open queries (e.g., $meeting(X, mary)$?) a temporal term (e.g., 1) and the rule R is returned.

Unfortunately, general $Datalog_{1S}$ programs are rather impractical: yes/no query processing has PSPACE-complete data complexity. A similar result is true for constructing a relational specification. In [Cho90b], classes of temporal rules for which these problems can be solved in polynomial time are identified.

With very similar problems in mind [AM89, Bau89a, Bau89b] proposed a logic programming language with *modal* temporal operators called *Templog*. The interested reader can consult [BCW93] for a thorough discussion of the expressive power and complexity of various languages for infinite periodic data. Other logic programming languages with modal operators have been presented in [RB89a, Gab89a, Gab89b, OW92, Brz91, Brz93a].

2.8 Indefinite Information in Relational Databases

There have been many attempts to extend the relational model to deal with *indefinite* or *incomplete* information. This is quite natural because many real-world applications sometimes involve

[24] The term 1 in $meets(1, mary)$ should therefore be understood as $0 + 1$.
[25] For each temporal term t_0, $M[t0] = \{B : (\exists P)(\exists \overline{x}) B = P(\overline{x}) \ \wedge\ P(t_0, \overline{x}) \in M\}$.

indefinite information. Codd himself initiated the study of null values in relational databases [Cod75, Cod79]. *Null values* are place-holders for pieces of missing information. Three kinds of null-values have been studied in the literature: (i) value exists but is at the present unknown (ii) value does not exist, and (iii) value is not applicable. The interested reader can find more details in [Vas79, Lip79, Vas80, Lip81, Bis81, Bis83, IL84, Zan84, DdC88, Imi89, Gra89]. We will only discuss the proposal of [IL84] which we consider to be the most satisfactory. In a similar spirit there have been several attempts to introduce disjunctive information in relational and deductive databases [LMR92] and object-oriented databases [IV89, INV91, INV91, Vad92].

Imielinski and Lipski [IL84] proposed conditional tables as a syntactic representation of indefinite information in relational databases. A *conditional table* is similar to a relation but it can also have *variables* as attribute values. These variables are *marked null values*: they represent values which *exist* but are not known precisely. [Gra89] also proposed to allow the specification of a *global condition* on these variables. The global condition is a quantifier-free formula in the first-order theory of equality over the database domain. The following is an example of a conditional table and the associated global condition:

EMP

NAME	DEPT	CON
John	ω_1	*true*
Tom	ω_2	*true*
Mary	Toys	*true*
Tina	Shoes	*true*

$$G: \ \omega_1 \neq \omega_2 \ \wedge \ (\omega_1 = \text{Toys} \ \vee \ \omega_2 = \text{Shoes})$$

Tables are syntactic devices for the representation of indefinite information. Each table is semantically equivalent to a set of relations which capture all the possible ways the domain of application could have been according to our indefinite knowledge. The above table gives data about employees and their departments. We have indefinite information about the departments of John and Tom. *All we know* about these departments is stated explicitly in the global condition G:

- John and Tom do not work in the same department.
- Tom works either in Toys or in Shoes.

The reader should realize that the above variables are *totally different* than the variables found in constraint databases (see section 2.6).

Let us observe that each tuple of a table has a *local condition* associated with it. In most practical cases the local condition will be *true* for all tuples in relations in the original database. However local conditions are useful for computing answers to queries or views. For example, let us consider the query "Find all employees that work in Toys". Using the algebraic query language of [IL84] this query can be expressed as follows:

$$\pi_{\text{NAME}}(\sigma_{\text{DEPT}=\text{Toys}}(\text{EMP})).$$

The following table is the result of this operation.

NAME	CON
John	$\omega_1 = Toys$
Tom	$\omega_2 = Toys$
Mary	*true*

The meaning of the above table is the following. Mary is certainly an employee of Toys. John and Tom are employees of Toys if and only if ω_1 and ω_2 is equal to Toys.

The problem of indefinite information has also been studied, with very similar results, by Reiter and Vardi using a *proof-theoretic* perspective of relational databases [Rei84, Rei86, Var86]. In this context null values are represented by existentially quantified variables or Skolem constants.

[Var86, AKG91] have studied the computational complexity of query answering in relational databases with indefinite information. Vardi adopts a proof-theoretic account of relational databases but his *logical database model* does not differ significantly from the model of conditional tables. [Var86] considers the problem of deciding whether a yes/no relational calculus query is true in *all* possible worlds represented by a logical database. He shows that this problem has co-NP-complete data complexity and PSPACE-complete combined complexity. If we consider relational calculus queries with k (fixed) alternations of quantifiers then the combined complexity of the same problem becomes Π^p_{k+1}-complete. [AKG91] have studied the same problem and tightened the lower bound of [Var86]. [AKG91] also studied the complementary problem of deciding whether a yes/no relational calculus query is true in *at least one* of the possible worlds represented by a conditional table. The data complexity of this problem is NP-complete for arbitrary relational calculus queries and PTIME for positive existential queries. Several other decision problems related to conditional tables are studied in [Gra89, AKG91].

The above results demonstrate that for several kinds of queries over relational databases with indefinite information, query answering is intractable. This is probably one of the reasons that there are no commercial strength implementations of relational databases with indefinite information.

Let us now consider the special case of indefinite temporal information.

2.9 Indefinite Temporal Information

We have *definite temporal information* when all times associated with events or facts are known to be *equal to* an *absolute time* i.e., a point or interval on the time line. In other words, all times associated with events or facts are known to full precision in the desired level of granularity. The following are some examples of definite temporal information with respect to various time lines:

- "The car was on service throughout March 25th, 1993" (with respect to the time line of the Gregorian calendar).
- "The car was on service for the chronon 25-3-1993" (with respect to the time line used in the examples of section 2.5.1).

Most temporal database models proposed to date can handle only definite temporal information [TCG+93]. However, if one considers advanced applications like planning [AKP91], scheduling [Fox87], project management [SFG85] or medical information systems [Dut88], one will soon realize that indefinite temporal information plays an important role as well.

We have *indefinite temporal information* when the time associated with an event or fact is either unknown or has not been fully specified. The time associated to an event or fact can be under-specified in various ways:[26]

- The time associated with an event or fact might be specified via a *qualitative relationship* (different than equality) to some absolute time. As an example, consider the sentence "John became manager *after* March, 1993".

[26] A similar discussion appears in [Sno92] and [DS93]. In these papers the term *indeterminate temporal information* characterizes a strict subclass of indefinite temporal information.

- The time associated with an event or fact might be specified via a relationship to the time associated with another event or fact. In this case, the two times are related through a *qualitative* or *quantitative temporal constraint.* As an example, consider the two statements "The explosion occured *after* John left the scene" and "The explosion occured *5 to 10 minutes after* John left the scene".
- The granularity of the system time line does not match the precision to which the time associated with an event or fact is known [DS93]. As an example, consider storing the information "John was hired on January 25, 1993" in a system with time-stamps in the granularity of a second.
- Dating techniques can be imperfect. All clocks have inherent imprecision [DS93].

Indefinite temporal information can be represented in most of the temporal formalisms developed by artificial intelligence researchers (see sections 2.2, 2.3 and sec:persistence) since these formalisms are very expressive. Several database researchers have recently recognized the importance of indefinite temporal information [Cha88, MBJK90, GNP92, MPB92, vdM92, Kou93a, DS93]. In the rest of this section we discuss all of these proposals in turn. We will not discuss the proposal of [Kou93a] since it is essentially the model of indefinite deTCL-constraint databases discussed in the examples of chapter 4.

The Possible Chronons Model

[DS93] extends the bitemporal database model of [Sno87] described in section 2.5.1 by allowing indefinite temporal information about events and facts. Dyreson and Snodgrass use the term *indeterminate* instead of indefinite.

Let us assume that the granularity of timestamps in our database is one day. Now consider the following relation.

EMP

NAME	DEPT	Valid Time	
		From	To
John	Toys	1-1-81	10-82
Tom	Toys	1-6-82	10-82
Mary	Shoes	1-3-81	∞
Tina	Toys	1-5-84	∞

EMP is an *indeterminate historical interval relation.*[27] It tells us that John and Tom left their departments in October, 1982 but it does not tell us the exact day that they left. In general, an *indeterminate interval* is an interval which is bounded by two indeterminate events. An *indeterminate event* is described by a *lower support chronon*, an *upper support chronon* and a *probability mass function.* The support chronons give the period during which the event might have occured; the probability mass function gives the probability that the event occured during each chronon.

Let us now consider the query "Find all employees that were possibly working in Toys on October 28, 1982". The following `retrieve` statement in the extended TQUEL expresses this query.

```
range of E is EMP with confidence 0
retrieve (NAME = E.NAME)
```

[27]Needless to say that transaction time cannot be indefinite.

```
valid at E
where E.DEPT=''Toys''
when E overlap 28-10-82 with plausibility 1
```

[DS93] propose two ways of refining the answer to a user query. *Range credibility* specifies a degree of confidence in the data. It can be any integer between 0 and 100 and appears optionally in the `range` clause of a `retrieve` statement. *Ordering plausibility* specifies the plausibility of the ordering of the times participating in the retrieval. It can be any integer between 1 and 100 and appears optionally in the `valid` or `when` clause of a `retrieve` statement. In the above query we have *full confidence* in our data. In addition, we would like to consider *all possible* cases where the valid time associated with E overlaps with "28-10-82".

The answer to the above query is given by the following relation.

NAME	Valid Time	
	From	To
John	1-1-81	10-82
Tom	1-6-82	10-82

The probabilistic features of the possible chronons model should make it very attractive for certain applications.

The Knowledge Representation Language Telos

Handling indefinite temporal knowledge is one of the strengths of the knowledge representation language Telos [MBJK90]. Telos allows the definition of structured objects with a temporal dimension organized in an extended semantic network structure. In addition, a first order logic with temporal arguments is offered for expressing deductive rules and integrity constraints on the knowledge base.

The knowledge model of Telos is bitemporal: a Telos knowledge base represents current but also past *beliefs* about the *history* of an application domain. The historical dimension of Telos adapts Allen's framework for qualitative temporal knowledge [All83]. The modifications in Allen's framework include (i) slight changes to the definitions of the thirteen temporal relationships, mostly dictated by language design considerations; (ii) incorporation of temporal constants, such as conventional dates and times (e.g., `1988/12/7` denoting December 7, 1988), semi-infinite intervals having conventional dates or times as one endpoint (e.g., `1986/10/25..*`), the infinite interval `Alltime` and the special interval `Now` denoting the current system time; (iii) restricting the power of temporal assertions that can be told to the system.

With such a framework on hand, it is possible to represent temporal information as shown in the following `TELL` statement:

```
TELL TOKEN martian IN Paper ( at 1986/10..* ) WITH
     author
       firstAuthor: Stanley ( at 1986/10..* );
                  : LaSalle ( at 1987/1..* );
                  : Wong ( before 1987/5 )
     title
         : 'The MARTIAN system'
END
```

This operation introduces the object `martian` in the knowledge base (we suppose it was not there already). The `IN` clause makes `martian` an instance of the class `Paper` for an unbounded time interval starting October 1986. Similarly, the `WITH` clause asserts that `Stanley` is the first author of `martian` during the interval `1986/10..*`, `LaSalle` is an author during the interval `1987/1..*` while `Wong` was an author for some time before May 1987 (but we do not know the exact time).

The history of the application domain can be modeled by augmenting knowledge base facts with a *history time* i.e., an interval representing the time during which these facts are true in the application domain. The notion of history time is the analogue of valid time found in temporal database models. History time is useful not only for tokens but also for generic propositions: for example, the *definition* "personal deduction" in an Income Tax Act, may only apply for one year.

The syntax of the language, illustrated by the above example, is restricted in the sense that it only allows a single temporal relationship to appear in each of the temporal components of a given definition. This is in contrast with Allen's original framework where disjunctions of relationships were allowed between time intervals (e.g., `PaulsDateOfBirth (before during) 1975`) in order to express further kinds of incomplete knowledge. The reasons for this expressive retreat are strictly pragmatic: verifying the consistency of a network of temporal relations for Allen's algebra is NP-hard, as is computing all the consequences of a network. However, reasoning with certain subsets of the framework, including the one adopted here, is tractable [VKvB89].

The TSOS system

TSOS is a temporal reasoning system presented in [MPB92]. TSOS assumes a discrete model of time and uses a hierarchical calendar representation for expressing times at different levels of granularity. The time entities considered are points, intervals and time extensions (e.g., "two days" or "three weeks").

Temporal information can be modeled in TSOS in terms of *instantaneous events* and *propositions*. Events can be specified by an assertion $Happens(e,t)$ where e is an event identifier and t is a time point. The time point t can be specified exactly or it can be constrained to belong to an interval called the *admissibility interval*. In this way indefinite information about events can be represented in TSOS. Propositions can be specified by an assertion $Holds(p, m, obs)$ where p is a proposition identifier, m is the modality *always* or *sometimes* and obs is the *observation interval*. If m is *always* then this assertion means that p is true for all time points in the observation interval obs. If m is *sometimes* then the meaning of the assertion is that p is true for at least one time points in the observation interval obs.

TSOS offers an expressive query language which allows one to ask *modal* queries e.g., "Is it *possible* that p holds before time t?" or "Is it *certain* that p holds after time t?". Finally, TSOS should be classified as a bitemporal system since it provides explicit support for transaction time.

The Temporal Data Model of Chaudhuri

The temporal data model of Chaudhuri [Cha88] is the most important proposal for databases with *relative* temporal information. In [Cha88] temporal data are stored in a relation $Temp$ with attributes E_1, R and E_2. The domain of E_1 is the set of event identifiers. The domain of E_2 is the set of temporal relations (e.g., Allen's 13 interval relations [All83]). Finally, the domain of E_2 is the union of the set of event identifiers and the set of absolute time values. For example, $(event3,\ before,\ event2)$, $(event3,\ <,\ 25Jan88)$ can be tuples of $Temp$. Temporal data can then be queried like ordinary relations. It is clear that indefinite temporal information can be represented very nicely in this model.

As follows from section 2.2, $Temp$ can be viewed as a graph where events are nodes and temporal relations are labels on the edges. Chaudhuri discusses why this is valuable in a database context for representation and query answering purposes.

The weak points of this model is that it cannot capture the full class of temporal constraints discussed in section 2.2 (e.g., it cannot represent bounds on the distance of two time points; it cannot represent disjunctive constraints). Moreover its query language is not as expressive as the ones usually found in a database context.

The Temporal Data Model of Gadia, Nair and Poon

The model of [GNP92] is an extension of the model of [GY88] using the ideas of [Bis83]. In this model two kinds of indefinite information can be expressed. The first one allows users to say that a tuple is in a relation at least over the interval $[a, b]$ and at most over the interval $[c, d]$ where $[a, b] \subseteq [c, d]$ (i.e., one can be uncertain about the existence of the tuple during interval $[c, d] - [a, b]$). The second kind of indefinite information allows attribute values to be unknown during the period of (possible or certain) existence of the tuple they belong.

Indefinite Order Databases

Van der Meyden studied the complexity of query answering in indefinite order databases [vdM92]. He considered a first-order language $\mathcal{L}$ with two sorts: an *object* sort and an *order* sort. The object sort represents traditional kinds of data while the order sort represents points in a linearly ordered domain. There are two types of atomic formulas:

- $P(\overline{a})$ where P is a predicate symbol and $\overline{a}$ is a tuple of constants or variables. These are called *proper* atoms.
- $u < v$ where u and v are order constants or variables. These are called *order* atoms.

An *indefininite order database* is a set of ground atomic formulas in the language $\mathcal{L}$. The query language for indefinite order databases is the language $\mathcal{L}$.

[vdM92] concentrated on the problem of deciding whether a yes/no positive existential query is true in all possible worlds represented by an indefinite order database. This problem has co-NP-complete data complexity and Π_2^p-complete combined complexity. To obtain tractability [vdM92] investigates indefinite order databases with monadic proper predicates. In this case the data complexity of conjunctive queries is in linear time but the combined complexity is co-NP-hard. Finally, [vdM92] investigated several specializations of this case where the combined complexity is in PTIME as well. The complexity bounds hold independently of whether $<$ is interpreted as a discrete, dense or finite linear order. These results seem to point out that indefinite order databases will be very hard to implement in their full generality.

2.10 Conclusion

We have presented a survey of temporal representation and reasoning as it has been developed in artificial intelligence and databases. The research carried out by the artificial intelligence community can be divided into three areas: (i) representation and reasoning with temporal constraints [Mei91a] (ii) logical formalisms for time, actions, events and persistence [KS86] and (iii) applications involving temporal reasoning e.g., planning and scheduling [AKP91, Fox87]. Research in temporal databases mostly concentrated on temporal models with definite temporal information [TCG+93]. Several recent proposals deal with infinite [BCW93] or indefinite temporal information [Cha88, MBJK90, GNP92, MPB92, vdM92, DS93]. The representation of infinite information using constraints has been studied in a more general context in

[JL87, KKR90]. The representation of indefinite information in relational databases has been studied in [IL84, Rei84, Rei86, Var86, Gra89, AKG91].

In this dissertation we present the *indefinite $\mathcal{L}$-constraint database scheme* which integrates and extends the proposals of [KKR90, IL84, Gra89]. By varying $\mathcal{L}$ one can obtain constraint-based models which can be useful for several applications. In this dissertation we only concentrate on *indefinite temporal constraint database* models. We study the complexity of query evaluation in these models and compare it with the complexity of query evaluation in standard relational databases. Finally, we present a new class of temporal constraints for which we study the basic problems discussed in section 2.2.

Chapter 3

Representation and Reasoning with Temporal Constraints

In this chapter we introduce the theories of temporal constraints which we will study in the rest of the dissertation. These theories are:

- the first-order theory of point constraints over the integers
- the first-order theory of point constraints over the rationals
- the first-order theory of point and interval constraints over the integers
- the first-order theory of point and interval constraints over the rationals

For each of the above theories, we initially concentrate on tools and algorithms for deciding consistency and performing variable elimination. This material will serve as a backbone for the results presented in the rest of the dissertation.

This chapter is organized as follows. In section 3.1 we present some background material from first-order logic. In section 3.2 we introduce the theories of temporal constraints which we will consider. In section 3.3 we introduce binary inequality constraint networks. Binary constraint networks are the technical tools that have been traditionally used in the study of temporal constraints. Finally in sections 3.4 and 3.5 we study deciding consistency and performing variable elimination for temporal constraints.

3.1 Background

We present the standard definitions of (many-sorted) first-order languages, structures and theories. With a few exceptions, we will follow [End72] in our notation and terminology.

A *many-sorted first-order language* $\mathcal{L}$ is specified by providing a collection of infinitely many symbols arranged as follows:

- *Parameters* or *non-logical symbols*
  - Sorts: We assume that we have a non-empty set whose elements are called *sorts*. This set will be denoted by $sorts(\mathcal{L})$.

- Predicate symbols: For each $n > 0$ and each n-tuple $(s_1, \cdots, s_n)$ of sorts, there is a (possibly empty) set of n-place predicate symbols, each of which is said to be of sort $(s_1, \cdots, s_n)$.
- Constant symbols: For each sort s, there is a (possibly empty) set of constant symbols of sort s.
- Function symbols: For each $n > 0$ and each $n+1$-tuple $(s_1, \cdots, s_n, s_{n+1})$ of sorts, there is a (possibly empty) set of n-argument function symbols, each of which is said to be of sort $(s_1, \cdots, s_n, s_{n+1})$.

- *Logical symbols*
  - Parentheses: (,).
  - Sentential connectives: $\neg$, $\vee$.
  - Variables: For each sort s, there is a countably infinite set $VAR^s_{\mathcal{L}}$ of variables of sort s. The set of all variables $\bigcup_{s \in sorts(\mathcal{L})} VAR^s_{\mathcal{L}}$ is denoted by $VAR_{\mathcal{L}}$.
  - Equality symbols: For every $s \in sorts(\mathcal{L})$, there may be an equality symbol $=_s$ of sort (s, s).

We will also use the derived connectives $\wedge$, $\supset$ and $\equiv$ with their usual meaning.

The concepts of *well-formed formulas* and *sentences* are defined as usual. The set of well-formed formulas of the language $\mathcal{L}$ will be denoted by $WFF(\mathcal{L})$.

A *many-sorted structure* for the language $\mathcal{L}$ (or an $\mathcal{L}$-structure) is a function $\mathbf{M}$ which makes the following assignments to the symbols of $\mathcal{L}$:

- To each sort s, $\mathbf{M}$ assigns a non-empty set $dom(s, \mathbf{M})$.
- To each predicate symbol P of sort $(s_1, \cdots, s_n)$, $\mathbf{M}$ assigns the relation

$$P^{\mathbf{M}} \subseteq dom(s_1, \mathbf{M}) \times \cdots \times dom(s_n, \mathbf{M}).$$

- To each function symbol f of sort $(s_1, \cdots, s_n, s_{n+1})$, $\mathbf{M}$ assigns the function

$$f^{\mathbf{M}} : dom(s_1, \mathbf{M}) \times \cdots \times dom(s_n, \mathbf{M}) \rightarrow dom(s_{n+1}, \mathbf{M}).$$

- To each constant symbol c of sort s, $\mathbf{M}$ assigns an element $c^{\mathbf{M}}$ of $dom(s, \mathbf{M})$.

Let $\mathbf{M}$ be an $\mathcal{L}$-structure. A *valuation* v in $\mathbf{M}$ is a function

$$v : VAR_{\mathcal{L}} \rightarrow \bigcup_{s \in sorts(\mathcal{L})} dom(s, \mathbf{M})$$

such that $v(VAR^s_{\mathcal{L}}) \subseteq dom(s, \mathbf{M})$.

The notions of satisfaction and truth in a structure $\mathbf{M}$ are defined in the usual way. We will use $\mathbf{M} \models \phi[v]$ to denote that the structure $\mathbf{M}$ *satisfies* ϕ with valuation v. If σ is a sentence then $\mathbf{M} \models \sigma$ denotes that σ is *true* in structure $\mathbf{M}$.

A *first-order theory* is a set of sentences, in some language $\mathcal{L}$, which is closed under logical implication. Let $\mathbf{M}$ be an $\mathcal{L}$-structure. The *theory of the structure* $\mathbf{M}$, denoted by $Th(\mathbf{M})$, is the set of sentences of $\mathcal{L}$ that are true in $\mathbf{M}$.

Remark 3.1.1 The languages that we will consider will not be interpreted over arbitrary structures but rather over *fixed* structures which we will specify in detail. The *intended structure* for a given language $\mathcal{L}$ will be denoted by $\mathbf{M}_{\mathcal{L}}$.

If ϕ, ψ are formulas of the language $\mathcal{L}$, we will say that ϕ *logically implies* ψ, denoted by $\phi \models \psi$, if and only if $\mathbf{M}_{\mathcal{L}} \models \phi \supset \psi$.

In addition to *many-sorted first-order languages, structures* and *theories*, we also deal with languages with a single sort. In this case the sort is not mentioned at all and the above definitions can be refined accordingly [End72].

3.1.1 Constraints

For every language $\mathcal{L}$, we will distinguish a class of quantifier free formulas called $\mathcal{L}$*-constraints*. The atomic formulas of $\mathcal{L}$ will be included in the class of $\mathcal{L}$-constraints. There will also be two distinguished $\mathcal{L}$-constraints *true* and *false* with obvious semantics. In the rest of this dissertation these assumptions will always be made implicitly. Similar assumptions have been made in [Mah93] in the context of the CLP scheme.

A set of $\mathcal{L}$-constraints will be the algebraic counterpart of the logical conjunction of its members. Thus we will freely mix the terms "set of $\mathcal{L}$-constraints" and "conjunction of $\mathcal{L}$-constraints".

Definition 3.1.1 Let C be a set of $\mathcal{L}$-constraints in variables $x_1, \ldots, x_n$ of sorts $s_1, \ldots, s_n$. The *solution set* of C, denoted by $Sol(C)$, is:

$$\{(a_1, \ldots, a_n) \in dom(s_1, \mathbf{M}_{\mathcal{L}}) \times \cdots \times dom(s_n, \mathbf{M}_{\mathcal{L}}) : \text{ for each } c \in C,\ \mathbf{M}_{\mathcal{L}} \models c[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n]\}$$

Each member of $Sol(C)$ is called a *solution* of C.

Definition 3.1.2 A set of $\mathcal{L}$-constraints C is called *consistent* or *satisfiable* if $Sol(C) \neq \emptyset$. Two sets of $\mathcal{L}$-constraints are called *equivalent* if they have the same solution set.

The notion of projection of a solution set will be important in the rest of this dissertation.

Definition 3.1.3 Let C be a set of constraints in variables $X = \{x_1, \ldots, x_n\}$. If $V = \{x_{j_1}, \ldots, x_{j_m}\}$ is a subset of X then the *projection* of $Sol(C)$ on V, denoted by $Proj_V(Sol(C))$, is the set $\{(a_{j_1}, \ldots, a_{j_m}) : (a_1, \ldots, a_n) \in Sol(C)\}$.

The above notions of solution set, consistency, equivalence and projection can easily be extended to *boolean combinations* of $\mathcal{L}$-constraints. We will omit these definitions since they are straightforward.

3.2 First-Order Theories of Temporal Constraints

In this section we introduce the first-order theories of point and interval constraints over two linear and unbounded time structures: the set of integers $\mathbb{Z}$ and the set of rationals $\mathbb{Q}$.

3.2.1 Point and Interval Constraints in Discrete Time

In this section we discuss two theories of temporal constraints over the integers. At first, we consider *points* to be our only time entities. Points will be identified with the integers. The language *diPCL* (*di*screte *P*oint *C*onstraint *L*anguage) will allow us to make stamements about points in time. diPCL is a first-order language with equality defined as follows. The non-logical symbols of diPCL include a countably infinite set of constants (the integer numerals), function symbol $-$ of arity 2 and predicate symbol $<$ of arity 2.

The set of *terms* of diPCL are defined by the following rules:

- Constants and variables are terms.
- If $t_1,\ t_2$ are variables or constants then $t_1 - t_2$ is a term.

An *atomic formula* of diPCL is a formula of the form $t \sim c$ or $c \sim t$ where $\sim$ is $<$ or $=$ and t is a term. The set of *well-formed formulas* of diPCL are defined as usual (i.e., they are built up from the atomic formulas using quantifiers and connectives). As usual, we will write $t_1 \leq t_2$ instead of $t_1 = t_2 \ \vee \ t_1 < t_2$, $t_2 > t_1$ instead of $t_1 < t_2$, and $t_1 \geq t_2$ instead of $t_1 = t_2 \ \vee \ t_1 > t_2$.

The symbols of diPCL are interpreted with respect to the fixed structure $\mathbf{Z}$ which captures our assumptions. The domain of $\mathbf{Z}$ is $\mathbb{Z}$. To each integer numeral, $\mathbf{Z}$ assigns its corresponding integer number. To function symbol $-$, $\mathbf{Z}$ assigns the function $-_{\mathbb{Z}}$ which is the subtraction operation for integers. To predicate symbol $<$, $\mathbf{Z}$ assigns the relation $<_{\mathbb{Z}}$ ("less than") over the integers.

We will take the theory $Th(\mathbf{Z})$ to be *the* theory of point constraints in linear, unbounded and discrete time. This theory will also be denoted by *diPC*.[1] diPC is a subtheory of Presburger arithmetic [Rab77].

Definition 3.2.1 A diPCL-*constraint* is a diPCL formula of the form $t \sim c$ where t is a term, c is a constant and $\sim$ is $=,\ <,\ >,\ \leq$ or $\geq$.

Example 3.2.1 The formulas $p_1 < p_2,\ p_3 - p_4 \geq 15,\ p_3 = 5$ are examples of diPCL-constraints.

We will now define the language diTCL (*di*screte *T*emporal *C*onstraint *L*anguage) which is an extension of diPCL. The time entities in this language are points and intervals. Points are identified with the integers while intervals are considered to be pairs of points. diTCL has two sorts: $\mathcal{Z}$ (for points or integers) and $\mathcal{I}_{\mathcal{Z}}$ (for integer intervals). The non-logical symbols of diTCL include a countably infinite set of constant symbols of sort $\mathcal{Z}$ (the integer numerals), function symbols L and R of sort $(\mathcal{I}_{\mathcal{Z}},\ \mathcal{Z})$, function symbol $-$ of sort $(\mathcal{Z},\ \mathcal{Z},\ \mathcal{Z})$ and predicate symbol $<$ of sort $(\mathcal{Z},\ \mathcal{Z})$.

The set of *simple terms* of diTCL are defined by the following rules:

- Constants are simple terms.
- A variable of sort $\mathcal{Z}$ is a simple term.
- If i is a variable of sort $\mathcal{I}_{\mathcal{Z}}$ then i_L and i_R are simple terms.

The set of *terms* of diTCL can now be defined as follows:

- Simple terms are terms.
- If t_1 and t_2 are simple terms then $t_1 - t_2$ is a term.

An *atomic formula* of diTCL is a formula of the form $t \sim c$ or $c \sim t$ where $\sim$ is $<$ or $=$, and t is a term. The set of *well-formed formulas* is defined as usual.

[1] In general we will adhere to the following convention. If a theory is denoted by Th, the language of Th will be denoted by ThL.

The symbols of diTCL are interpreted with respect to the fixed structure $\mathbf{ZI_Z}$ which captures our assumptions. $\mathbf{ZI_Z}$ assigns to the sort $\mathcal{Z}$ the set of integers $\mathbb{Z}$, and to the sort $\mathcal{I_Z}$ the set of *integer intervals*

$$Int(\mathbb{Z}) = \{\langle a, b\rangle : \ a, b \in \mathbb{Z} \text{ and } a <_{\mathbb{Z}} b\}.$$

To each constant symbol of sort $\mathcal{Z}$, $\mathbf{ZI_Z}$ assigns its corresponding element of $\mathbb{Z}$. To function symbols L and R, $\mathbf{ZI_Z}$ assigns the functions

$$l, r : \ Int(\mathbb{Z}) \rightarrow \mathbb{Z}$$

such that $l(\langle a, b\rangle) = a$ and $r(\langle a, b\rangle) = b$. These functions map each interval to its left and right endpoint respectively. To function symbol $-$, $\mathbf{ZI_Z}$ assigns the function $-_{\mathbb{Z}}$ which is the subtraction operation over the integers. To predicate symbol $<$, $\mathbf{ZI_Z}$ assigns the relation $<_{\mathbb{Z}}$ over the integers.

We will take the theory $Th(\mathbf{ZI_Z})$ to be *the* theory of point and interval constraints in linear, unbounded and discrete time. This theory will also be denoted by *diTC*.

Definition 3.2.2 A diTCL-*constraint* is a diTCL formula of the form $t \sim c$ where t is a term, c is a constant and $\sim$ is $=$, $<$, $>$, $\leq$ or $\geq$.

Example 3.2.2 The formulas $p_1 \leq 2$, $i_R - i_L > 5$, $p_1 - i_R = 4$ are examples of diTCL-constraints.

Finally, observe that all interval-to-interval and point-to-interval relations of [Mei91a] can be introduced as defined relations in diTC.

3.2.2 Point and Interval Constraints in Dense Time

In this section we discuss two theories of temporal constraints which are exactly like the theories of the previous section except that our time structure now is the rational numbers. At first, we consider *points* to be our only time entities. Points will be identified with the rationals. The language dePCL (*de*nse *P*oint *C*onstraint *L*anguage) will allow us to make stamements about points in time. dePCL is a first-order language with equality defined as follows. The non-logical symbols of dePCL include a countably infinite set of constants (the rational numerals), function symbol $-$ of arity 2 and predicate symbol $<$ of arity 2.

The set of *terms* of dePCL are defined as for diPCL:

- Constants and variables are terms.
- If t_1, t_2 are variables or constants then $t_1 - t_2$ is a term.

An *atomic formula* of dePCL is a formula of the form $t \sim c$ or $c \sim t$ where $\sim$ is $<$ or $=$, and t is a term. The set of *well-formed formulas* of dePCL are defined as usual.

The symbols of dePCL are interpreted with respect to the fixed structure $\mathbf{Q}$ which captures our assumptions for dense time. The domain of $\mathbf{Q}$ is $\mathbb{Q}$. To each rational numeral, $\mathbf{Q}$ assigns its corresponding element of $\mathbb{Q}$. To function symbol $-$, $\mathbf{Q}$ assigns the function $-_{\mathbb{Q}}$ which is the subtraction operation over the rationals. To predicate symbol $<$, $\mathbf{Q}$ assigns the relation $<_{\mathbb{Q}}$ over the rationals.

We will take the theory $Th(\mathbf{Q})$ to be *the* theory of point constraints in dense time. This theory will also be denoted by *dePC*. dePC is a subtheory of the theory of real addition with order [Rab77].

Let us now define constraints in the language dePCL.

Definition 3.2.3 A dePCL-*constraint* is a dePCL formula of the form $t \sim c$ where t is a term, c is a constant and $\sim$ is $=$, $<$, $>$, $\leq$ or $\geq$.

If C is a set of dePCL-equalities in n variables, the solution set of C is an affine subset of $\mathbb{Q}^n$. If C is a set of dePCL-inequalities in n variables, the solution set of C is a convex polyhedron in $\mathbb{Q}^n$.

Example 3.2.3 The formulas $p_1 < p_2$, $p_3 - p_4 \leq 5$, $p_6 - p_7 \leq 3.5$ are examples of dePCL-constraints.

We will now define the language deTCL (*de*nse *T*emporal *C*onstraint *L*anguage). The time entities in this language are points and intervals. Points are *identified* with the rationals while intervals are considered to be pairs of points. The language deTCL is an extension of dePCL. It is a first-order language with two sorts: $\mathcal{Q}$ (for points or rationals) and $\mathcal{I}_{\mathcal{Q}}$ (for rational intervals). The non-logical symbols of deTCL include a countably infinite set of constant symbols of sort $\mathcal{Q}$ (the rational numerals), function symbols L and R of sort $(\mathcal{I}_{\mathcal{Q}},\ \mathcal{Q})$, function symbol $-$ of sort $(\mathcal{Q},\ \mathcal{Q},\ \mathcal{Q})$ and predicate symbol $<$ of sort $(\mathcal{Q},\ \mathcal{Q})$.

The set of *simple terms* of deTCL are defined by the following rules:

- Constants are simple terms.
- A variable of sort $\mathcal{Q}$ is a simple term.
- If i is a variable of sort $\mathcal{I}_{\mathcal{Q}}$ then i_L and i_R are simple terms.

The set of *terms* of deTCL can now be defined as follows:

- Simple terms are terms.
- If t_1 and t_2 are simple terms then $t_1 - t_2$ is a term.

An *atomic formula* of deTCL is a formula of the form $t \sim c$ or $c \sim t$ where $\sim$ is $<$ or $=$ and t is a term. The set of *well-formed formulas* is defined as usual.

The symbols of deTCL are interpreted by the many-sorted structure $\mathbf{QI_Q}$. $\mathbf{QI_Q}$ assigns to the sort $\mathcal{Q}$ the set of rationals $\mathbb{Q}$, and to the sort $\mathcal{I}_{\mathcal{Q}}$ the set of *rational intervals*

$$Int(\mathbb{Q}) = \{\langle a, b\rangle :\ a, b \in \mathbb{Q} \text{ and } a <_{\mathbb{Q}} b\}.$$

To each constant symbol of sort $\mathcal{Q}$, $\mathbf{QI_Q}$ assigns its corresponding element of $\mathbb{Q}$. To function symbols L and R, $\mathbf{QI_Q}$ assigns the functions

$$l, r :\ Int(\mathbb{Q}) \rightarrow \mathbb{Q}$$

such that $l(\langle a, b\rangle) = a$ and $r(\langle a, b\rangle) = b$. These functions map each interval to its left and right endpoint respectively. To function symbol $-$, $\mathbf{QI_Q}$ assigns the function $-_{\mathbb{Q}}$ which is the subtraction operation over the rationals. To predicate symbol $<$, $\mathbf{QI_Q}$ assigns the relation $<_{\mathbb{Q}}$ over the rationals.

We will take the theory $Th(\mathbf{QI_Q})$ to be *the* theory of point and interval constraints in dense time. This theory will also be denoted by *deTC*.

Definition 3.2.4 A deTCL-*constraint* is a deTCL formula of the form $t \sim c$ where t is a term, c is a constant and $\sim$ is $=$, $<$, $>$, $\leq$ or $\geq$.

Example 3.2.4 The formulas $p_1 \leq 2.5$, $i_R - i_L > 5$, $p \leq i_L$ are examples of constraints in deTCL.

3.2.3 From Intervals To Points

We will now consider translating formulas involving points and intervals into "equivalent" formulas involving only points and vice versa. In the forthcoming sections, we will use these translations for solving all problems associated with diTCL- or deTCL-constraints. The translation of formulas involving intervals to formulas involving points has been a standard technique in temporal reasoning literature [VK86, vB90b, Lad88]. Our approach is closely related to [Lad88] who considered translating formulas of the first-order theory of the binary qualitative interval relations defined by Allen [All84]) to formulas in the first-order theory of the structure $(\mathbb{Q}, <)$.

We will first define the translation function

$$itp : WFF(\text{diTCL}) \rightarrow WFF(\text{diPCL}).$$

Let us assume that ϕ is a well-formed formula of diTCL. For every variable i of sort $\mathcal{I}_{\mathcal{Z}}$ which appears in ϕ, let B_i and E_i be brand new variables of diPCL. The well-formed formula $itp(\phi)$ of diPCL is defined by the following rules:

1. If ϕ is an atomic formula then $itp(\phi)$ is the same as ϕ except that terms i_L and i_R have been substituted by variables B_i and E_i respectively.
2. If ϕ is $\phi_1 \vee \phi_2$ then $itp(\phi)$ is $itp(\phi_1) \vee itp(\phi_2)$.
3. If ϕ is $\neg\phi_1$ then $itp(\phi)$ is $\neg itp(\phi_1)$.
4. If ϕ is $(\exists x/\mathcal{Z})\phi_1$ then $itp(\phi)$ is $(\exists x) itp(\phi_1)$.
5. If ϕ is $(\exists i/\mathcal{I}_{\mathcal{Z}})\phi_1$ then $itp(\phi)$ is $(\exists B_i)(\exists E_i)(B_i < E_i \wedge itp(\phi_1))$.

The following proposition relates the satisfiability of formulas ϕ and $itp(\phi)$.

Proposition 3.2.1 *Let ϕ be a formula of diTCL with free variables $p_1, \ldots, p_n$ of sort $\mathcal{Z}$ and $i_1, \ldots, i_m$ of sort $\mathcal{I}_{\mathcal{Z}}$. If $p_1^0, \ldots, p_n^0 \in \mathbb{Z}$ and $i_1^0, \ldots, i_m^0 \in Int(\mathbb{Z})$ then*

$$\mathbf{ZI_Z} \models \phi[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, i_1 \leftarrow i_1^0, \ldots, i_m \leftarrow i_m^0]$$

if and only if

$$\mathbf{Z} \models itp(\phi)[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, B_{i_1} \leftarrow l(i_1^0), \ldots, B_{i_m} \leftarrow l(i_m^0), E_{i_1} \leftarrow r(i_1^0), \ldots, E_{i_m} \leftarrow r(i_m^0)]$$

where l and r are functions defined by structure $\mathbf{ZI_Z}$.

Proof: By induction on the structure of formula ϕ. ■

Associated with itp is the inverse translation function

$$pti : WFF(\text{diPCL}) \rightarrow WFF(\text{diTCL})$$

which takes as argument a well-formed formula ϕ of diPCL and returns a well-formed formula $pti(\phi)$ of diTCL which is obtained from ϕ as follows:

- If B_i or E_i is a *free* variable of ϕ then i will be a free variable of sort $\mathcal{I}_{\mathcal{Z}}$ of $pti(\phi)$.
- If B_i (resp. E_i) is a *free* variable of ϕ then all the occurences of B_i (resp. E_i) are substituted by i_L (resp. i_R).

The following proposition relates the satisfiability of formulas ϕ and $pti(\phi)$.

Proposition 3.2.2 *Let ϕ be a formula of diPCL with free variables $p_1, \ldots, p_n, B_{i_1}, \ldots, B_{i_m}$, $E_{i_1}, \ldots, E_{i_m}$. If $p_1^0, \ldots, p_n^0, \beta_1, \ldots, \beta_m, \epsilon_1, \ldots, \epsilon_m \in \mathbb{Z}$ then*

$$\mathbf{Z} \models \phi[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, B_{i_1} \leftarrow \beta_1, \ldots, B_{i_m} \leftarrow \beta_m, E_{i_1} \leftarrow \epsilon_1, \ldots, E_{i_m} \leftarrow \epsilon_m]$$

if and only if

$$\mathbf{ZI_Z} \models pti(\phi)[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, i_1 \leftarrow \langle \beta_1, \epsilon_1 \rangle, \ldots, i_m \leftarrow \langle \beta_m, \epsilon_m \rangle]$$

Proof: By induction on the structure of formula ϕ. ∎

The same translations can also be defined for deTCL formulas. The details will be omitted since they are straightforward.

3.3 Binary Constraint Networks

Traditionally binary constraints are represented by *binary constraint networks* [Mon74, Mac77, MF85, DP88]. Let C be a set of binary inequality constraints in variables $x_1, \ldots, x_n$. The *binary inequality constraint network* (BICN) associated with C is a labeled directed graph $G = (V, E)$ where $V = \{1, \ldots, n\}$. Node i represents variable x_i and edge (i, j) represents the binary constraints involving x_i and x_j.

Let us assume that the constraints on $x_j - x_i$ are $x_j - x_i \leq d_{ij}$ and $x_j - x_i \geq -d_{ji}$ where $-d_{ji} < d_{ij}$. Then the corresponding BICN N will have an edge $i \rightarrow j$ labeled by the *convex* interval $N_{ij} = [-d_{ji}, d_{ij}]$. Interval N_{ij} can also be open from the right or left if some of the above inequalities are strict.

In BICN unary constraints can be represented as binary ones with the introduction of a special variable x_0 and the addition of the constraint $x_0 = 0$ (e.g., $x_1 \leq 2$ will be represented as $x_1 - x_0 \leq 2$). Thus in the rest of this document, we will not consider unary constraints.

Example 3.3.1 In figure 3.1 the network on the left represents the constraints

$$1 \leq x_2 - x_1 \leq 4,\ 2 \leq x_2 - x_3 \leq 5,\ 1 \leq x_4 - x_1 \leq 4,\ 2 \leq x_4 - x_3 \leq 5.$$

When we draw a BICN, we never draw edges (i, i). When we draw an edge $i \rightarrow j$, we do not draw edge $j \rightarrow i$. Finally, we do not draw edges labeled with $(-\infty, \infty)$.

The function *Constraints* gives the set of binary constraints represented by N. More formally, if N is a BICN with n nodes then $Constraints(N)$ is the following set of constraints in variables $x_1, \ldots, x_n$:

$$\begin{aligned}
Constraints(N) \quad = \quad & \{x_j - x_i \leq d_{ij},\ x_j - x_i \geq -d_{ji} \mid N_{ij} = [-d_{ji}, d_{ij}],\ i < j\} \cup \\
& \{x_j - x_i < d_{ij},\ x_j - x_i \geq -d_{ji} \mid N_{ij} = [-d_{ji}, d_{ij}),\ d_{ij} \neq \infty,\ i < j\} \cup \\
& \{x_j - x_i \leq d_{ij},\ x_j - x_i < -d_{ji} \mid N_{ij} = (-d_{ji}, d_{ij}],\ d_{ji} \neq \infty,\ i < j\} \cup \\
& \{x_j - x_i < d_{ij},\ x_j - x_i < -d_{ji} \mid N_{ij} = (-d_{ji}, d_{ij}),\ d_{ij}, d_{ji} \neq \infty,\ i < j\} \cup \\
& \{x_j - x_i \geq -d_{ji} \mid N_{ij} = [-d_{ji}, \infty),\ i < j\} \cup \\
& \{x_j - x_i \leq d_{ij}, \mid N_{ij} = (-\infty, d_{ij}],\ i < j\} \cup \\
& \{x_j - x_i < -d_{ji} \mid N_{ij} = (-d_{ji}, \infty),\ i < j\} \cup \\
& \{x_j - x_i < d_{ij}, \mid N_{ij} = (-\infty, d_{ij}),\ i < j\}.
\end{aligned}$$

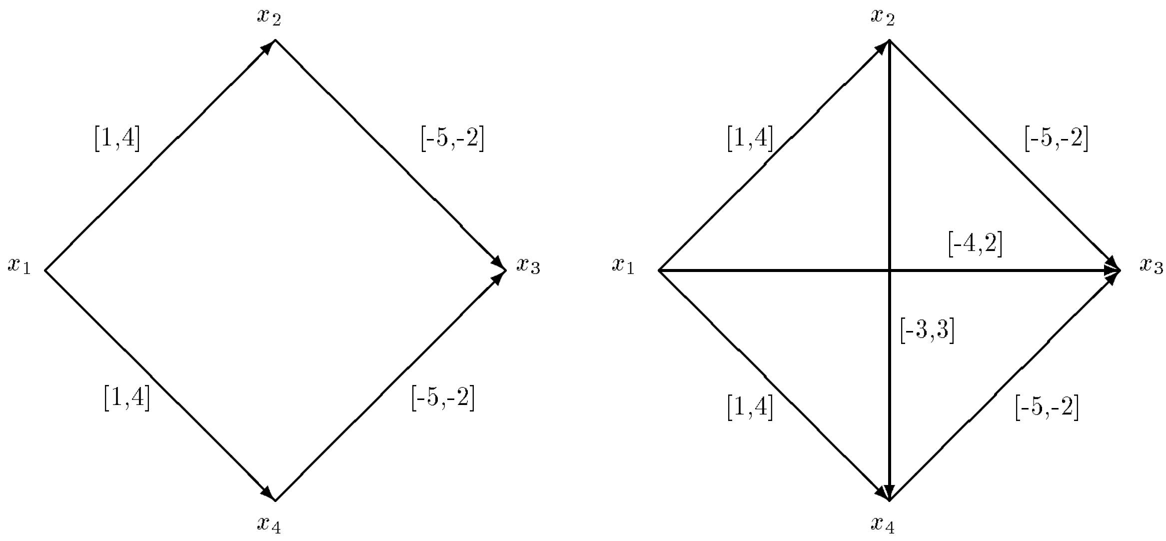

Figure 3.1: Binary inequality constraint networks

The notions of *solution, consistency* and *equivalence* can also be defined for BICN in the obvious way. Let N and N' be two BICN having the same set of variables. We will say that N is *tighter* than N', denoted by $N \subseteq N'$, if for each pair of nodes i, j, $N_{ij} \subseteq N'_{ij}$. For each BICN N there is a unique equivalent BICN N' which is minimal with respect to $\subseteq$. This network is called the *minimal network*. If C is a set of binary inequalities then $NET_m(C)$ will denote the minimal BICN corresponding to C. In figure 3.1 the network on the right is the minimal network equivalent to the network on the left.

For the case of weak inequality constraints, an alternative graph representation will also be useful. Let C be a set of weak inequality constraints in variables $x_1, \ldots, x_n$. The *distance graph* associated with C is a directed labelled graph $G = (V, E)$ where $V = \{1, \ldots, n\}$. Node i represents variable x_i and edge (i, j) represents the binary constraints involving x_i and x_j. If there is a constraint $x_j - x_i \leq d_{ij}$ in C then edge $i \rightarrow j$ of the associated distance graph will be labelled by d_{ij}. The concept of *minimal distance graph* can be defined similarly with the concept of minimal network. Given the minimal network associated with a set of weak inequality constraints, it is trivial to construct the associated minimal distance graph (and vice versa). The set of constraints $Constraints(G)$ represented by a distance graph G is defined as for BICN. The distance graph machinery can easily be extended to the case of strict inequalities [KL91, Dil89].

3.4 Consistency of Inequalities

Inequalities in the language dePCL (dePCL-inequalities) have been studied in [DMP89, DMP91, KL91, Dil89]. Inequalities in dePCL (diPCL-inequalities) have been studied in [Cha77]. In the case of diPCL-inequalities, one can concentrate on weak inequalities with no loss of generality: every strict diPCL-inequality $x_i - x_j < a,\ a \in \mathbb{Z}$ can be equivalently written as $x_i - x_j \leq a - 1$. There is also a great deal of work on *qualitative* inequalities i.e., inequalities of the form $x \leq y,\ x < y$ and $x \neq y$ [VK86, VKvB89, vBC90, vB92, GS93].

The following theorem is a "folk" result. It has been mentioned (with or without proof) in

Algorithm DPCONSISTENCY
Input: A BICN N.
Output: YES or NO depending on whether N is consistent or not.

Method:
1. For $k = n$ downto 1 do
2.   For $i, j < k$ such that (i, k) and (k, j) are edges of N do
3. $N_{ij} := N_{ij} \oplus (N_{ik} \otimes N_{kj})$
4.   If $N_{ij} = \emptyset$ then Return NO
5.   EndFor
6. EndFor
7. Return YES

Figure 3.2: Deciding consistency of a BICN

[Pra77, Cha77, Sho81, DMP89, CLR90, Dil89].

Theorem 3.4.1 *A set of weak inequalities is consistent if and only if its distance graph has no negative directed cycles.*

The presence of negative directed cycles can be detected by running the Floyd-Warshall all-pairs shortest path algorithm on the distance graph of the given set of inequalities. Floyd-Warshall decides consistency but also computes the minimal distance graph (and thus the minimal network). The complexity of the algorithm is $O(n^3)$ where n is the number of nodes in the graph (i.e. the number of variables in the given network). The temporal reasoning system Tachyon [SAD93] uses the Bellman-Ford shortest path algorithm to compute the minimal network. Although the worst case complexity of this algorithm is also $O(n^3)$, it has provided a "substantial performance increase over the Floyd-Warshall algorithm, especially when the corresponding graphs are fairly sparse" [SAD93].

If we are not interested in computing the minimal network then algorithm DPCONSISTENCY shown in figure 3.2 can also be used for deciding consistency [DMP91]. The algorithm is applicable to weak and strict inequalities.

Algorithm DPCONSISTENCY makes use of the composition operation [DMP91].

Definition 3.4.1 Let I_1, I_2 be intervals. The *composition* of I_1 and I_2, denoted by $I_1 \otimes I_2$, is defined as follows:

$$I_1 \otimes I_2 = \{z : \exists x \in I_1, \ \exists y \in I_2 \text{ and } x + y = z\}.$$

The operation $\oplus$ is the usual set-theoretic *intersection.*

DPCONSISTENCY visits all nodes of N in the order $n, n-1, \ldots, 1$. When visiting node k, this node is used for tightening the constraint between nodes i and j. The time complexity of the above algorithm is $O(nw^2)$ where w is the *width* of the final network [DMP91].

The width of a network can be defined as follows. The *width* of a network node is the maximum number of edges that lead from that node to nodes with a lower index. The *width* of a network is the maximum width of all nodes. Although in the worst case the width of a network can be $O(n)$, it will usually be much smaller than n. Thus DPCONSISTENCY will outperform Floyd-Warshall in most cases.

It is not difficult to extend the above results to the case of strict dePCL-inequalities. The interested reader can consult [KL91, Dil89] for more details.

Notation 3.4.1 In this section, complexity bounds are calculated in terms of *cardinality* and *number of variables* of constraint sets. The cardinality of a set of constraints C is denoted by $|C|$. The number of variables in a set of constraints C is denoted by $v(C)$.

We are now ready to give a theorem which summarizes the above discussion.

Theorem 3.4.2 *Let C be a set of inequalities. The consistency of C can be decided in $O(v(C)w^2)$ time where $w = O(v(C))$ is the width parameter defined above. The minimal BICN corresponding to C can be computed in $O(v(C)^3)$ time.*

The above results can easily be extended to the case of diTCL- and deTCL-constraints. If C is a set of diTCL constraints, then proposition 3.2.1 tell us that C is satisfiable if and only if $itp(C)$ is satisfiable. Therefore we can decide whether C is consistent by computing $itp(C)$ and using the above algorithms. The complexity bounds for the new algorithms remain unchanged.

3.5 Variable Elimination

In this section we discuss the problem of variable elimination for boolean combinations of constraints. Variable elimination is the algebraic counterpart of the quantifier elimination operation of mathematical logic. Much of the machinery of this section will be used in chapter 6 where we discuss quantifier elimination. Let us start by defining our terminology.

Notation 3.5.1 In the rest of this dissertation, $\overline{o}$ will denote the vector of symbols $(o_1, \ldots, o_n)$. The natural number n will be called the *size* of $\overline{o}$ and will be denoted by $|\overline{o}|$. This notation will be used for vectors of variables but also for vectors of domain elements. Variables will be denoted by x, y, z, t etc. and vectors of variables by $\overline{x}, \overline{y}, \overline{z}, \overline{t}$ etc. If $\overline{x}$ and $\overline{y}$ are vectors of variables then $\overline{x} \setminus \overline{y}$ will denote the vector obtained from $\overline{x}$ by deleting the variables in $\overline{y}$. If $\overline{x}$ is a vector of variables then $\overline{x}^0$ will be a vector of constants of the same size.

When we want to talk about a set of variables (rather than a vector), we will use capital letters X, Y, Z, T etc. If X and Y are sets then $X \setminus Y$ will denote the (set-theoretic) difference of X and Y.

Definition 3.5.1 Let $\mathcal{L}$ be a many-sorted first-order language. The class of $\mathcal{L}$-constraints *admits variable elimination* iff for every boolean combination ϕ of $\mathcal{L}$-constraints in variables $\overline{x}$, and every vector of variables $\overline{z} \subseteq \overline{x}$, there exists a disjunction ϕ' of conjunctions of $\mathcal{L}$-constraints in variables $\overline{x} \setminus \overline{z}$ such that

1. If $\overline{x}^0$ is a solution of ϕ then $\overline{x}^0 \setminus \overline{z}^0$ is a solution of ϕ'.
2. If $\overline{x}^0 \setminus \overline{z}^0$ is a solution of ϕ' then this solution can be extended to a solution $\overline{x}^0$ of ϕ.

Some people might find the above definition overly strong. But requiring ϕ' to be just a boolean combination of $\mathcal{L}$-constraints would turn out to be unsatisfactory for the constraint database models of chapter 4. The reason is very simple: when we eliminate variables, we would have to deal with negations of $\mathcal{L}$-constraints. Similar arguments and definitions appear in [Stu91].

The following definition is needed below.

Definition 3.5.2 Let $\mathcal{L}$ be a many-sorted first-order language. The class of $\mathcal{L}$-constraints is *weakly closed under negation* if the negation of every $\mathcal{L}$-constraint is equivalent to a disjunction of $\mathcal{L}$-constraints.

The following proposition shows that if the class of $\mathcal{L}$-constraints is weakly closed under negation then we can determine whether it also admits variable elimination by considering only conjunctions of $\mathcal{L}$-constraints.

Proposition 3.5.1 *Let $\mathcal{L}$ be a many-sorted first-order language. The class of $\mathcal{L}$-constraints admits variable elimination if*

- *it is weakly closed under negation, and*
- *for every conjunction θ of $\mathcal{L}$-constraints in variables $\overline{x}$ and every vector of variables $\overline{z} \subseteq \overline{x}$, there exists a disjunction θ' of conjunctions of $\mathcal{L}$-constraints in variables $\overline{x} \setminus \overline{z}$ such that*
    1. *If $\overline{x}^0$ is a solution of θ then $\overline{x}^0 \setminus \overline{z}^0$ is a solution of θ'.*
    2. *If $\overline{x}^0 \setminus \overline{z}^0$ is a solution of θ' then this solution can be extended to a solution $\overline{x}^0$ of θ.*

Proof: Let ϕ be a boolean combination of $\mathcal{L}$-constraints. The result can be shown by induction on the structure of ϕ. For the base case, ϕ is an $\mathcal{L}$-constraint and the result is trivial. For the inductive step, we have to consider the following cases:

- ϕ is $\neg\psi$. In this case the result follows from the inductive hypothesis, De Morgan's laws, the distributive laws and the fact that the class of $\mathcal{L}$-constraints is weakly closed under negation.
- ϕ is $\psi_1 \vee \psi_2$. In this case the result follows immediately from the inductive hypothesis.

■

The above proposition provides us with the following algorithm for eliminating variables $\overline{z}$ from a boolean combination ϕ of $\mathcal{L}$-constraints in variables $\overline{x}$:

1. Transform ϕ into a formula where negation applies only to $\mathcal{L}$-constraints (by applying De Morgan's laws and laws for negation).
2. Substitute every negated $\mathcal{L}$-constraint by its equivalent disjunction of $\mathcal{L}$-constraints.
3. Transform ϕ into disjunctive normal form $\theta_1 \vee \cdots \vee \theta_m$ where each θ_i is a conjunction of $\mathcal{L}$-constraints.
4. Perform variable elimination to each θ_i. In other words, substitute each disjunct θ_i of ϕ by the equivalent disjunction of conjunctions of $\mathcal{L}$-constraints in variables $\overline{x} \setminus \overline{z}$.

For this algorithm to be effective, we must know how to perform steps 2 and 4. For most languages of interest step 2 will be obvious. Step 4 will usually be more involved. For any given language $\mathcal{L}$ we need to know how to perform variable elimination for conjunctions of $\mathcal{L}$-constraints. In what follows we will concentrate on variable elimination for sets (i.e., conjunctions) of diPCL- and dePCL-constraints.

For diPCL- or dePCL-inequalities variable elimination can be achieved using Fourier's algorithm which was originally devised for linear inequality constraints over the real numbers [Sch86]. Fourier's algorithm can be summarized as follows. Any weak linear inequality involving a variable x can be written in the form $x \leq r_u$ or $x \geq r_l$ i.e., it gives an upper or a lower bound on x. Thus if we are given two linear inequalities, one of the form $x \leq r_u$ and the other of the form $x \geq r_l$, we can eliminate x and obtain the inequality $r_l \leq r_u$. Obviously, $r_l \leq r_u$ is a logical consequence of the given inequalities. In addition, any solution of $r_l \leq r_u$ can be extended to a solution of the given inequalities (simply by choosing for x any value between the values of r_l and

r_u). Following this observation, Fourier's elimination algorithm forms all pairs $x \leq r_u$ and $x \geq r_l$, eliminates x and returns the resulting constraints. The generalization of this algorithm to strict linear inequalities is obvious.

The specialization of Fourier's algorithm to dePCL-inequalities is straightforward. [KSW90] have showed that Fourier's algorithm gives correct results for diPCL-inequalities as well.[2]

Example 3.5.1 Eliminating variable x_1 from the set of dePCL-inequalities

$$\{x_3 \leq x_1,\ x_5 \leq x_1,\ x_1 - x_2 \leq 2,\ x_4 \leq x_5\}$$

gives

$$\{x_3 - x_2 \leq 2,\ x_5 - x_2 \leq 2,\ x_4 \leq x_5\}.$$

Figure 3.3 gives the variable elimination algorithm PCL-VarElimination. For simplicity, the algorithm assumes that one-variable inequalities are given as two-variable inequalities (with the introduction of an auxiliary variable x_0 which is understood to be equal to zero).

Notation 3.5.2 If o is a syntactic object (term, atomic formula, formula) of a given language then $o[i/j]$ is the same as o except that syntactic object j has been substituted by syntactic object i. This notation is used in algorithm PCL-VarElimination and in many other places in this dissertation.

The same goal could have been achieved using the *adaptive consistency* techniques discussed by [DP88] in the context of finding backtrack-free solutions to binary constraint satisfaction problems. If we want to eliminate variables $x_1, \ldots, x_m$ from a set of inequalities C represented by a BICN N then we could run the algorithm Adaptive-consistency of [DP88, page 18] on N with arguments $x_m, \ldots, x_1$. In the resulting network the set of constraints between variables $x_{m+1}, \ldots, x_n$ ($Constraints(x_{m+1}, \ldots, x_n)$) is equivalent to the set of constraints generated by Fourier's algorithm.

The algorithm DPConsistency of figure 3.2 can be considered as a variable elimination algorithm. The careful reader can notice that DPConsistency generates all constraints that would be generated by PCL-VarElimination if variables X were eliminated in the order $x_n,\ x_{n-1}, \ldots, x_1$.

The following proposition shows that PCL-VarElimination is correct. Its proof for dePCL-constraints is an easy generalization of the previous discussion and is given in theorem 8.4.1 where an extension of dePCL-constraints is covered. For the case of diPCL-constraints this result was originally proved in [KSW90].

Theorem 3.5.1 *Let C be a set of diPCL- or dePCL-constraints in variables $x_1, x_2, \ldots, x_n$ and C' be the set of constraints produced by algorithm* PCL-VarElimination *when x_1 is eliminated. If $(x_1^0, x_2^0, \ldots, x_n^0)$ is a solution of C then $(x_2^0, x_3^0, \ldots, x_n^0)$ is a solution of C'. Conversely, every solution $(x_2^0, x_3^0, \ldots, x_n^0)$ of C' can be extended to a solution $(x_1^0, x_2^0, \ldots, x_n^0)$ of C.*

Remark 3.5.1 In practice algorithm PCL-VarElimination will be augmented with some redundancy checks. In our case redundancy can be classified into three types: trivial, syntactic and semantic redundancy [LM89]. A constraint is called *trivially redundant* if it is a tautology (e.g., the constraint $3 \leq 4$). A constraint $x_i - x_j \leq r$ is called *syntactically redundant* in the set of constraints C if and only if there is a constraint $x_i - x_j \leq r'$ in C such that $r' \leq r$ (e.g., $x - y \leq 8$ is redundant in the set of constraints $\{y - z \leq 6, x - y \leq 5\}$). A constraint is called *semantically*

[2] However Fourier's algorithm does *not* gives correct results for arbitrary linear inequalities over the integers.

Algorithm PCL-VarElimination

Input: A consistent constraint set $C = C_e \cup C_i$, in variables X where C_e is a set of equalities and C_i is a set of inequalities. A set of variables $X \subseteq E$ to be eliminated from C.

Output: A constraint set C' in variables $X \setminus E$ such that $Sol(C') = Proj_{X\setminus E}(Sol(C))$.

Method:

```
C'_e := ∅; C'_i := ∅
For each variable x ∈ E do
    If x appears in equation e_i : y_i − x = r ∈ C_e then
        For each constraint c in C_e − {e_i} do
            C'_e := C'_e ∪ {c[y_i − r/x]}
        EndFor
        For each constraint c in C_i do
            C'_i := C'_i ∪ {c[y_i − r/x]}
        EndFor
        C_e := C'_e; C_i := C'_i; C'_e := ∅; C'_i := ∅
    Else (the variable x appears only in C_i)
        For each constraint x_k − x ≤ (<)r_k in C_i do
            For each constraint x − x_j ≤ (<)r_j in C_i (j ≠ k) do
                C'_i := C'_i ∪ {x_k − x_j ≤ (<)r_k + r_j}
            EndFor
        EndFor
        Add all constraints of C_i which do not contain x to C'_i
        C_e := C'_e; C_i := C'_i
        C'_e := ∅; C'_i := ∅
    EndIf
EndFor
Return C_e ∪ C_i
```

Algorithm diTCL-VarElimination

Input: A consistent set of diTCL-constraints C in variables X.
A set of variables E to be eliminated from C.

Output: A set of diTCL-constraints C' in variables $X \setminus E$
such that $Sol(C') = Proj_{X\setminus E}(Sol(C'))$.

Method:

```
X' := ∅; A := itp(C)
For each variable x ∈ X do
    If x is of sort Z then X' := X' ∪ {x}
    If x is of sort I_Z then X' := X' ∪ {B_x, E_x}
EndFor
A' :=PCL-VarElimination(A, X')
Return pti(A')
```

Figure 3.3: Variable elimination

redundant if and only if it is not trivially or syntactically redundant (e.g., $x - z \leq 14$ is redundant in the set of constraints $\{y - z \leq 6, x - y \leq 5\}$).

Trivial and syntactic redundancy can be checked very cheaply. Checking for semantic redundancy is more expensive. In what follows, we will assume that whenever a variable is eliminated by PCL-VarElimination, trivially redundant or syntactically redundant constraints are not added to C'_e or C'_i.

Let us now investigate the complexity of PCL-VarElimination.

Lemma 3.5.1
If C is a set of inequalities which does not contain trivial or syntactic redundancies then $|C| \leq v(C)^2 + v(C)$.

Proof: Given two variables x_i, x_j, there can be at most two constraints involving x_i and x_j in C (an upper bound and/or a lower bound on $x_i - x_j$). In our case we have $v(C) + 1$ variables including x_0. Thus we can have at most

$$2\begin{pmatrix} v(c)+1 \\ 2 \end{pmatrix} = v(C)^2 + v(C)$$

constraints. ■

Lemma 3.5.2 *Let $C = C_e \cup C_i$ be a set of diPCL- or dePCL-constraints. If m variables are eliminated using algorithm* PCL-VarElimination *and the resulting set is $C' = C'_e \cup C'_i$ then $|C'_e| \leq |C_e|$ and $|C'_i| \leq 2(v(C_i) - m)^2$.*

Proof: When m variables are eliminated we are left with $v(C) - m$ variables. Therefore lemma 3.5.1 and the above remark imply that $|C_i| \leq 2(v(C_i) - m)^2$. The first inequality is obvious. ■

Theorem 3.5.2 *Let $C = C_e \cup C_i$ be a set of diPCL- or dePCL-constraints. We can eliminate m variables from C, using* PCL-VarElimination, *in $O(mv(C_i)^2)$ time.*

Proof: Straightforward consequence of the previous lemma and the above remark. ■

Let us now consider diTCL- and deTCL-constraints. The algorithm diTCL-VarElimination in figure 3.3 is a variable elimination algorithm for diTCL-constraints. The following theorem demonstrates that diTCL-VarElimination is correct.

Theorem 3.5.3 *Let C be a set of diTCL-constraints in variables $x_1, \ldots, x_n, z_1, \ldots, z_m$ and C' be the set of constraints produced by algorithm* diTCL-VarElimination *when variables $x_1, \ldots, x_m$ are eliminated. If $(x_1^0, \ldots, x_n^0, z_1^0, \ldots, z_m^0)$ is a solution of C then $(z_1^0, \ldots, z_m^0)$ is a solution of C'. Conversely, every solution $(z_1^0, \ldots, z_m^0)$ of C' can be extended to a solution $(x_1^0, \ldots, x_n^0, z_1^0, \ldots, z_m^0)$ of C.*

Proof: Let $x_1, \ldots, x_k$ be all the variables of sort $\mathcal{Z}$ among $x_1, \ldots, x_n$ and $x_{k+1}, \ldots, x_n$ be the remaining variables (of sort $\mathcal{I}_{\mathcal{Z}}$). In a similar way, let $z_1, \ldots, z_j$ be all the variables of sort $\mathcal{Z}$ among $z_1, \ldots, z_m$ and $z_{j+1}, \ldots, z_m$ be the remaining variables (of sort $\mathcal{I}_{\mathcal{Z}}$). If

$$\mathbf{ZI_Z} \models C[x_1 \leftarrow x_1^0, \ldots, x_n \leftarrow x_n^0, z_1 \leftarrow z_1^0, \ldots, z_m \leftarrow z_m^0]$$

then proposition 3.2.1 implies

$$\begin{aligned} \mathbf{Z} \models \quad & itp(C)[x_1 \leftarrow x_1^0, \ldots, x_k \leftarrow x_k^0, B_{x_{k+1}} \leftarrow l(x_{k+1}^0), E_{x_{k+1}} \leftarrow r(x_{k+1}^0), \ldots, B_{x_n} \leftarrow l(x_n^0), E_{x_n} \leftarrow r(x_n^0), \\ & z_1 \leftarrow z_1^0, \ldots, z_j \leftarrow z_j^0, B_{z_{j+1}} \leftarrow l(z_{j+1}^0), E_{z_{j+1}} \leftarrow r(z_{j+1}^0), \ldots, B_{z_m} \leftarrow l(z_m^0), E_{z_m} \leftarrow r(z_m^0)]. \end{aligned}$$

Then theorem 3.5.1 implies

$$\mathbf{Z} \models A'[z_1 \leftarrow z_1^0, \ldots, z_j \leftarrow z_j^0, B_{z_{j+1}} \leftarrow l(z_{j+1}^0), E_{z_{j+1}} \leftarrow r(z_{j+1}^0), \ldots, B_{z_m} \leftarrow l(z_m^0), E_{z_m} \leftarrow r(z_m^0)].$$

Finally proposition 3.2.2 gives us

$$\mathbf{ZI_Z} \models pti(A')[z_1 \leftarrow z_1^0, \ldots, z_j \leftarrow z_j^0, z_{j+1} \leftarrow z_{j+1}^0, \ldots, z_m \leftarrow z_m^0].$$

For the converse, let us assume that

$$\mathbf{ZI_Z} \models C'[z_1 \leftarrow z_1^0, \ldots, z_j \leftarrow z_j^0, z_{j+1} \leftarrow z_{j+1}^0, \ldots, z_m \leftarrow z_m^0].$$

Then

$$\mathbf{Z} \models A'[z_1 \leftarrow z_1^0, \ldots, z_j \leftarrow z_j^0, B_{z_{j+1}} \leftarrow l(z_{j+1}^0), E_{z_{j+1}} \leftarrow r(z_{j+1}^0), \ldots, B_{z_m} \leftarrow l(z_m^0), E_{z_m} \leftarrow r(z_m^0)].$$

Now theorem 3.5.1 tells us that there exist $x_1^0, \ldots, x_k^0, \beta_1, \ldots, \beta_{n-k}, \epsilon_1, \ldots, \epsilon_{n-k} \in \mathbb{Z}$ such that

$$\begin{aligned}\mathbf{Z} \models \quad & A[x_1 \leftarrow x_1^0, \ldots, x_k \leftarrow x_k^0, B_{x_{k+1}} \leftarrow \beta_1, E_{x_{k+1}} \leftarrow \epsilon_1, \ldots, B_{x_n} \leftarrow \beta_{n-k}, E_{x_n} \leftarrow \epsilon_{n-k}, \\ & z_1 \leftarrow z_1^0, \ldots, z_j \leftarrow z_j^0, B_{z_{j+1}} \leftarrow l(z_{j+1}^0), E_{z_{j+1}} \leftarrow r(z_{j+1}^0), \ldots, B_{z_m} \leftarrow l(z_m^0), E_{z_m} \leftarrow r(z_m^0)].\end{aligned}$$

Finally

$$\begin{aligned}\mathbf{ZI_Z} \models \quad & C[x_1 \leftarrow x_1^0, \ldots, x_k \leftarrow x_k^0, x_{k+1} \leftarrow \langle \beta_1, \epsilon_1 \rangle, \ldots, x_n \leftarrow \langle \beta_{n-k}, \epsilon_{n-k} \rangle, \\ & z_1 \leftarrow z_1^0, \ldots, z_j \leftarrow z_j^0, z_{j+1} \leftarrow z_{j+1}^0, \ldots, z_m \leftarrow z_m^0].\end{aligned}$$

■

The following result is an easy consequence of theorem 3.5.2.

Theorem 3.5.4 *Let C be a set of diTCL-constraints. We can eliminate m variables from C, using* DITCL-VARELIMINATION, *in $O(mv(C_i)^2)$ time.*

Variable elimination for deTCL-constraints can be achieved similarly.

Example 3.5.2 Let us eliminate variable $i2$ from the following set of deTCL-constraints:

$$i1_R - i1_L \leq 5,\ i2_R - i2_L \geq 2,\ i3_R - i3_L \geq 3,\ i2_L > i1_R,\ i3_L > i2_R.$$

First we translate into dePCL-constraints:

$$E_{i1} - B_{i1} \leq 5,\ E_{i2} - B_{i2} \geq 2,\ E_{i3} - B_{i3} \geq 3,\ B_{i2} > E_{i1},\ B_{i3} > E_{i2},\ B_{i1} < E_{i1},\ B_{i2} < E_{i2},\ B_{i3} < E_{i3}$$

Now we can eliminate variables B_{i2}, E_{i2}. Eliminating variable B_{i2} leaves us with the following set of constraints:

$$E_{i1} - B_{i1} \leq 5,\ E_{i3} - B_{i3} \geq 3,\ B_{i3} > E_{i2},\ B_{i1} < E_{i1},\ B_{i3} < E_{i3},\ E_{i1} - E_{i2} \leq -2.$$

Now if we eliminate E_{i2} we obtain:

$$E_{i1} - B_{i1} \leq 5,\ E_{i3} - B_{i3} \geq 3,\ B_{i1} < E_{i1},\ B_{i3} < E_{i3},\ E_{i1} - B_{i3} < -2.$$

The final result is obtained by translating back into deTCL-constraints:

$$i1_R - i1_L \leq 5,\ i3_R - i3_L \geq 3,\ i1_L < i1_R,\ i3_L < i3_R,\ i1_R - i3_L < -2.$$

3.6 Conclusion

In this chapter we introduced the theories of temporal constraints which will be studied in detail in the rest of the dissertation. We also studied the problems of deciding consistency, computing the minimal constraint network and performing variable elimination.

Temporal constraints and their associated reasoning problems have so far been studied in isolation [All83, VK86, VKvB89, vBC90, vB92, DMP91, KL91, Mei91a, Lad88]. The following chapter presents a hierarchy of database schemes which integrates constraints expressed in some language $\mathcal{L}$ with relational databases. As a special case of this hierarchy, we obtain *indefinite temporal constraint databases* which are more powerful than any of the above temporal constraint representations.

Chapter 4

Foundations of Constraint Databases

This chapter lays the foundations of a theory of *constraint databases* based on the relational model [Mai83]. Following the approach of [JL87, Mah93], we develop three database schemes: $\mathbf{M}_{\mathcal{L}}$-relational databases, $\mathcal{L}$-constraint databases and indefinite $\mathcal{L}$-constraint databases. The language $\mathcal{L}$, the parameter, defines the constraint alphabet and $\mathbf{M}_{\mathcal{L}}$ is the structure over which $\mathcal{L}$-constraints will be interpreted. The last two schemes are interesting from a practical point of view. $\mathbf{M}_{\mathcal{L}}$-relational databases simply serve as a theoretical tool for providing semantics for the other two schemes. The scheme of $\mathcal{L}$-constraint databases has been originally proposed in [KKR90].

The most important contributions of this chapter are the following:

- We develop an algebraic query language for the $\mathcal{L}$-constraint database scheme. The query languages of [KSW90, Kou93a, Kou93b] are instances of our scheme. This language can be used as the basis of practical bottom-up query evaluation algorithms. Alternative algorithms like the ones considered in [KKR90] do not appear to have practical implementations; they simply achieve better theoretical bounds.

- We extend the scheme of $\mathcal{L}$-constraint databases to include indefinite information using the methodology of [IL84, Gra89]. We define declarative and procedural query languages for the new scheme and study the semantics of query evaluation.

As examples of the above schemes, we will consider $\mathcal{L}$-constraint databases and indefinite $\mathcal{L}$-constraint databases where $\mathcal{L}$ is diPCL, dePCL, diTCL or deTCL. We will collectively refer to these models as *temporal constraint databases* and *indefinite temporal constraint databases* respectively.[1] Because our developments take place in the context of the above abstract schemes, our results are applicable to many other domains.

This chapter is organized as follows. In section 4.1 we present an example in the context of indefinite deTCL-constraint databases. This example is intended to motivate the subsequent theoretical development. In section 4.2 we develop a hierarchy of representation schemes: $\mathbf{M}_{\mathcal{L}}$-relational databases, $\mathcal{L}$-constraint databases and indefinite $\mathcal{L}$-constraint databases. Declarative and procedural query languages for these representation schemes are presented in sections 4.3, 4.4 and 4.5. In section 4.6 we study the semantics of query evaluation in the proposed algebraic query

[1] In section 5.2 we will modify these definitions to suit the alternative schemes developed in chapter 5.

languages. In section 4.7 we briefly discuss the issue of canonical forms for indefinite $\mathcal{L}$-constraint databases. In section 4.8 we present some examples of queries in the context of deTCL and dePCL-constraint databases. Finally, section 4.9 discusses the translation of queries expressed in the declarative query language we developed into queries expressed in the corresponding procedural query languages.

4.1 An Example

Let us assume that we want to design a database for a transportation company. This database will be used by the scheduling personnel to keep track of orders, vehicles, loads, trips and deliveries. The following $\mathbf{QI_Q}$-*relations* can be part of this database.

TRIP

VEH	FROM	TO	TIME
V3	Athens	Patra	$\langle 5, 8\rangle$
V3	Patra	Athens	$\langle 9, 12\rangle$

ON_SERVICE

VEH	CITY	TIME	CON
V1	Athens	$i1$	$9 \leq i1_L,\ i1_R \leq 15$
V2	Athens	$i2$	$10 \leq i2_L,\ i2_R \leq 14$

The $\mathbf{QI_Q}$-relation TRIP is exactly like a standard relation except that the values of attribute TIME are *not* atomic data values but rational intervals (i.e., elements of the domain of sort $\mathcal{I}_\mathcal{Q}$ in the structure $\mathbf{QI_Q}$ defined in section 3.2.2). In general this chapter will deal with $\mathcal{M}$-relations where $\mathcal{M}$ is a first-order structure. $\mathcal{M}$-relations have two kinds of attributes: data attributes (as found in standard relations) and attributes whose values are elements of the domains of $\mathcal{M}$.

The $\mathbf{QI_Q}$-relation TRIP represents *definite* and *finite* information about trip *events* in the world. The first tuple of TRIP states that the trip of vehicle V3 from Athens to Patra occured over the interval $\langle 5, 8\rangle$. Let us now consider the *fact* "vehicle V1 is on service for interval $\langle 9, 15\rangle$". This says implicitly that V1 is also on service for all subintervals of interval $\langle 9, 15\rangle$. How can we represent this sort of infinite information? In the spirit of [KSW90, KKR90] we will use *deTCL-constraint relations* for this job. deTCL-constraint relations are essentially *finite representations* of unrestricted (i.e., finite or infinite) $\mathbf{QI_Q}$-relations.

In general this chapter considers $\mathcal{L}$-constraint relations which are finite representations of unrestricted $\mathbf{M}_\mathcal{L}$-relations. $\mathcal{L}$-constraint relations extend $\mathbf{M}_\mathcal{L}$-relations in the following way:

- There is a new type of attributes called *variable attributes*. These attributes are allowed to have a special kind of variables, called *u-variables*, as values.

- Every tuple is tagged with a conjunction of $\mathcal{L}$-constraints on its u-variables. This conjunction is called the *local condition* of the tuple. In our examples conjuncts will be separated by commas instead of the usual "$\wedge$" operator.

For example, in deTCL-constraint relations like ON_SERVICE attributes are allowed to have variables of sort $\mathcal{Q}$ or $\mathcal{I}_\mathcal{Q}$ as values. As we saw in section 3.2.2, deTCL-constraints can be used to express qualitative or quantitative properties of points and intervals. In the first tuple of relation

ON_SERVICE

VEH	CITY	TIME	CON
V1	Athens	i1	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R$
V2	Athens	i2	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R$

TRIP

VEH	FROM	TO	TIME	CON
V3	Athens	Patra	$\langle 5, 8\rangle$	true
V3	Patra	Athens	$\langle 9, \nu\rangle$	true

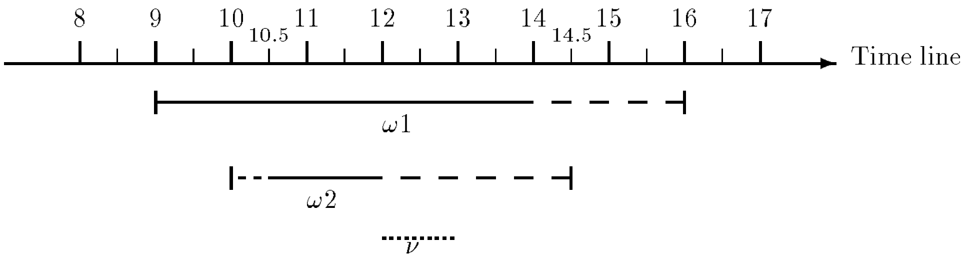

Figure 4.1: Indefinite deTCL-constraint relations

ON_SERVICE the variable $i1$ represents the interval during which vehicle V1 is in Athens for service. The local condition of this tuple constrains interval $i1$ to be equal to or be contained in interval $\langle 9, 15\rangle$. This containment relation is expressed by order constraints on the endpoints of the intervals $i1$ and $\langle 9, 15\rangle$.

A tuple of a deTCL-constraint relation must be understood as an *implication* whose antecedent is the local condition. In this implication u-variables are universally quantified. Therefore a deTCL-constraint tuple is a *finite representation* of an unrestricted set of $\mathbf{QI_Q}$-relational tuples. For example, the first tuple of relation ON_SERVICE asserts that vehicle V1 is in Athens for service for interval $\langle 9, 15\rangle$ and all of its subintervals. This is an example of *definite* and *infinite* temporal information about some *fact* in the world. The attribute TIME of relation ON_SERVICE is semantically equivalent to *valid time* in traditional historical models [Sno92].

Let us now turn to *indefinite* information. An example of such a situation is illustrated by the indefinite deTCL-constraint relations in figure 4.1. *Indefinite deTCL-constraint relations* extend deTCL-constraint relations by allowing the representation of indefinite information in the spirit of *conditional tables* [IL84, Gra89]. Indefinite deTCL-constraint relations are exactly like *deTCL-constraint relations* but they can also involve *e-variables* of sort $\mathcal{Q}$ or $\mathcal{I}_\mathcal{Q}$. E-variables are semantically equivalent to the *marked nulls* of conditional tables. They represent values that *exist* but are not known precisely. A boolean combination of deTCL-constraints called the *global condition* asserts all known constraints on these e-variables.

In our example, the e-variables are $\omega 1, \omega 2$ of sort $\mathcal{I}_\mathcal{Q}$ and ν of sort $\mathcal{Q}$.[2] The global condition

$$G:\ \omega 1_L = 9,\ 14 \leq \omega 1_R \leq 16,\ 10 \leq \omega 2_L \leq 10.5,\ 2 \leq \omega 2_R - \omega 2_L \leq 4,\ 12 \leq \nu \leq 13$$

[2] A note on typography: Interval u-variables will be denoted by $i, i1, i2, \ldots$. Point u-variables will be denoted by $p, p1, p2, \ldots$. Interval e-variables will be denoted by $\omega, \omega 1, \omega 2, \ldots$. Point e-variables will be denoted by $\nu, \nu 1, \nu 2, \ldots$.

holds for these variables. The variables $\omega 1$ and $\omega 2$ denote time intervals whose duration is not known precisely. The variable ν denotes a time point whose position in the time line is not known exactly. The global condition G states that (i) interval $\omega 1$ starts at time 9 and ends at some time between 14 and 16, (ii) interval $\omega 2$ starts at some time between 10 and 10.5 and its duration is 2 to 4 time units, and (iii) point ν is between 12 and 13. This information is also presented graphically in figure 4.1. Intervals are represented by line segments. Solid line segments depict the extent of an interval about which we have definite information. Dashed line segments depict the extent of an interval about which we have indefinite information. Similar conventions are followed for time points.

In general this chapter will consider *indefinite $\mathcal{L}$-constraint relations.* Indefinite $\mathcal{L}$-constraint relations are syntactic devices for the representation of indefinite information (finite or infinite). Each indefinite $\mathcal{L}$-constraint relation is semantically equivalent to a set of possible worlds which capture all the possible ways the domain of application could have been according to our indefinite information. Each possible world can be represented by an $\mathcal{L}$-constraint relation. One can find all the possible worlds represented by an indefinite $\mathcal{L}$-constraint relation as follows:

- Find an assignment to the e-variables which satisfies the global condition.
- Substitute these values for the e-variables in the indefinite $\mathcal{L}$-constraint relation.

For example, the deTCL-constraint relation ON_SERVICE in figure 4.1 represents one of the possible worlds corresponding to indefinite deTCL-constraint relation ON_SERVICE.

Let us now consider the query "Find all vehicles that are on service throughout the first hour of the afternoon". Using *relational calculus with deTCL-constraints* as our query language, this query can be phrased as follows:

$$\{ANS1(VEH), x/\mathcal{D} : \ (\exists i/\mathcal{I}_{\mathcal{Q}})(\exists c/\mathcal{D})(ON\_SERVICE(x,c,i) \ \wedge \ i_L \leq 12 \ \wedge \ i_R \geq 13)\}.$$

In general one of our query languages will be *relational calculus with $\mathcal{L}$-constraints* (see section 4.3). Every expression in relational calculus with $\mathcal{L}$-constraints has two parts. The first part gives the name and the scheme of an indefinite $\mathcal{L}$-constraint relation where the answer to the query will be stored. The second part is a declarative specification of what the user wants in a first-order language which is the union of relational calculus and $\mathcal{L}$ [KKR90]. In our case, we want all values x such that there exist values i and c such that x is on service in city c throughout interval i. Interval i should also *contain* interval $\langle 12, 13\rangle$ (the first hour of the afternoon in our time-line). The sort of x and c is $\mathcal{D}$ i.e., they represent data values. The sort of i is $\mathcal{I}_{\mathcal{Q}}$ i.e., it represents a rational interval.

What is the answer to the above query ? If we examine the database on figure 4.1, we can see that vehicle V1 satisfies the qualification of the query. The situation is more involved with V2. Observe that it is possible that $\omega 2$ extends until time 13. In this case V2 is in the answer as well. But what if $\omega 2$ does not extend until time 13?

The above situation is quite typical for a query posed to a database with indefinite information. There are objects that qualify as answers to the posed query in all possible worlds represented by the database. There are also objects that qualify in one possible world but do not qualify in another. Therefore the answer to a query must be *conditional* [IL84]. For every object that qualifies as an answer *only in a subset* of the possible worlds represented by the database, the query answering procedure should also provide a *condition* characterizing this possible world subset.

In our example, vehicle V1 qualifies in all possible worlds represented by the database. Vehicle V2 qualifies only in the possible worlds where interval $\omega 2$ extends until time 13 or later. Therefore the answer to this query should be the following indefinite dePCL-constraint relation ANS1.

ANS1

VEH	CON
V1	*true*
V2	$\omega 2_R \geq 13$

The global condition associated with ANS1 is the initial constraint G. Queries do not alter the global condition associated with the input database.

The utility of the local conditions must be obvious now. If a local condition contains u-variables, it provides a finite representation of an unrestricted set of tuples (e.g., witness indefinite deTCL-constraint relation ON_SERVICE). If it contains only e-variables, it is a condition under which its associated tuple belongs in the relation. In ANS1 the condition for tuple (V1) is *true*. The condition for tuple (V2) is $\omega 2_R \geq 13$. The set of possible worlds represented by ANS1 is given by the set of $\mathbf{QI_Q}$-relations $\left\{ \begin{array}{|c|}\hline \text{V1} \\ \hline \end{array}, \begin{array}{|c|}\hline \text{V1} \\ \text{V2} \\ \hline \end{array} \right\}$. Each one of these relations represents one set of vehicles that are possibly on service throughout the first hour of the afternoon.

In some cases the user might know that the information in the database is indefinite. Thus he (or she) might wish to find all tuples that *possibly* or *certainly* satisfy some qualification. For example, let us consider the query "Find all vehicles that are possibly on service throughout the first hour of the afternoon". In this case our query language will be *modal* relational calculus with deTCL-constraints:

$$\{ANS2(VEH), x/\mathcal{D} : \ \diamond(\exists i/\mathcal{I}_Q)(\exists c/\mathcal{D})(ON\_SERVICE(x,c,i) \ \wedge \ i_L \leq 12 \ \wedge \ i_R \geq 13)\}.$$

This query uses the modal operator $\diamond$ for *possibility*. The answer to this query is the following indefinite deTCL-constraint relation ANS2.

ANS2

VEH	CON
V1	*true*
V2	*true*

Now the constraint for V2 is true indicating that *unconditionally* V2 is possibly on service throughout the first hour of the afternoon. The relations returned as answers to queries involving $\diamond$ do *not* contain e-variables. Thus they represent a *single* possible world.

We can also pose queries asking for all tuples that *certainly* satisfy a qualification. In this case, we have to use the modal operator $\Box$ for *necessity*. Let us consider the query "Find all times during which vehicles are certainly on service; output vehicle identification and time". This query can be phrased as follows:

$$\{ANS3(VEH), x/\mathcal{D}, i/\mathcal{I}_Q : \ \Box((\exists c/\mathcal{D})ON\_SERVICE(x,c,i))\}.$$

The answer to this query is the following indefinite deTCL-constraint relation ANS3:

ANS3

VEH	TIME	CON
V1	$i1$	$i1_L \leq 10, \ i1_R \leq 14$
V2	$i2$	$i2_L \leq 10.5, \ i2_R \leq 12$

Similarly with ANS2 this indefinite deTCL-constraint relation does not contain e-variables. Thus it also represents a single possible world.

This discussion must have given the reader the basic ideas of our proposal. Let us now turn to formal definitions.

4.2 A Hierarchy of Representation Schemes

Throughout this section $\mathcal{L}$ will denote a many-sorted first-order language and $\mathbf{M}_{\mathcal{L}}$ the intended $\mathcal{L}$-structure. We will assume that the class of $\mathcal{L}$-constraints admits variable elimination and is weakly closed under negation.

For each sort $s \in sorts(\mathcal{L})$, let U_s be a countably infinite set of *variable attributes* of sort s. The set of all variable attributes, denoted by $U_{\mathcal{V}}$, is $\bigcup_{s \in sorts(\mathcal{L})} U_s$. Let us also assume the existence of a set $U_{\mathcal{D}}$, the set of *regular attributes*. The set of all attributes, denoted by $\mathcal{U}$, is $U_{\mathcal{D}} \cup U_{\mathcal{V}}$. The sort of attribute A will be denoted by $sort(A)$.

With each $A \in \mathcal{U}$ we associate a set of values $dom(A)$, called the *domain* of A. We assume the existence of a set of *atomic data values* Dom which is called the *data domain*. If $A \in U_{\mathcal{D}}$ then $dom(A) = Dom$. If $A \in U_{\mathcal{V}}$ then $dom(A) = dom(s, \mathbf{M}_{\mathcal{L}})$.

Definition 4.2.1 A *relation scheme* R is a finite subset of $\mathcal{U}$. If R is a scheme $\{A_1, \ldots, A_n\}$ it will also be written as $R(A_1, \ldots, A_n)$. If R is a scheme then R_d and R_v will denote the set of regular and variable attributes in R respectively.

Definition 4.2.2 Let R be a relation scheme. An $\mathbf{M}_{\mathcal{L}}$*-relational tuple* t over scheme R is a mapping from R to $Dom \cup \bigcup_{s \in sorts(\mathcal{L})} dom(s, \mathbf{M}_{\mathcal{L}})$ such that $t(A) \in Dom$ if $A \in R_d$ and $t(A) \in dom(sort(A), \mathbf{M}_{\mathcal{L}})$ if $A \in R_v$.

Definition 4.2.3 Let R be a relation scheme. An $\mathbf{M}_{\mathcal{L}}$*-relation* r over scheme R is an *unrestricted* set of $\mathbf{M}_{\mathcal{L}}$-relational tuples over R.

The scheme of an $\mathbf{M}_{\mathcal{L}}$-relation r will be denoted by $sch(r)$. The set of all $\mathbf{M}_{\mathcal{L}}$-relations over scheme R will be denoted by $\mathbf{M}_{\mathcal{L}}$-$REL(R)$.

For every $s \in sorts(\mathcal{L})$, we assume the existence of two disjoint countably infinite sets of *variables*: the set of *u-variables* $UVAR^s_{\mathcal{L}} \subseteq VAR^s_{\mathcal{L}}$ and the set of *e-variables* $EVAR^s_{\mathcal{L}} \subseteq VAR^s_{\mathcal{L}}$. Let $UVAR_{\mathcal{L}}$ and $EVAR_{\mathcal{L}}$ denote $\bigcup_{s \in sorts(\mathcal{L})} UVAR^s_{\mathcal{L}}$ and $\bigcup_{s \in sorts(\mathcal{L})} EVAR^s_{\mathcal{L}}$ respectively. The intersection of the sets $UVAR_{\mathcal{L}}$ and $EVAR_{\mathcal{L}}$ with the domains of attributes is empty.

Let us now define constraint tuples and relations [KKR90].

Definition 4.2.4 Let R be a relation scheme. An $\mathcal{L}$*-constraint tuple* t over scheme R is a mapping from $R \cup \{CON\}$ to $Dom \cup UVAR_{\mathcal{L}} \cup WFF(\mathcal{L})$ such that (i) $t(A) \in Dom$ for each $A \in R_d$, (ii) $t(A) \in UVAR^{sort(A)}_{\mathcal{L}}$ for each $A \in R_v$, (iii) $t(A_i)$ is different than $t(A_j)$ for all distinct $A_i, A_j \in R_v$, (iv) $t(CON)$ is a conjunction of $\mathcal{L}$-constraints and (v) the variables of $t(CON)$ are included in $\{t(A) : A \in R_v\}$.

If t is an $\mathcal{L}$-constraint tuple then $t(CON)$ is called the *local condition* of the tuple t while $t(R)$ is called the *proper part* of t. The proper part of an $\mathcal{L}$-constraint tuple t is further divided into the *regular part* $t(R_d)$ and the *variable part* $t(R_v)$.

Definition 4.2.5 Let R be a relation scheme. An $\mathcal{L}$*-constraint relation* r over R is a *finite* set of $\mathcal{L}$-constraint tuples over R.

The scheme of an $\mathcal{L}$-constraint relation r will be denoted by $sch(r)$. The set of all $\mathcal{L}$-constraint relations over scheme R will be denoted by $\mathcal{L}$-$CREL(R)$.

Let us now define indefinite constraint tuples and relations.

Definition 4.2.6 Let R be a relation scheme. An *indefinite $\mathcal{L}$-constraint tuple* t over scheme R is a mapping from $R \cup \{CON\}$ to $Dom \cup UVAR_{\mathcal{L}} \cup WFF(\mathcal{L})$ such that (i) $t(A) \in Dom$ for each $A \in R_d$, (ii) $t(A) \in UVAR_{\mathcal{L}}^{sort(A)}$ for each $A \in R_v$, (iii) $t(A_i)$ is different than $t(A_j)$ for all distinct $A_i,\ A_j \in R_v$, (iv) $t(CON)$ is a conjunction of $\mathcal{L}$-constraints and (v) the free variables of $t(CON)$ are included in $\{t(A) :\ A \in R_v\} \cup EVAR_{\mathcal{L}}$.

For indefinite tuples, the notions of local condition, proper part, regular part and variable part are defined as for definite ones.

Definition 4.2.7 Let R be a relation scheme. An *indefinite $\mathcal{L}$-constraint relation* over scheme R is a *finite* set of indefinite $\mathcal{L}$-constraint tuples over R. Each indefinite $\mathcal{L}$-constraint relation r is associated with a boolean combination of $\mathcal{L}$-constraints $G(r)$. $G(r)$ is called the *global condition* of r.

The scheme of an indefinite $\mathcal{L}$-constraint relation r will be denoted by $sch(r)$. The set of all indefinite $\mathcal{L}$-constraint relations over scheme R will be denoted by $\mathcal{L}$-$ICREL(R)$.

We will now define database schemes and databases.

Definition 4.2.8 A *database scheme* $\widetilde{R}$ is a sequence $(R_1, \ldots, R_n)$ where each $R_j,\ j = 1, \ldots, n$ is a relation scheme.

Definition 4.2.9 An *indefinite $\mathcal{L}$-constraint database* $\widetilde{r}$ over scheme $(R_1, \ldots, R_n)$ is a sequence $(r_1, \ldots, r_n)$, where each r_j is an indefinite $\mathcal{L}$-constraint relation over scheme R_j. Each indefinite $\mathcal{L}$-constraint database $\widetilde{r}$ is associated with a boolean combination of $\mathcal{L}$-constraints $G(\widetilde{r})$. $G(\widetilde{r})$ is called the *global condition* of $\widetilde{r}$. By definition $G(r_j) = G(\widetilde{r})$, for all $j,\ 1 \leq j \leq n$. In other words, the global condition is common to all relations in an indefinite database $\widetilde{r}$.

The *scheme* of a database $\widetilde{r}$ will be denoted by $sch(\widetilde{r})$. The set of all indefinite $\mathcal{L}$-constraint databases over scheme $\widetilde{R}$ is denoted by $\mathcal{L}$-$ICDB(\widetilde{R})$. The set of all indefinite $\mathcal{L}$-constraint databases (over any scheme) is denoted by $\mathcal{L}$-$ICDB$.

We can similarly define *$\mathbf{M}_{\mathcal{L}}$-relational databases* (as sequences of $\mathbf{M}_{\mathcal{L}}$-relations) and *$\mathcal{L}$-constraint databases* (as sequences of $\mathcal{L}$-constraint relations). The set of all $\mathbf{M}_{\mathcal{L}}$-relational databases over scheme $\widetilde{R}$ is denoted by $\mathbf{M}_{\mathcal{L}}$-$RDB(\widetilde{R})$. The set of all $\mathbf{M}_{\mathcal{L}}$-relational databases (over any scheme) is denoted by $\mathbf{M}_{\mathcal{L}}$-RDB. The set of all $\mathcal{L}$-constraint databases over scheme $\widetilde{R}$ is denoted by $\mathcal{L}$-$CDB(\widetilde{R})$. The set of all $\mathcal{L}$-constraint databases (over any scheme) is denoted by $\mathcal{L}$-CDB.

4.2.1 Closed-World Semantics

Let us now give semantics to the concepts developed above. We will first define two special kinds of valuations.[3]

Definition 4.2.10 An *e-valuation* in $\mathbf{M}_{\mathcal{L}}$ is a valuation whose domain is restricted to the set $EVAR_{\mathcal{L}}$. Similarly, a *u-valuation* in $\mathbf{M}_{\mathcal{L}}$ is a valuation whose domain is restricted to the set $UVAR_{\mathcal{L}}$.

The symbols $Val^e_{\mathbf{M}_{\mathcal{L}}}$ and $Val^u_{\mathbf{M}_{\mathcal{L}}}$ will denote the set of e-valuations and u-valuations in $\mathbf{M}_{\mathcal{L}}$ respectively.

[3] The concept of valuation is defined in section 3.1.

The *concatenation* of a u-valuation u and an e-valuation v in $\mathbf{M}_{\mathcal{L}}$ is a function

$$u;v : EVAR_{\mathcal{L}} \cup UVAR_{\mathcal{L}} \rightarrow \bigcup_{s \in sorts(\mathcal{L})} dom(s, \mathbf{M}_{\mathcal{L}})$$

such that $(u;v)(x) = u(x)$ if $x \in UVAR_{\mathcal{L}}$ and $(u;v)(x) = v(x)$ if $x \in EVAR_{\mathcal{L}}$.

The result of applying an e-valuation v to an indefinite $\mathcal{L}$-constraint relation r over R will be denoted by $v(r)$. $v(r)$ is an $\mathcal{L}$-constraint relation over R obtained from r by substituting each e-variable ω of r by the constant symbol whose denotation in structure $\mathbf{M}_{\mathcal{L}}$ is $v(\omega)$.

The result of applying a u-valuation of $\mathbf{M}_{\mathcal{L}}$ to the proper part of a tuple can be defined as follows. If t is an $\mathcal{L}$-constraint tuple on scheme R and u is a u-valuation then $u(t)$ is an $\mathbf{M}_{\mathcal{L}}$-relational tuple over R. In addition, for each $A \in R$, $u(t)(A) = u(t(A))$ if $t(A) \in UVAR_{\mathcal{L}}$ and $u(t)(A) = t(A)$ otherwise.

The semantics of an $\mathcal{L}$-constraint relation is given by the function *points* [KKR90]. *points* takes as argument an $\mathcal{L}$-constraint relation r over R and returns the $\mathbf{M}_{\mathcal{L}}$-relation over R which is finitely represented by r:

$$points(r) = \{u(t) : \ t \in r, \ u \in Val^u_{\mathbf{M}_{\mathcal{L}}} \text{ and } \mathbf{M}_{\mathcal{L}} \models t(CON)[u]\}.$$

Let us now define the function *rep* which gives the set of $\mathcal{L}$-constraint relations represented by an indefinite $\mathcal{L}$-constraint relation r over scheme R:

$$rep(r) = \{v(r) \in \mathcal{L}\text{-}CREL(R) : \text{ there exists } v \in Val^e_{\mathbf{M}_{\mathcal{L}}} \text{ s.t. } \mathbf{M}_{\mathcal{L}} \models G(r)[v]\}$$

The *semantics* of an indefinite $\mathcal{L}$-constraint relation r over scheme R is given by the following set of relations:

$$sem(r) = \{points(r_1) \in \mathbf{M}_{\mathcal{L}}\text{-}REL(R) : \ r_1 \in rep(r)\}.$$

The above definition implies that indefinite relations are interpreted in a *closed-world* fashion with respect to their data part. In other words, they are assumed to represent all facts relevant to an application domain. However the exact value of any variable attribute of these facts may not be known precisely.

Note that $sem(r) = \emptyset$ iff $G(r)$ is inconsistent. Let us then assume that $G(r)$ is consistent and for each valuation $v \in Val^e_{\mathbf{M}_{\mathcal{L}}}$ such that $\mathbf{M}_{\mathcal{L}} \models G(r)[v]$, there is no tuple $t \in r$ and valuation $u \in Val^u_{\mathbf{M}_{\mathcal{L}}}$ such that $\mathbf{M}_{\mathcal{L}} \models t(CON)[u;v]$. Then $sem(r) = \{\emptyset\}$.

The functions *rep*, *points* and *sem* can be extended to databases in the following way. If $\widetilde{r} = (r_1, \ldots, r_n)$ is an indefinite $\mathcal{L}$-constraint database over scheme $\widetilde{R} = (R_1, \ldots, R_n)$ then

$$rep(\widetilde{r}) = \{v(\widetilde{r}) \in \mathcal{L}\text{-}CDB(\widetilde{R}) : \text{ there exists valuation } v \in Val^e_{\mathbf{M}_{\mathcal{L}}} \text{ s.t. } \mathbf{M}_{\mathcal{L}} \models G(\widetilde{r})[v]\}.$$

If $\widetilde{r} = (r_1, \ldots, r_n)$ is an $\mathcal{L}$-constraint database over $\widetilde{R}$ then $points(\widetilde{r})$ is an $\mathbf{M}_{\mathcal{L}}$-relational database over $\widetilde{R}$ which is defined as follows:

$$points(\widetilde{r}) = (points(r_1), \ldots, points(r_n)).$$

The semantics of an indefinite $\mathcal{L}$-constraint database $\widetilde{r}$ over scheme $\widetilde{R}$ is the following set of relational databases:

$$sem(\widetilde{r}) = \{points(\widetilde{r_1}) \in \mathbf{M}_{\mathcal{L}}\text{-}RDB(\widetilde{R}) : \ \widetilde{r_1} \in rep(\widetilde{r})\}.$$

Note also that $sem(\widetilde{r}) = \emptyset$ iff $G(\widetilde{r})$ is inconsistent. Let us then assume that $G(\widetilde{r})$ is consistent and for each valuation $v \in Val^e_{\mathbf{M}_{\mathcal{L}}}$ such that $\mathbf{M}_{\mathcal{L}} \models G(\widetilde{r})[v]$ there is no relation $r \in \widetilde{r}$, tuple $t \in r$ and valuation $u \in Val^u_{\mathbf{M}_{\mathcal{L}}}$ such that $\mathbf{M}_{\mathcal{L}} \models t(CON)[u;v]$. Then $sem(\widetilde{r}) = (\emptyset, \ldots, \emptyset)$.

Remark 4.2.1 Notice that constants and e-variables are not allowed as values of variable attributes of tuples. In addition, the u-variables of a tuple are assumed to be different from each other. These are not real restrictions of our model; they only make some of the definitions simpler. Whenever an $\mathcal{L}$-constraint relation which does not obey these rules is encountered, we can transform it to an equivalent one which respects these rules by adding appropriate $\mathcal{L}$-constraints to its tuples.

4.3 Declarative Query Languages

In this section we present a declarative query language for each of the kinds of databases discussed in section 4.2.

Let $\mathcal{L}$ be a given language and $\widetilde{R}$ a database scheme. The language $\mathcal{L}^+(\widetilde{R})$ is the union of two languages: the first-order calculus with equality over predicate vocabulary $\widetilde{R}$ and the language $\mathcal{L}$. More formally, $\mathcal{L}^+(\widetilde{R})$ is defined by extending $\mathcal{L}$ as follows:

- A new sort $\mathcal{D}$ is introduced. The set of constants of sort $\mathcal{D}$ is *Dom* (the data domain introduced in section 4.2). A countably infinite set of variables of sort $\mathcal{D}$ and an equality symbol $=_{\mathcal{D}}$ are also introduced.
- For each scheme $R(C_1, \ldots, C_m)$ in $\widetilde{R}$, an m-place predicate symbol R of sort

  $$sort(C_1) \times \cdots \times sort(C_m)$$

  is introduced. These will be called *database predicate symbols.*

$\mathcal{L}^+(\widetilde{R})$ will simply be written as $\mathcal{L}^+$ when the database predicate vocabulary is of no importance. The well-formed formulas of $\mathcal{L}^+(\widetilde{R})$ are defined in the usual way.

A many-sorted $\mathcal{L}^+(\widetilde{R})$-structure $\mathbf{M}_{\mathcal{L}^+} = (\mathbf{M}_{\mathcal{L}}, Dom, \widetilde{r})$ interprets the symbols of $\mathcal{L}^+(\widetilde{R})$ as follows:

- To the symbols of $\mathcal{L}^+(\widetilde{R})$ which are also symbols of $\mathcal{L}$, $\mathbf{M}_{\mathcal{L}^+}$ makes the same assignments as the structure $\mathbf{M}_{\mathcal{L}}$.
- To the sort $\mathcal{D}$, $\mathbf{M}_{\mathcal{L}^+}$ assigns the set *Dom*.
- To every constant c of sort $\mathcal{D}$, $\mathbf{M}_{\mathcal{L}^+}$ assigns $c \in Dom$.
- To the finite sequence $\widetilde{p}$ of database predicate symbols, $\mathbf{M}_{\mathcal{L}^+}$ assigns the sequence of $\mathbf{M}_{\mathcal{L}}$-relations $\widetilde{r}$.

The expression $sorts(\mathcal{L}^+(\widetilde{R}))$ will denote the set $sorts(\mathcal{L}) \cup \{\mathcal{D}\}$.

4.3.1 Relational Calculus with $\mathcal{L}$-Constraints

The declarative query language for $\mathbf{M}_{\mathcal{L}}$-relational and $\mathcal{L}$-constraint databases will be relational calculus with $\mathcal{L}$-constraints. This query language was proposed originally by [KKR90].

Definition 4.3.1 Let $\widetilde{R}$ be a database scheme and $R(C_1, \ldots, C_m)$ be a relation scheme. An expression over $\widetilde{R}$ of *relational calculus with $\mathcal{L}$-constraints* is

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \; : \; \phi(x_1, \ldots, x_m)\}$$

where $s_i \in sorts(\mathcal{L}^+(\widetilde{R})$ is the sort of C_i, ϕ is a well-formed formula of $\mathcal{L}^+(\widetilde{R})$ and $x_1, \ldots, x_m$ are the only free variables of ϕ.

Expressions of relational calculus with $\mathcal{L}$-constraints define functions from $\mathbf{M}_{\mathcal{L}}$-relational databases to $\mathbf{M}_{\mathcal{L}}$-relations as follows.

Definition 4.3.2 Let $\widetilde{R}$ be a database scheme and f be the following expression over $\widetilde{R}$ in relational calculus with $\mathcal{L}$-constraints:

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \phi(x_1, \ldots, x_m)\}.$$

Let $\widetilde{r}$ be an $\mathbf{M}_{\mathcal{L}}$-relational database over $\widetilde{R}$. The *value* of f on input $\widetilde{r}$, denoted by $f(\widetilde{r})$, is the $\mathbf{M}_{\mathcal{L}}$-relation

$$\{(a_1, \ldots, a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \ (\mathbf{M}_{\mathcal{L}}, Dom, \widetilde{r}) \models \phi(a_1, \ldots, a_m)\}.$$

Notice that in the definition of expressions in relational calculus with $\mathcal{L}$-constraints the number of variables (m) can be zero. Then ϕ is a *sentence* of $\mathcal{L}^+(\widetilde{R})$ and we have a *closed* or *yes/no* expression. The value of a yes/no expression over any $\mathbf{M}_{\mathcal{L}}$-relational database can be $\{()\}$ or $\{\}$. The value $\{()\}$ denotes the answer "yes" while $\{\}$ denotes the answer "no". The situation is the same for yes/no expressions in the rest of the query languages defined in this section.

Expressions of relational calculus with $\mathcal{L}$-constraints can also be understod as defining functions from $\mathcal{L}$-constraint databases to $\mathbf{M}_{\mathcal{L}}$*-relations* as follows.

Definition 4.3.3 Let $\widetilde{R}$ be a database scheme and f be the following expression over $\widetilde{R}$ in relational calculus with $\mathcal{L}$-constraints:

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \phi(x_1, \ldots, x_m)\}.$$

Let $\widetilde{r}$ be an $\mathcal{L}$-constraint database over $\widetilde{R}$. The *value* of f on input database $\widetilde{r}$, denoted by $f(\widetilde{r})$, is the following $\mathbf{M}_{\mathcal{L}}$-relation:

$$\{(a_1, \ldots, a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \ (\mathbf{M}_{\mathcal{L}}, Dom, points(\widetilde{r})) \models \phi(a_1, \ldots, a_m)\}.$$

The above definition is somewhat problematic. The result of a query on an $\mathcal{L}$-constraint database is defined to be an unrestricted $\mathbf{M}_{\mathcal{L}}$-relation. Intuitively we would like to guarantee *closure* as pointed out in [KKR90]. In other words, given an expression f of relational calculus with $\mathcal{L}$-constraints, and an $\mathcal{L}$-constraint database $\widetilde{r}$, is it possible to find an $\mathcal{L}$-constraint relation which finitely represents $f(\widetilde{r})$? In section 4.9, we show that this closure property can be guaranteed for *safe* expressions in relational calculus with $\mathcal{L}$-constraints.

Expressions of relational calculus with $\mathcal{L}$-constraints can also be used for querying indefinite $\mathcal{L}$-constraint relations. To achieve this we have to resort to the set of unrestricted $\mathbf{M}_{\mathcal{L}}$-relations represented by the given indefinite $\mathcal{L}$-constraint relations. Similar definitions have appeared in [IL84, Gra89] in the special case of conditional tables.

Definition 4.3.4 Let $\widetilde{R}$ be a database scheme and f be the following expression over $\widetilde{R}$ in relational calculus with $\mathcal{L}$-constraints:

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \phi(x_1, \ldots, x_m)\}$$

Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database over $\widetilde{R}$. The *value* of f on input database $\widetilde{r}$, denoted by $f(\widetilde{r})$, is the set of $\mathbf{M}_{\mathcal{L}}$-relations

$$\{f(\widetilde{r'}) : \widetilde{r'} \in rep(\widetilde{r})\}.$$

As can be seen from the above definition, the value of a relational calculus expression on an indefinite $\mathcal{L}$-constraint database can be an infinite set whose elements are infinite sets of tuples! In section 4.9, we will show that safe expressions in relational calculus with $\mathcal{L}$-constraints can be evaluated in closed form. In other words, given a safe expression f of relational calculus with $\mathcal{L}$-constraints and an indefinite $\mathcal{L}$-constraint database $\widetilde{r}$, it is possible to find an indefinite $\mathcal{L}$-constraint relation which finitely represents $f(\widetilde{r})$.

4.3.2 Modal Relational Calculus With $\mathcal{L}$-Constraints

As we explained in section 4.1 modal queries are important in any indefinite information context. In this section, we extend the relational calculus with $\mathcal{L}$-constraints by introducing the modal operators of *possibility* and *certainty* [HC68]. Note that the use of these operators is restricted. Similar query languages have been investigated in [Lip79, Lev84, Rei88].

Definition 4.3.5 Let $\widetilde{R}$ be a database scheme and $R(C_1, \ldots, C_m)$ be a relation scheme. An expression over $\widetilde{R}$ in *modal relational calculus with $\mathcal{L}$-constraints* is

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \Diamond \ \phi(x_1, \ldots, x_m)\} \ \text{ or}$$

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \Box \ \phi(x_1, \ldots, x_m)\}$$

where $s_i \in sorts(\mathcal{L}^+(\widetilde{R}))$ is the sort of C_i, ϕ is a well-formed formula of $\mathcal{L}^+(\widetilde{R})$, $x_1, \ldots, x_m$ are the only free variables of ϕ, $\Box$ is the modal operator for *certainty* and $\Diamond$ is the modal operator for *possibility*.

Expressions with the operator $\Box$ (resp. $\Diamond$) will be called $\Box$-expressions (resp. $\Diamond$-expressions).

Expressions of modal relational calculus with $\mathcal{L}$-constraints define functions from indefinite $\mathcal{L}$-constraint databases to *singleton sets* of $\mathbf{M}_\mathcal{L}$-relations as follows.

Definition 4.3.6 Let $\widetilde{R}$ be a database scheme and f be the following expression over $\widetilde{R}$ in modal relational calculus with $\mathcal{L}$-constraints:

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \Box \ \phi(x_1, \ldots, x_m)\}$$

Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database over $\widetilde{R}$. The value of f on input database $\widetilde{r}$, denoted by $f(\widetilde{r})$, is the following set which contains a *single* $\mathbf{M}_\mathcal{L}$-relation.

$$\{ \ \{(a_1, \ldots, a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \text{ for every } \mathbf{M}_\mathcal{L} - \text{relational database } \widetilde{r}' \in sem(\widetilde{r}), \\ (\mathbf{M}_\mathcal{L}, Dom, \widetilde{r}') \models \phi(a_1, \ldots, a_m)\} \ \}$$

Definition 4.3.7 Let $\widetilde{R}$ be a database scheme and f be the following expression over $\widetilde{R}$ in modal relational calculus with $\mathcal{L}$-constraints:

$$\{R(C_1, \ldots, C_m), x_1/s_1, \ldots, x_m/s_m \ : \ \Diamond \ \phi(x_1, \ldots, x_m)\}$$

Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database over the same scheme. The value of f on input database $\widetilde{r}$, denoted by $f(\widetilde{r})$, is the following set which contains a *single* $\mathbf{M}_\mathcal{L}$-relation:

$$\{ \ \{(a_1, \ldots, a_m) \in dom(s_1) \times \cdots \times dom(s_m) : \text{ there exists an } \mathbf{M}_\mathcal{L} - \text{relational database } \widetilde{r}' \in sem(\widetilde{r}), \\ \text{such that } (\mathbf{M}_\mathcal{L}, Dom, \widetilde{r}') \models \phi(a_1, \ldots, a_m)\} \ \}$$

Section 4.9 demonstrates that safe expressions in modal relational calculus with $\mathcal{L}$-constraints can be evaluated in closed form. For every safe expression f in modal relational calculus with $\mathcal{L}$-constraints and indefinite $\mathcal{L}$-constraint database $\widetilde{r}$, it is possible to find an indefinite $\mathcal{L}$-constraint relation without e-variables which finitely represents $f(\widetilde{r})$.

Definition 4.3.8 Let f be an expression of modal relational calculus with $\mathcal{L}$-constraints. Let f' be the expression of relational calculus with $\mathcal{L}$-constraints which is obtained from f by discarding the modal operator. Then f and f' will be called *corresponding*.

The next lemma demonstrates an intuitive property of modal relational calculus with $\mathcal{L}$-constraints. If $\mathcal{S}$ is a set of sets then $\bigcap \mathcal{S}$ (resp. $\bigcup \mathcal{S}$) denotes the set $\{\cap_{s \in \mathcal{S}} s\}$ (resp. $\{\cup_{s \in \mathcal{S}} s\}$).

Lemma 4.3.1 *Let f be a $\Box$-expression (resp. $\Diamond$-expression) over $\widetilde{R}$ in modal relational calculus with $\mathcal{L}$-constraints. Let f' be the expression of relational calculus with $\mathcal{L}$-constraints which corresponds to f. Then for all indefinite $\mathcal{L}$-constraint databases $\widetilde{r}$ over $\widetilde{R}$, $f(\widetilde{r}) = \bigcap f'(\widetilde{r})$ (resp. $f(\widetilde{r}) = \bigcup f'(\widetilde{r})$).*

Proof: Let ϕ be the $\mathcal{L}^+(\widetilde{R})$ formula of f and f'. The following equivalences prove the lemma for $\Box$-expressions. The proof is similar for $\Diamond$-expressions.

$(a_1, \ldots, a_m) \in f(\widetilde{r})$ iff
for every $\mathbf{M}_\mathcal{L}$-relational database $\widetilde{r_1} \in sem(\widetilde{r})$, $(\mathbf{M}_\mathcal{L}, Dom, \widetilde{r_1}) \models \phi(a_1, \ldots, a_m)$ iff
for every $\mathcal{L}$-constraint database $\widetilde{r_2} \in rep(\widetilde{r})$,
$(\mathbf{M}_\mathcal{L}, Dom, points(\widetilde{r_2})) \models \phi(a_1, \ldots, a_m)$ iff
for every $\mathcal{L}$-constraint database $\widetilde{r_2} \in rep(\widetilde{r})$, $(a_1, \ldots, a_m) \in f'(\widetilde{r_2})$ iff
$(a_1, \ldots, a_m) \in \bigcap f'(\widetilde{r})$.

■

Let us now turn to the development of a procedural query language for each of our database schemes.

4.4 The $\mathbf{M}_\mathcal{L}$-Relational and the $\mathcal{L}$-Constraint Algebra

In this section, we develop two procedural query languages: the $\mathbf{M}_\mathcal{L}$-relational algebra and the $\mathcal{L}$-constraint algebra. The $\mathbf{M}_\mathcal{L}$-relational algebra is interesting only from a theoretical point of view because it operates on unrestricted $\mathbf{M}_\mathcal{L}$-relations. Its operations can be defined as in the case of finite relations (verbatim!) [Kan90].

The operations of the $\mathcal{L}$-constraint algebra are extensions of similar operations of standard relational algebra [Mai83]. We will give definitions for selection, projection, join, difference, union and rename. The definitions of cartesian product and intersection are omitted. Similar operations have been defined by [KSW90] for the special case of generalized relations (see section 2.7).

Selection. Let us first define *selection conditions* as follows:

1. An $\mathcal{L}$-constraint whose variables are elements of $\mathcal{U}_V$ (the set of variable attributes) is an *atomic $\mathcal{L}$-selection condition.*

2. A formula of the form $A = a, A \neq a, A = B$ or $A \neq B$, where A and B are data attributes, is an *atomic data selection condition.*

3. If E_1 and E_2 are selection conditions then $E_1 \wedge E_2$ is a selection condition.

A selection condition E will usually be written as $E_d \wedge E_v$ where E_v is a conjunction of $\mathcal{L}$-selection conditions and E_d is a conjunction of data selection conditions.

Let r be an $\mathcal{L}$-constraint relation over scheme R and $E = E_d \wedge E_v$ be a selection condition. Then the selection $\sigma_E(r)$ is defined as follows:

1. $sch(\sigma_E(r)) = sch(r)$
2. $\sigma_E(r) = \{\sigma_E(t) : \; t \in r \text{ and } t(R_d) \text{ satisfies } E_d\}$

where $\sigma_E(t)(r) = t(r)$ and $\sigma_E(t)(CON) = t(CON) \wedge E(t)$. The constraint $E(t)$ is obtained from E by substituting $t(A)$ for each variable attribute A that appears in E. For the data part of t satisfaction is defined as for the standard relational case.

Projection. If r is an $\mathcal{L}$-constraint relation on scheme R and $X \subseteq R$ then $\pi_X(r)$ is defined as follows:

1. $sch(\pi_X(r)) = X$
2. $\pi_X(r) = \bigcup_{t \in r} \pi_X(t)$.

The $\mathcal{L}$-constraint *relation* $\pi_X(t)$ is defined as follows. Let V be the set of u-variables u such that there exists $A \in R \setminus X$ and $t(A) = u$. Then $\pi_X(t)$ is the set of tuples t_1 over scheme X such that $t_1(X) = t(X)$ and $t_1(CON)$ is a disjunct in the disjunction of conjunctions of $\mathcal{L}$-constraints obtained by eliminating variables V from $t(CON)$. The latter formula is well-defined since the class of $\mathcal{L}$-constraints admits variable elimination (see definition 3.5.1).

Join. Let r_1 and r_2 be $\mathcal{L}$-constraint relations over schemes R_1 and R_2. Let X_v be the set of variable attributes common to R_1 and R_2 and X_d be the set of regular attributes common to R_1 and R_2. The join $r_1 \bowtie r_2$ can be defined as follows:

1. $sch(r_1 \bowtie r_2) = sch(r_1) \cup (sch(r_2) \setminus (X_v \cup X_d))$.
2. $r_1 \bowtie r_2 = \{t_1 \bowtie t_2 : \; t_1 \in r_1, \; t_2 \in r_2 \text{ and } t_1(X_d) = t_2(X_d)\}$.

The join of two tuples $t_1 \bowtie t_2$ is defined by the following:

1. $(t_1 \bowtie t_2)(A) = t_1(A)$ if $A \in R_1$
2. $(t_1 \bowtie t_2)(A) = t_2(A)$ if $A \in R_2 \setminus (X_v \cup X_d)$
3. $(t_1 \bowtie t_2)(CON) = t_1(CON) \wedge t_2'(CON)$ where $t_2'(CON)$ is the same as $t_2(CON)$ except that each variable $t_2(A)$ has been substituted by $t_1(A)$ for each $A \in X_v$.

Difference. Let r_1 and r_2 be $\mathcal{L}$-constraint relations over scheme $R = R_v \cup R_d$. The difference $r_1 - r_2$ can now be defined as follows:

1. $sch(r_1 - r_2) = sch(r_1) = sch(r_2)$.
2. $r_1 - r_2 = \bigcup_{t \in r_1} (t_{r_2})^{\dagger}$ where $r_2 = \{t_1, \cdots, t_n\}$ and $t_{r_2} = (((t - t_1) - t_2) - \cdots - t_n)$.

The difference of two $\mathcal{L}$-constraint tuples $t_1 - t_2$ is an $\mathcal{L}$-*constraint relation* which is defined as follows:

1. $t_1 - t_2 = \{t_1\}$ if $t_2(R_d) \neq t_1(R_d)$.
2. $t_1 - t_2 = \{s : \; s(R_d) = t_1(R_d) \text{ and } s(R_v \cup \{CON\}) = \overline{t_2(R_v \cup \{CON\})} \cap t_1(R_v \cup \{CON\}) \; \}$ otherwise.

To understand the above formulas, we need the following clarifications. The tuple $\bar{t}$ is the complement of t. It is computed by negating the local condition of t and leaving the rest of it unchanged. We will abuse our terminology and call $\bar{t}$ a tuple although its local condition is not of the form required by the definition of section 4.2.

The intersection of two $\mathcal{L}$-constraint tuples $t_1 \cap t_2$ is defined as follows:

$$(t_1 \cap t_2)(R_v) = t_1(R_v) \text{ and } (t_1 \cap t_2)(CON) = t_1(CON) \wedge t_2'(CON)$$

where $t_2'(CON)$ is the same as $t_2(CON)$ but with each variable $t_2(A)$ substituted by $t_1(A)$ for each $A \in R_v$.

The operation $r^{\uparrow}$ has the effect of *normalizing* the local conditions of a relation r in order to obtain a true $\mathcal{L}$-constraint relation. This is done by the following three steps:

- Application of De Morgan's laws to transform the negated part of each local condition of r (which was produced by the complementation operation) into a disjunction whose disjuncts are $\mathcal{L}$-constraints. This operation is well-defined since the class of $\mathcal{L}$-constraints is weakly closed under negation.
- Application of the law of associativity of conjunction with respect to disjunction to transform each local condition of r into a disjunction of conjunctions of $\mathcal{L}$-constraints.
- Splitting of disjuncts into different tuples with the same variable part.

Union. Let r_1 and r_2 be $\mathcal{L}$-constraint relations over scheme R. The union $r_1 \cup r_2$ is defined as follows:

1. $sch(r_1 \cup r_2) = sch(r_1) = sch(r_2)$.
2. $r_1 \cup r_2 = \{t : t \in r_1 \text{ or } t \in r_2\}$.

Rename: Let r be an $\mathcal{L}$-constraint relation over scheme R. Let $A \in R$, $C \in \mathcal{U} \setminus R$ and $sort(A) = sort(C)$. The operation $\theta_A^C(r)$ of renaming attribute C to A is defined as follows:

- $sch(\theta_A^C(r)) = (R \setminus \{A\}) \cup \{C\}$
- $\theta_A^C(r) = \{t : t \in r\}$

Cartesian Product, Intersection. The definitions of these two operations are omitted for brevity.

Constant $\mathcal{L}$-constraint relations are needed sometimes in algebra expressions. We will follow [Mai83] and denote constant $\mathcal{L}$-constraint relations as sets of tuples with explicit attribute naming.

Example 4.4.1 If $A \in U_{\mathcal{D}}, B \in U_{\mathcal{Q}}, a_1, a_2 \in Dom$ and $t_1, t_2 \in UVAR_{dePCL}^{\mathcal{Q}}$ then

$$\{(A : a_1, B : t_1, CON : t_1 \leq 5),\ (A : a_2, B : t_2, CON : true)\}$$

is a constant dePCL-constraint relation.

Definition 4.4.1 An $\mathcal{L}$-*constraint algebra expression* over database scheme $\widetilde{R}$ is any well-formed expression built-up from constant $\mathcal{L}$-constraint relations, scheme names from $\widetilde{R}$ and the above operators.

$\mathcal{L}$-constraint algebra expressions define functions from $\mathcal{L}$-constraint databases to $\mathcal{L}$-constraint relations as follows. If e is an $\mathcal{L}$-constraint algebra expression over $\widetilde{R}$ and $\widetilde{r}$ is an $\mathcal{L}$-constraint database over $\widetilde{R}$, then $e(\widetilde{r})$ (the value of e on input database $\widetilde{r}$) is the $\mathcal{L}$-constraint relation r' obtained by substituting relations from $\widetilde{r}$ for relation schemes in e and evaluating the expression in a *bottom-up* fashion using the above definitions. The scheme $sch(r') = sch(e(\widetilde{r}))$ is also determined by the previous definitions.

Definition 4.4.2 Let e_1, e_2 be $\mathcal{L}$-constraint algebra expressions over $\widetilde{R}$ such that $sch(e_1) = sch(e_2)$. The expressions e_1 and e_2 are called *equivalent* iff for all $\mathcal{L}$-constraint databases $\widetilde{r}$ over $\widetilde{R}$, $points(e_1(\widetilde{r})) = points(e_2(\widetilde{r}))$.

The following proposition presents a subset of $\mathcal{L}$-constraint algebra operations which has the same expressive power as the full algebra.

Proposition 4.4.1 *Let e_1 be an expression of the $\mathcal{L}$-constraint algebra over $\widetilde{R}$. Then there exists an equivalent $\mathcal{L}$-constraint algebra expression e_2 over the same scheme which uses only constant $\mathcal{L}$-constraint relations, selection, projection, union, join, difference and rename. More precisely, (i) constant $\mathcal{L}$-constraint relations can have a single tuple with a single attribute and local condition "true", and (ii) selections can involve a single atomic condition.*

Proof: The proof of this theorem is an easy generalization of theorem 3.1 of [Mai83] which covers the standard relational case. ∎

4.5 The Modal $\mathcal{L}$-Constraint Algebra

Let us now turn our attention to the modal $\mathcal{L}$-constraint algebra. The operations of this algebra take as input one (or two) indefinite $\mathcal{L}$-constraint relations associated with a common global condition and return an indefinite $\mathcal{L}$-constraint relation associated with the *same* global condition.

The modal $\mathcal{L}$-constraint algebra contains an operation for every $\mathcal{L}$-constraint algebra operation defined in the previous section. The definitions of these operations were originally given in [Kou93a] for the special case of indefinite dePCL-constraint relations.[4] These operations treat e-variables as uninterpreted parameters thus they are defined exactly as the $\mathcal{L}$-constraint algebra operations. The modal algebra also includes two additional operations $POSS$ and $CERT$, which take a more active stand towards e-variables.

Given an indefinite $\mathcal{L}$-constraint relation r, the expression $POSS(r)$ evaluates to an $\mathcal{L}$-constraint relation which finitely represents the set of all tuples contained in *any* relation of $sem(r)$. The expression $CERT(r)$ evaluates to an $\mathcal{L}$-constraint relation which finitely represents the set of all tuples contained in *every* relation of $sem(r)$.

Possibility. Let r be an indefinite $\mathcal{L}$-constraint relation on scheme R. Then $POSS(r)$ is an *$\mathcal{L}$-constraint relation* defined as follows:

1. $sch(POSS(r)) = sch(r)$
2. $POSS(r) = \{poss(t) : t \in r\}$.

For each tuple t on scheme R, $poss(t)$ is a tuple on scheme R such that $poss(t)(R) = t(R)$ and $poss(t)(CON) = \psi$ where ψ is obtained by eliminating all e-variables from the boolean combination of $\mathcal{L}$-constraints $G(r) \wedge t(CON)$. The expression $poss(t)(CON)$ is well-defined since the class of $\mathcal{L}$-constraints admits variable elimination.

[4] [Kou93a] uses the term *temporal tables* for indefinite dePCL-constraint relations.

Certainty. Let r be an indefinite $\mathcal{L}$-constraint relation on scheme R. Then $CERT(r)$ is an $\mathcal{L}$-constraint relation defined as follows:

1. $sch(CERT(r)) = sch(r)$
2. $CERT(r) = \{cert(t) : t \in r^{\downarrow}\}^{\uparrow}$.

For each tuple t on scheme R, $cert(t)$ is a tuple on scheme R such that $cert(t)(R) = t(R)$ and $cert(t)(CON) = \neg\psi$ where ψ is obtained by eliminating all e-variables from the boolean combination of $\mathcal{L}$-constraints $G(r) \wedge \neg t(CON)$. The expression $cert(t)(CON)$ is well-defined since the class of $\mathcal{L}$-constraints admits variable elimination.

The operation $r^{\downarrow}$ has the effect of *denormalizing* $\mathcal{L}$-constraint relation r. This is achieved by collecting tuples $\{t_1, \ldots, t_k\}$ of r with the same data part $(a_1, \ldots, a_{|R_d|})$ into a single tuple t' on scheme R such that $t'(R_d) = (a_1, \ldots, a_{|R_d|})$, $t'(R_v) = (x_1, \ldots, x_{|R_v|})$ and $t'(CON) = t'_1(CON) \vee \cdots \vee t'_k(CON)$. In the new tuple t' u-variables have been standardized apart: $x_1, \ldots, x_{|R_v|}$ are brand new u-variables, and for $1 \leq i \leq k$, $t'_i(CON)$ is the same as $t_i(CON)$ except that $t(X)$ has been substituted by $t'(X)$ for each $X \in R_v$.

Let us now define modal $\mathcal{L}$-constraint algebra expressions.

Definition 4.5.1 A *pure expression* over scheme $\widetilde{R}$ in modal $\mathcal{L}$-constraint algebra is any well-formed expression built from constant $\mathcal{L}$-constraint relations, scheme names from $\widetilde{R}$ and the above operators excluding $POSS$ and $CERT$. An *modal $\mathcal{L}$-constraint algebra expression* is a pure expression, or an expression of the form $CERT(g)$ or $POSS(g)$ where g is a pure expression. Expressions of the form $CERT(g)$ or $POSS(g)$ are called *$CERT$-expressions* or *$POSS$-expressions* respectively.

Modal $\mathcal{L}$-constraint algebra expressions define functions from indefinite $\mathcal{L}$-constraint databases to indefinite $\mathcal{L}$-constraint relations. The result of applying an expression e to an indefinite $\mathcal{L}$-constraint database $\widetilde{r}$ is defined as for the $\mathcal{L}$-constraint algebra. Let us simply stress that $G(e(\widetilde{r})) = G(\widetilde{r})$ for all indefinite $\mathcal{L}$-constraint databases $\widetilde{r}$ and expressions e over $\widetilde{R}$.

The following lemma gives an intuitive property of $POSS$ and $CERT$.

Lemma 4.5.1 *Let e be a pure expression over scheme $\widetilde{R}$ in modal $\mathcal{L}$-constraint algebra. Then for all indefinite $\mathcal{L}$-constraint databases $\widetilde{r}$ over $\widetilde{R}$*

$$sem(CERT(e(\widetilde{r}))) = \bigcap sem(e(\widetilde{r})) \text{ and } sem(POSS(e(\widetilde{r}))) = \bigcup sem(e(\widetilde{r})).$$

Proof: Let $r' = e(\widetilde{r})$ be the result of applying e to $\widetilde{r}$, and let $sch(r') = R'$. We will show that $sem(CERT(r')) = \bigcap sem(r')$.

Let δ be a sequence of elements of Dom, τ a sequence of elements of $\bigcup_{s \in sorts(\mathcal{L})} dom(s, \mathbf{M}_{\mathcal{L}})$ and χ a sequence of elements of $UVAR_{\mathcal{L}}$. An $\mathbf{M}_{\mathcal{L}}$-relational tuple t over $\widetilde{R}$ such that $t(R_d) = \delta$ and $t(R_v) = \tau$ will be denoted by (δ, τ). Similarly, an indefinite $\mathcal{L}$-constraint tuple t over $\widetilde{R}$ such that $t(R_d) = \delta$ and $t(R_v) = \chi$ will be denoted by (δ, χ).

Let us assume $sem(CERT(r')) = \{r_1\}$ and $\bigcap sem(r') = \{r_2\}$ where r_1 and r_2 are relations over R'. Initially, we will show that $r_1 \subseteq r_2$.

Let us assume $(\delta, \tau) \in r_1$. Then there exists a tuple $t = (\delta, \chi) \in CERT(r')^{\downarrow}$ such that $\mathbf{M}_{\mathcal{L}} \models t(CON)[\chi \leftarrow \tau]$. Therefore $\mathbf{M}_{\mathcal{L}} \not\models \neg t(CON)[\chi \leftarrow \tau]$. Notice that $\neg t(CON)$ is obtained by eliminating all e-variables from $G(r') \wedge \neg t'(CON)$ where t' is the tuple of $r'^{\downarrow}$ such that $t'(R_d) = \delta$.

Now assume that $(\delta, \tau) \notin r_2$. Therefore there exists a valuation $v \in Val^e_{\mathbf{M}_\mathcal{L}}$ such that $\mathbf{M}_\mathcal{L} \models G(r')[v]$ and $\mathbf{M}_\mathcal{L} \not\models t'(CON)[v; \chi \leftarrow \tau]$. In this case, $\mathbf{M}_\mathcal{L} \models G(r')[v]$ and $\mathbf{M}_\mathcal{L} \models \neg t'(CON)[v; \chi \leftarrow \tau]$ thus $\mathbf{M}_\mathcal{L} \models \neg t(CON)[\chi \leftarrow \tau]$. Contradiction! Then it must be $(\delta, \tau) \in r_2$.

Let us now show $r_2 \subseteq r_1$. Assume $(\delta, \tau) \in r_2$. Then there exists a tuple $t = (\delta, \chi) \in r'^{\downarrow}$ such that, for all valuations $v \in Val^e_{\mathbf{M}_\mathcal{L}}$, if $\mathbf{M}_\mathcal{L} \models G(r')[v]$ then $\mathbf{M}_\mathcal{L} \models t(CON)[v; \chi \leftarrow \tau]$.

Now assume $(\delta, \tau) \notin r_1$. If t' is the tuple of $CERT(r)^{\downarrow}$ such that $t'(R_d) = \delta$ and $t'(R_v) = \chi$ then $\mathbf{M}_\mathcal{L} \not\models t'(CON)[\chi \leftarrow \tau]$. Therefore $\mathbf{M}_\mathcal{L} \models \neg t'(CON)[\chi \leftarrow \tau]$. The formula $\neg t'(CON)$ is obtained after all e-variables are eliminated from $G(r) \wedge \neg t(CON)$. Therefore valuation $\chi \leftarrow \tau$ can be extended to a new valuation $v'; \chi \leftarrow \tau$ such that $v' \in Val^e_{\mathbf{M}_\mathcal{L}}$ and $\mathbf{M}_\mathcal{L} \models G(r) \wedge \neg t(CON)[v'; \chi \leftarrow \tau]$. But this contradicts the conclusion of the previous paragraph. Therefore $(\delta, \tau) \in r_1$.

The proof for $POSS$ is similar. ■

Definition 4.5.2 Let e be a pure expression of modal $\mathcal{L}$-constraint algebra. Let e_1 and e_2 be the $\mathcal{L}$-constraint algebra expression and the $\mathbf{M}_\mathcal{L}$-relational algebra expression which are syntactically the same like e. Then e, e_1 and e_2 will be called *corresponding.*

Definition 4.5.3 Let e_1 and e_2 be modal $\mathcal{L}$-constraint algebra expressions over scheme $\widetilde{R}$ such that $sch(e_1) = sch(e_2)$. Expressions e_1 and e_2 are called *equivalent* iff for all indefinite $\mathcal{L}$-constraint databases $\widetilde{r}$ over $\widetilde{R}$, $sem(e_1(\widetilde{r})) = sem(e_2(\widetilde{r}))$.

The following proposition presents a subset of modal $\mathcal{L}$-constraint algebra operations which has the same expressive power as the full algebra.

Proposition 4.5.1 *Let e_1 be an modal $\mathcal{L}$-constraint algebra expression over $\widetilde{R}$. Then there exists an equivalent modal $\mathcal{L}$-constraint algebra expression e_2 over $\widetilde{R}$ which uses only constant $\mathcal{L}$-constraint relations, selection, projection, union, join, difference, rename and POSS (or CERT). More precisely, (i) constant $\mathcal{L}$-constraint relations have a single tuple with a single attribute and local condition "true", and (ii) selections can involve a single atomic condition.*

Proof: Initially assume that e_1 is pure. Let e'_1 be the $\mathcal{L}$-constraint algebra expression corresponding to e_1. Then there exists an equivalent expression e'_2 which uses only constant $\mathcal{L}$-constraint relations, selection, projection, union, join, difference and rename (proposition 4.4.1). Let e_2 be the pure expression of modal $\mathcal{L}$-constraint algebra which corresponds to e'_2. The equivalence of e_1 and e_2 is demonstrated by the following:

$$sem(e_1(\widetilde{r})) = \{points(e'_1(\widetilde{r_1})) : \widetilde{r_1} \in rep(\widetilde{r})\} = \{points(e'_2(\widetilde{r_1})) : \widetilde{r_1} \in rep(\widetilde{r})\} = sem(e_2(\widetilde{r}))$$

The extension to $CERT$-expressions and $POSS$-expressions is trivial. ■

4.6 On the Semantics of Algebraic Query Evaluation

Let $\widetilde{r}$ be an $\mathcal{L}$-constraint database, e an $\mathcal{L}$-constraint algebra expression and $e1$ its corresponding $\mathbf{M}_\mathcal{L}$-relational algebra expression. Recall that a $\mathcal{L}$-constraint relation $\widetilde{r}$ is a finite representation of the unrestricted set of tuples $points(\widetilde{r})$. The following theorem shows that when we evaluate e on $\widetilde{r}$, we essentially evaluate $e1$ on the unrestricted relation $points(\widetilde{r})$. This is an important result which demostrates that the operations of $\mathcal{L}$-constraint algebra have been defined "correctly".

Theorem 4.6.1 *Let e be an $\mathcal{L}$-constraint algebra expression over $\widetilde{R}$ and $e1$ be its corresponding $\mathbf{M}_\mathcal{L}$-relational algebra expression. If $\widetilde{r}$ is an $\mathcal{L}$-constraint database over scheme $\widetilde{R}$, then $points(e(\widetilde{r})) = e1(points(\widetilde{r}))$.*

Proof: Let r be an $\mathcal{L}$-constraint relation and op a $\mathcal{L}$-constraint algebra operation in the restricted set of operations defined in proposition 4.4.1. We will show that $points(op(r)) = op(points(r))$. Then the theorem follows easily. A detailed proof will be given only for projection and selection. The proofs for the rest of the operations are only sketched. Whenever it is not stated explicitly, relations are assumed to have scheme R.

Projection. We will first prove that $points(\pi_X(r)) \subseteq \pi_X(points(r))$. Let us assume that $t \in points(\pi_X(r))$. Equivalently, there exists a tuple $t' \in \pi_X(r)$ and a valuation $u \in Val^u_{\mathbf{M}_\mathcal{L}}$ such that $t = u(t')$ and $\mathbf{M}_\mathcal{L} \models t'(CON)[u]$. The fact $t' \in \pi_X(r)$ implies that there exists an $\mathcal{L}$-constraint tuple $t'' \in r$ such that $t''(X) = t'$ and $t'(CON)$ is a disjunct in the disjunction of conjunctions of $\mathcal{L}$-constraints which is obtained by eliminating the u-variables in $t''(R \setminus X)$ from $t''(CON)$. Therefore u can be extended to a valuation $u' \in Val^u_{\mathbf{M}_\mathcal{L}}$ such that $\mathbf{M}_\mathcal{L} \models t''(CON))[u']$ and $u'(t''(X)) = u(t'(X)) = t$. Equivalently, there exists an $\mathbf{M}_\mathcal{L}$-relational tuple $s = u'(t'') \in points(r)$ and $s(X) = t$, therefore $t \in \pi_X(points(r))$.

Let us now prove that $\pi_X(points(r)) \subseteq points(\pi_X(r))$. Assume that $t \in \pi_X(points(r))$. Equivalently, there exists an $\mathbf{M}_\mathcal{L}$-relational tuple $t' \in points(r)$ and $t = t'(X)$. The fact that $t' \in points(r)$ implies that there exists an $\mathcal{L}$-constraint tuple $t'' \in r$ and a valuation $u \in Val^u_{\mathbf{M}_\mathcal{L}}$ such that $\mathbf{M}_\mathcal{L} \models t''(CON)[u]$ and $t' = u(t'')$. Then there exists an $\mathcal{L}$-constraint tuple $s \in \pi_X(t'')$ such that $s(X) = t''(X)$, $s(CON)$ is a disjunct in the disjunction of conjunctions of $\mathcal{L}$-constraints obtained by eliminating u-variables $t''(R \setminus X)$ from $t''(CON)$ and $u'(s) = u(t''(X)) = t'(X) = t$. In other words, $t \in points(\pi_X(r))$.

Selection. We will only show that $points(\sigma_E(r)) \subseteq \sigma_E(points(r))$ where E is an atomic selection condition. The inverse inclusion can be shown in a similar way. Let us assume that $t \in points(\sigma_E(r))$. Equivalently, there exists an $\mathcal{L}$-constraint tuple $t' \in \sigma_E(r)$ and a valuation $u \in Val^u_{\mathbf{M}_\mathcal{L}}$ such that $t = u(t'(S))$ and $\mathbf{M}_\mathcal{L} \models t'(CON)[u]$. The fact $t' \in \sigma_E(r)$ implies that there exists a tuple $t'' \in r$ such that $t'(R) = t''(R)$. If E is a data selection condition then $t'(CON)$ is the same as $t''(CON)$. If E is an $\mathcal{L}$-selection condition then $t'(CON)$ is $t''(CON) \wedge E(t'')$ where $E(t'')$ is obtained from E by substituting $t''(A)$ for each attribute A that appears in E. In other words, there exists a $\mathbf{M}_\mathcal{L}$-relational tuple $s = u(t'') \in points(r)$ such that $t = u(t'(R)) = u(t''(R)) = s$ and $E(t)$ evaluates to *true*. Therefore $t \in \sigma_E(points(r))$.

Join. Assume r_1 and r_2 have schemes R_1 and R_2 respectively. We want to prove that $points(r_1 \bowtie r_2) = points(r_1) \bowtie points(r_2)$. Let us assume that $t \in points(r_1 \bowtie r_2)$. Equivalently, there exists an $\mathcal{L}$-constraint tuple $t' \in r_1 \bowtie r_2$ and a valuation $u \in Val^u_{\mathbf{M}_\mathcal{L}}$ and $t = u(t'(R_1 \cup R_2))$ and $\mathbf{M}_\mathcal{L} \models t'(CON)[u]$. In other words $u(t'(R_1)) \in points(r_1)$ and $u(t'(R_2)) \in points(r_2)$. Therefore $t \in points(r_1) \bowtie points(r_2)$.

Difference. We want to prove that $points(r_1 - r_2) = points(r_1) - points(r_2)$. Let us assume that $t \in points(r_1 - r_2)$. Equivalently, there exists $\mathcal{L}$-constraint tuple t' and valuation u such that $t' \in r_1 - r_2$, $u \in Val^u_{\mathbf{M}_\mathcal{L}}$, $t = u(t'(r))$ and $\mathbf{M}_\mathcal{L} \models t'(CON)[u]$. In other words, $u(t') \in points(r_1)$ and $u(t') \notin points(r_2)$. Therefore $t \in points(r_1) - points(r_2)$.

Rename, Union. The proof in this case is trivial. ■

Let us now assume that $\widetilde{r}$ is an indefinite $\mathcal{L}$-constraint database, e is a pure expression of modal $\mathcal{L}$-constraint algebra and $e1$ is its corresponding expression in $\mathcal{L}$-constraint algebra. Recall that the semantic function $sem(\widetilde{r})$ returns all the "possible worlds" represented by $\widetilde{r}$. When we evaluate e on indefinite $\mathcal{L}$-constraint database $\widetilde{r}$ using the operations defined above, we essentially evaluate the corresponding $\mathbf{M}_\mathcal{L}$-relational algebra expression on each possible world in $sem(\widetilde{r})$. As discussed in [Imi89], an extension of a $\mathbf{M}_\mathcal{L}$-relational algebra expression $e1$ to an expression e for an $\mathcal{L}$-constraint representation of indefinite information can claim to be "faithful to the underlying

$$\begin{tikzcd}
\mathcal{L}\text{-}ICDB \arrow[r, "\mathbf{e}"] \arrow[d, "\mathbf{rep}"'] \arrow[dd, bend right=60, "\mathbf{sem}"'] & \mathcal{L}\text{-}ICDB \arrow[d, "\mathbf{rep}"] \arrow[dd, bend left=60, "\mathbf{sem}"] \\
\{\mathcal{L}\text{-}CDB\} \arrow[r, "\mathbf{e1}"] \arrow[d, "\mathbf{points}"'] & \{\mathcal{L}\text{-}CDB\} \arrow[d, "\mathbf{points}"] \\
\{\mathbf{M}_{\mathcal{L}}\text{-}RDB\} \arrow[r, "\mathbf{e2}"] & \{\mathbf{M}_{\mathcal{L}}\text{-}RDB\}
\end{tikzcd}$$

Figure 4.2: Relating the three algebras

semantics" if and only if for every indefinite $\mathcal{L}$-constraint database $\widetilde{r}$,

$$sem(e(\widetilde{r})) = e1(sem(\widetilde{r})) = \{e1(r_1) : \; r_1 \in sem(\widetilde{r})\}.$$

Equivalently, one would like to guarantee that there is always an indefinite $\mathcal{L}$-constraint database $\widetilde{r'}$ such that $sem(\widetilde{r'}) = e1(sem(\widetilde{r}))$. The following theorem demonstrates that $\mathcal{L}$-constraint databases satisfy this form of algebraic completeness.

Theorem 4.6.2 *Let e be a pure expression of modal $\mathcal{L}$-constraint algebra over $\widetilde{R}$, $e1$ its corresponding $\mathcal{L}$-constraint algebra expression and $e2$ its corresponding $\mathbf{M}_{\mathcal{L}}$-relational algebra expression. If $\widetilde{r}$ is an indefinite $\mathcal{L}$-constraint database over scheme $\widetilde{R}$ then $rep(e(\widetilde{r}))$ is equivalent to $e1(rep(\widetilde{r}))$ and $sem(e(\widetilde{r})) = e2(sem(\widetilde{r}))$.*

Proof: We will only prove the second part of the theorem. The proof of the first part is similar. From the definition of sem, $sem(e(\widetilde{r})) = \{points(\widetilde{r_1}) : \; \widetilde{r_1} \in rep(e(\widetilde{r}))\}$. In addition

$$\begin{aligned} rep(e(\widetilde{r})) = \{\widetilde{r_2} : \text{ there exists } v \in Val^e_{\mathbf{M}_{\mathcal{L}}} \text{ s.t. } \mathbf{M}_{\mathcal{L}} \models G(e(\widetilde{r}))[v] \text{ and } \widetilde{r_2} = v(e(\widetilde{r}))\} = \\ \{v(e(\widetilde{r})) : \text{ there exists } v \in Val^e_{\mathbf{M}_{\mathcal{L}}} \text{ s.t. } \mathbf{M}_{\mathcal{L}} \models G(\widetilde{r})[v]\}. \end{aligned}$$

Therefore $sem(e(\widetilde{r})) = \{points(v(e(\widetilde{r}))) : \text{ there exists } v \in Val^e_{\mathbf{M}_{\mathcal{L}}} \text{ s.t. } \mathbf{M}_{\mathcal{L}} \models G(\widetilde{r})[v]\}$.

When a pure expression of $\mathcal{L}$-constraint algebra is evaluated, using the definitions in section 4.4, the e-variables are treated as uninterpreted parameters (i.e., exactly like constants). Thus $v(e(\widetilde{r})) = e1(v(\widetilde{r}))$ for all indefinite $\mathcal{L}$-constraint databases $\widetilde{r}$ over $\widetilde{R}$. Finally,

$$\begin{aligned} sem(e(\widetilde{r})) &= \{points(e1(v(\widetilde{r}))) : \text{ there exists } v \in Val^e_{\mathbf{M}_{\mathcal{L}}} \text{ s.t. } \mathbf{M}_{\mathcal{L}} \models G(\widetilde{r})[v]\} \\ &= \{points(e1(\widetilde{r'})) : \text{ there exists } v \in Val^e_{\mathbf{M}_{\mathcal{L}}} \text{ s.t. } \mathbf{M}_{\mathcal{L}} \models G(\widetilde{r})[v] \text{ and } \widetilde{r'} = v(\widetilde{r})\} \\ &= \{points(e1(\widetilde{r'})) : \; \widetilde{r'} \in rep(\widetilde{r})\} = \{e2(points(\widetilde{r'})) : \; \widetilde{r'} \in rep(\widetilde{r})\} \\ &= e2(sem(\widetilde{r})) \end{aligned}$$

■

The above theorems are summarized graphically in the commutative diagram of figure 4.2. Since these results have been proved in our general framework, special cases of constraint databases [KKR90, KSW90, Kou93a] can simply refer to these theorems to demonstrate the "correctness" of the operations of their algebraic query languages.

4.7 On Canonical Forms For Constraint Databases

Constraint databases have a rather annoying difference from relational databases: two databases can be semantically equivalent although they are not syntactically identical. Therefore it is important to devise a *canonical form* which helps to alleviate this problem. More generally a canonical form can be useful in several ways. It can be used for simplifying and standardizing the internal constraint representation used by the system or the output representation shown to the user. This usually involves eliminating redundant constraints and transforming the remaining ones into a certain special form. It can also be used for answering queries more efficiently and for determining the equivalence of two sets of constraints, two constraint relations or two constraint databases.

Computing a canonical form might not be an easy task. For example, for the case of linear inequality constraints [LM89] defines a canonical form which can be computed in PTIME. On the other hand [Sri92] shows that eliminating redundancy in constraint databases with very simple linear inequality constraints is NP-complete.

In general many canonical forms are possible for a given class of constraint databases. Our choice will most likely be affected by implementation considerations thus it falls out of the scope of this dissertation. We will make an exception in section 8.3 where we study a canonical form for dePCL-constraints. In the examples of the following section we will not utilize any particular canonical form for presenting constraint databases. We will only make sure that redundant constraints are eliminated.

4.8 Examples

Let us first consider the deTCL-constraint database of section 4.1. The following are examples of queries.

Example 4.8.1 Find the vehicles that are on service throughout the first hour of the afternoon.

This query can be answered by evaluating the expression

$$\pi_{VEH}(\sigma_{TIME_L \leq 12 \,\wedge\, 13 \leq TIME_R}(ON\_SERVICE))$$

The following indefinite deTCL-constraint relation is the result of the inside selection operation.

VEH	STATION	TIME	CON
V1	Athens	$i1$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R,\ i1_L \leq 12,\ 13 \leq i1_R$
V2	Piraeus	$i2$	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R,\ i2_L \leq 12,\ 13 \leq i2_R$

To execute the projection operation, we need to eliminate the variables $i1$ and $i2$ from the local conditions. After translating the deTCL-constraints of the first tuple into dePCL-constraints as we described in section 3.2.3, we are left with the following:

$$B_{i1} < E_{i1},\ B_{\omega 1} < E_{\omega 1},\ B_{\omega 1} \leq B_{i1},\ B_{i1} \leq E_{\omega 1},\ B_{i1} \leq 12,\ 13 \leq E_{i1}.$$

Eliminating B_{i1} and E_{i1} gives: $B_{\omega 1} \leq E_{\omega 1},\ 13 \leq E_{\omega 1},\ B_{\omega 1} \leq 12,\ B_{\omega 1} < E_{\omega 1}$. Thus when the translation back to deTCL-constraints is performed, we get $\omega 1_L \leq 12,\ 13 \leq \omega 1_R$. Similarly, $\omega 2_L \leq 12,\ 13 \leq \omega 2_R$ is obtained from the second tuple. Therefore, we arrive at the following indefinite deTCL-constraint relation:

VEH	CON
V1	$\omega 1_L \leq 12,\ 13 \leq \omega 1_R$
V2	$\omega 2_L \leq 12,\ 13 \leq \omega 2_R$

Now observe that the condition of the first tuple and the first conjunct of condition of the second tuple are logically implied by the global condition G. Thus we can simplify the local conditions and arrive at the relation ANS1 of section 4.1.

Example 4.8.2 Find all times during which it is certain that both vehicles V1 and V2 are on service.

This query can be answered by evaluating the expression

$$CERT(\pi_{TIME}(\sigma_{VEH=V1}(ON\_SERVICE)) \bowtie \pi_{TIME}(\sigma_{VEH=V2}(ON\_SERVICE))).$$

The evaluation of the argument of $CERT$ results in the following indefinite deTCL-constraint relation:

TIME	CON
i	$\omega 1_L \leq i_L,\ i_R \leq \omega 1_R,\ \omega 2_L \leq i_L,\ i_R \leq \omega 2_R$

To proceed with the evaluation of $CERT$, we have to eliminate all e-variables from

$$G' \wedge (i1_L < i1_R \vee i1_L < \omega 1_L \vee i1_R > \omega 1_R \vee i1_L < \omega 2_L \vee i1_R > \omega 2_R)$$

where G' is the conjunction of G and the constraints in the set

$$\{i_L < i_R :\ i \in V_i^e \text{ and } i \text{ appears in } G\}.$$

Transfoming the above constraint in disjunctive normal form and eliminating e-variables results in $i_L < 9 \vee i_R > 14 \vee i_L < 10.5 \vee i_R > 12$ or equivalently $i_L < 10.5 \vee i_R > 12$. Thus the answer is the following indefinite deTCL-constraint relation without e-variables:

ANS5

TIME	CON
i	$i_L \geq 10.5,\ i_R \leq 12$

Example 4.8.3 Find all times during which it is certain that vehicle V1 is on service in Athens but vehicle V2 is not.

This query can be answered by evaluating the following expression:

$$CERT(\pi_{TIME}(\sigma_{VEH=V1 \wedge CITY=Athens}(ON\_SERVICE)) \\ -\pi_{TIME}(\sigma_{VEH=V2 \wedge CITY=Athens}(ON\_SERVICE)))$$

Let r_1 and r_2 denote the first and the second operand of the above difference operator. Let us compute r_1 and r_2 first:

r_1

TIME	CON
$i1$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R$

r_2

TIME	CON
$i2$	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R$

Since r_1 and r_2 have only one tuple each, we simply compute the difference of these two tuples s_1 and s_2. Observe that $\overline{s_2}(TIME) = t_2$, $\overline{s_2}(CON) = \neg(\omega 2_L \leq i2_L \wedge i2_R \leq \omega 2_R)$ therefore $(s_1 - s_2)(TIME) = i_1$ and $(s_1 - s_2)(CON) = \omega 1_L \leq i1_L \wedge i1_R \leq \omega 1_R \wedge (\omega 2_L > i2_L \vee i2_R > \omega 2_R)$.

After normalizing we get the following indefinite deTCL-constraint relation $r_1 - r_2$:

$r_1 - r_2$

TIME	CON
$i1$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R,\ \omega 2_L > i2_L$
$i3$	$\omega 1_L \leq i3_L,\ i3_R \leq \omega 1_R,\ i3_R > \omega 2_R$

Now we have to compute $CERT(r_1 - r_2)$. First we compute the indefinite deTCL-constraint relation $(r_1 - r_2)^{\downarrow}$:

$r_1 - r_2)^{\downarrow}$

TIME	CON
i	$(\omega 1_L \leq i_L\ \wedge\ i_R \leq \omega 1_R\ \wedge\ \omega 2_L > i_L) \vee$ $(\omega 1_L \leq i_L\ \wedge\ i_R \leq \omega 1_R\ \wedge\ i_R > \omega 2_R)$

Now we have to eliminate all e-variables from the following formula

$$G' \wedge \neg(\omega 1_L \leq i_L \wedge i_R \leq \omega 1_R \wedge \omega 2_L > i_L \wedge i_L < i_R) \wedge \neg(\omega 1_L \leq i_L \wedge i_R \leq \omega 1_R \wedge i_R > \omega 2_R \wedge i_L < i_R).$$

This elimination results in the formula $i_L < 9\ \vee\ i_R > 10$. Finally, the answer is given by the following indefinite deTCL-constraint relation without e-variables.

ANS6

TIME	CON
i	$i_L \geq 9,\ i_R \leq 10$

Example 4.8.4 Find all pairs of vehicles SV and AV such that it is possible that SV is on service in city CITY throughout interval SVT and AV arrives at CITY at some time point AVT during SVT. Output SV, CITY, SVT, AV and AVT.

The algebraic expression for this query is a bit complex so it will be built in steps. Let f be

$$\theta_{SV}^{VEH}(\theta_{SVT}^{TIME}(ON\_SERVICE)) \bowtie \theta_{CITY}^{TO}(\theta_{AV}^{VEH}(\theta_{ITRIP}^{TIME}(TRIP))),$$

g be $\sigma_{SV \neq AV\ \wedge\ SVT_L \leq ITRIP_R\ \wedge\ ITRIP_R \leq SVT_R}(f)$ and h be $\pi_{SV,CITY,SVT,AV,AVT}(ITP_{ITRIP:DT,AVT}(g))$. Then the algebraic expression for this query is $POSS(h)$.

The indefinite deTCL-constraint relation corresponding to g is

SV	CITY	SVT	AV	FROM	ITRIP	CON
V1	Athens	$i1$	V3	Patra	$i3$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R,\ i3_L = 9,$ $i3_R = \nu,\ i1_L \leq i3_R,\ i3_R \leq i1_R$
V2	Athens	$i2$	V3	Patra	$i4$	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R,\ i4_L = 9,$ $i4_R = \nu,\ i2_L \leq i4_R,\ i4_R \leq i2_R$

The indefinite deTCL-constraint relation corresponding to $ITP_{ITRIP:DT,AVT}(g)$ is

SV	CITY	SVT	AV	FROM	DT	AVT	CON
V1	Athens	$i1$	V3	Patra	$p3$	$p1$	$\omega 1_L \leq i1_L,\ i1_R \leq \omega 1_R,\ p3 = 9,\ p1 = \nu,$ $i1_L \leq p1,\ p1 \leq i1_R,\ p1 < p2$
V2	Athens	$i2$	V3	Patra	$p4$	$p2$	$\omega 2_L \leq i2_L,\ i2_R \leq \omega 2_R,\ p4 = 9,\ p2 = \nu,$ $i2_L \leq p2,\ p2 \leq i2_R,\ p1 < p2$

Finally, $POSS(h)$ is the following indefinite deTCL-constraint relation without e-variables.

ANS4

SV	CITY	SVT	AV	AVT	CON
V1	Athens	$i1$	V3	$p1$	$i1_L = 9,\ i1_R \leq 14,\ i1_L \leq p1 \leq i1_R,\ 12 \leq p1 \leq 13$
V2	Athens	$i2$	V3	$p2$	$i2_L \geq 10,\ i2_R \leq 14.5,\ i2_L \leq p2 \leq i2_R,\ 12 \leq p2 \leq 13$

Example 4.8.5 In the database of section 4.1 the global condition was a conjunction of $\mathcal{L}$-constraints. We will now present an example with a more complex global condition. Let us consider the following story. "David and Murray arrived at the accident scene together. Murray left at least five minutes later than Bill. Bill left at least three minutes later than David. John and Bill did not leave the scene together."

The indefinite dePCL-constraint relation SCENE represents the information in the above story. The e-variables d_s, d_e, m_s, m_e, b_s, d_e and j_s, j_e denote the endpoints of the time intervals David, Murray, Bill and John were at the accident scene.[5]

SCENE

PERSON	FROM	TO	CON
David	d_s	d_e	$true$
Murray	m_s	m_e	$true$
Bill	b_s	b_e	$true$
John	j_s	j_e	$true$

$$G:\ d_s = m_e\ \wedge\ m_e - b_e \geq 5\ \wedge\ b_e - d_e \geq 3\ \wedge\ (b_e > j_e\ \vee\ b_e < j_e).$$

Let us now assume that the query "What are the times that John, David and Murray were possibly at the scene?" is posed. In modal relational calculus with dePCL-constraints this query is expressed by

$$\{TIMES(JS,\ JE,\ DS,\ DE,\ MS,\ ME),\ J_s/\mathcal{Q}, J_e/\mathcal{Q}, D_s/\mathcal{Q}, D_e/\mathcal{Q}, M_s/\mathcal{Q}, M_e/\mathcal{Q}:$$
$$\Diamond(SCENE(John, J_s, J_e) \wedge SCENE(David, D_s, D_e) \wedge SCENE(Murray, M_s, M_e))\}.$$

The equivalent expression in modal dePCL-constraint algebra is $POSS(CP)$ where CP is the cartesian product of expressions

$$\theta^{TO}_{JE}(\theta^{FROM}_{JS}(\pi_{FROM,TO}(\sigma_{NAME=John}(SCENE)))),$$

$$\theta^{TO}_{DE}(\theta^{FROM}_{DS}(\pi_{FROM,TO}(\sigma_{NAME=David}(SCENE))))$$

and

$$\theta^{TO}_{ME}(\theta^{FROM}_{MS}(\pi_{FROM,TO}(\sigma_{NAME=Murray}(SCENE)))).$$

When we evaluate operation POSS, we have to eliminate all e-variables from the dePCL formula

$$J_s = j_s\ \wedge\ J_e = j_e\ \wedge\ D_s = d_s\ \wedge\ D_e = d_e\ \wedge\ M_s = m_s\ \wedge\ M_e = m_e\ \wedge$$
$$d_l = m_e\ \wedge\ m_e - b_e \geq 5\ \wedge\ b_e - d_e \geq 3\ \wedge\ (b_e > j_e\ \vee\ b_e < j_e)$$

where J_s, J_e, D_s, D_e, M_s, M_e are u-variables introduced for tranforming relation SCENE into the form used in the definitions of section 4.2.

The above query will be answered by returning the following dePCL-constraint relation.

[5] In this example we will deviate from our convention of using Greek letters for e-variables.

TIMES

JS	JE	DS	DE	MS	ME	CON
J_s	J_e	D_s	D_e	M_s	M_e	$D_s < D_e,\ M_s < M_e,\ J_s < J_e,$ $D_s = M_s,\ M_e - D_e \geq 8,\ J_e - D_e < 3$
J_s	J_e	D_s	D_e	M_s	M_e	$D_s < D_e,\ M_s < M_e,\ J_s < J_e,$ $D_s = M_s,\ M_e - D_e \geq 8,\ J_e - M_e < -5$

4.9 Translating Safe Calculus Expressions into Algebraic Expressions

This section introduces the class of *safe* expressions in relational calculus and modal relational calculus with $\mathcal{L}$-constraints. Safe calculus expressions have equivalent algebraic expressions. Thus we can answer a safe calculus query by evaluating its equivalent algebraic query. As we have seen in sections 4.4 and 4.5, algebraic query evaluation can be done bottom-up and the answer is obtained in closed form. Therefore safe calculus expressions can also be evaluated bottom-up in closed form on $\mathcal{L}$-constraint databases or indefinite $\mathcal{L}$-constraint databases.[6]

Our definition of safety is syntactic and is based on Ullman's definition of safe domain relational calculus [Ull88]. We conjecture that the results of this section still hold even if we consider a larger class of "safe" calculus formulas e.g., the formulas considered by [vGT91].

Definition 4.9.1 An expression f in (modal) relational calculus with $\mathcal{L}$-constraints is called *safe* if its $\mathcal{L}^+$ formula ϕ satisfies the following:

1. There are no uses of quantifier $\forall$.
2. If $\phi_1 \vee \phi_2$ is a subformula of ϕ then ϕ_1 and ϕ_2 must have the same free variables of sort $\mathcal{D}$.
3. If $\psi = \psi_1 \wedge \ldots \wedge \psi_n$ is a maximal formula of ϕ then all free variables of ψ of sort $\mathcal{D}$ must be limited in the following sense:
   - A variable is limited if it is free in some g_i which is not negated.
   - If ψ_i is $x = a$ or $a = x$ where $a \in Dom$ then x is limited.
   - If ψ_i is $x = y$ or $y = x$ and y is limited then x is limited.
4. The negation operator $\neg$ may only apply to a subformula in a conjunction of the type discussed in the previous rule.

The above notion of safety takes into account *only* the terms of sort $\mathcal{D}$ of a formula. The rest of the terms do *not* present any problems thanks to the local conditions of tuples. For example, notice that the expression $f = \{R(B), t/\mathcal{P} :\ \neg P(t)\}$ in relational calculus with dePCL-constraints is safe. If P is the dePCL-constraint relation $\{(A : t_1, CON : t_1 \leq 5)\}$ then evaluating f on P returns the *finite* dePCL-constraint relation $R = \{(B : t, CON : t > 5)\}$. However, the expression $g = \{R(A, B), x/\mathcal{D}, t/\mathcal{P} :\ \neg Q(x, t)\}$ is not safe. This expression exhibits the usual problems of unsafe formulas with respect to the data domain.

The following theorem demonstrates that safe expressions of relational calculus with $\mathcal{L}$-constraints can be evaluated bottom-up in closed form on $\mathcal{L}$-constraint databases.

[6] This is the most basic requirement of the constraint database framework of [KKR90].

Theorem 4.9.1 *For every safe expression f over $\widetilde{R}$ in relational calculus with $\mathcal{L}$-constraints there exists an $\mathcal{L}$-constraint algebra expression e over $\widetilde{R}$ such that the following property holds. If $\widetilde{r}$ is an $\mathcal{L}$-constraint database over $\widetilde{R}$ then $f(\widetilde{r}) = points(e(\widetilde{r}))$.*

Proof: The proof of this theorem is straightforward for somebody who is familiar with similar proofs from relational database theory [Mai83, Ull88]. We will only give the $\mathcal{L}$-constraint algebra expression for negated database predicates.

Let r be an $\mathcal{L}$-constraint relation over scheme R. The $\mathcal{L}$-constraint algebra expression for $\neg R(a_1, \ldots, a_k)$ is

$$(E^{|R_d|} \bowtie \{(B_1 : x_1, CON : true)\} \bowtie \cdots \bowtie \{(B_{|R_v|} : x_{|R_v|}, CON : true)\}) - e_r$$

where e_r is the $\mathcal{L}$-constraint algebra expression for r, $B_1, \ldots, B_{|R_v|}$ are the attributes in R_v, $x_1, \ldots, x_{|R_v|}$ are u-variables of appropriate sorts, and E is an $\mathcal{L}$-constraint algebra expression for the extended active data domain relative to the given calculus expression.

If f is an expression of relational calculus with $\mathcal{L}$-constraints then the *extended active data domain* relative to f is the set of elements of Dom which appear in database relations mentioned in f or as constants in f [Mai83]. The $\mathcal{L}$-constraint algebra expression for E can be found easily. ∎

Lemma 4.9.1 *Let e be a pure expression over $\widetilde{R}$ in modal $\mathcal{L}$-constraint algebra and e' be its corresponding $\mathcal{L}$-constraint algebra expression. Then*

$$sem(e(\widetilde{r})) = \{points(e'(\widetilde{r}')) : \widetilde{r}' \in rep(\widetilde{r})\}$$

for all indefinite $\mathcal{L}$-constraint databases $\widetilde{r}$ over $\widetilde{R}$.

Proof: This has already been shown explicitly in the proof of theorem 4.6.2. ∎

The following theorem demonstrates that safe expressions of relational calculus with $\mathcal{L}$-constraints over indefinite $\mathcal{L}$-constraint databases can also be evaluated bottom-up in closed form.

Theorem 4.9.2 *For every safe expression f over $\widetilde{R}$ in relational calculus with $\mathcal{L}$-constraints there exists a pure expression e over $\widetilde{R}$ in modal $\mathcal{L}$-constraint algebra such that the following property holds. If $\widetilde{r}$ is an indefinite $\mathcal{L}$-constraint database over $\widetilde{R}$ then $f(\widetilde{r}) = sem(e(\widetilde{r}))$.*

Proof: Let e' be the $\mathcal{L}$-constraint algebra expression provided by theorem 4.9.1 for f. Then e is the pure expression of modal $\mathcal{L}$-constraint algebra which corresponds to e'. The result holds because of the following equalities:

$$f(\widetilde{r}) = \{f(\widetilde{r}') : \widetilde{r}' \in rep(\widetilde{r})\} = \{points(e'(\widetilde{r}')) : \widetilde{r}' \in rep(\widetilde{r})\} = sem(e(\widetilde{r}))$$

The last equality follows from lemma 4.9.1. ∎

Finally we turn to safe expressions in modal relational calculus with $\mathcal{L}$-constraints.

Theorem 4.9.3 *Let f be a safe $\Box$-expression (resp. $\Diamond$-expression) over $\widetilde{R}$ in modal relational calculus with $\mathcal{L}$-constraints. Then there exists a $CERT$-expression (resp. $POSS$-expression) e over $\widetilde{R}$ in modal $\mathcal{L}$-constraint algebra such that the following property holds. If $\widetilde{r}$ is an indefinite $\mathcal{L}$-constraint database over $\widetilde{R}$ then $f(\widetilde{r}) = sem(e(\widetilde{r}))$.*

Proof: Let f' be the safe expression in relational calculus with $\mathcal{L}$-constraints which corresponds to $\Box$-expression f. From lemma 4.3.1 we have $f(\widetilde{r}) = \bigcap f'(\widetilde{r})$. Then theorem 4.9.2 provides a pure expression e' in modal $\mathcal{L}$-constraint algebra such that $f(\widetilde{r}) = \bigcap sem(e(\widetilde{r}))$. Finally, lemma 4.5.1 gives $f(\widetilde{r}) = sem(CERT(e'(\widetilde{r})))$ which proves the theorem for $\Box$-expressions. The proof for $\Diamond$-expressions is similar. ∎

4.10 Conclusion

This chapter laid the foundations of a theory of *indefinite constraint databases* based on the relational model [Mai83]. Among other things, it provided us with algorithms for evaluating queries over constraint databases and indefinite constraint databases. We expect these results to form the basis of query evaluation algorithms for instances of the psoposed schemes.

In the next chapter we present a modification of the schemes of this chapter where regular attributes are no longer available. The new schemes does not lose any of their expressive power. Atomic data values can still be represented by equality constraints over the data domain as shown in [KKR90].

Chapter 5

An Alternative Hierarchy of Constraint Database Schemes

In this chapter we present a simple modification of the constraint database schemes discussed in chapter 4. In the new schemes regular attributes have been discarded. Nothing is lost in the new model in terms of expressive power: data values, as found in the traditional relational model, can be represented by equality constraints over the data domain *Dom*. In the rest of this dissertation we will concentrate on the new schemes only.

The motivation behind the proposed modification is purely theoretical. In the new schemes there are *no unsafe formulas*: all queries in (modal) relational calculus with $\mathcal{L}$-constraints have answers that can be evaluated in closed form. This can be demonstrated by transforming query evaluation over indefinite $\mathcal{L}$-constraint databases into quantifier elimination in the theory of the structure $\mathbf{M}_{\mathcal{L}}$. This technique, originally suggested for $\mathcal{L}$-constraint databases by [KKR90], will also turn out to be important in chapter 7 where we study the complexity of query evaluation in (indefinite) temporal constraint databases.

This chapter is organized as follows. In section 5.1 we show how data values from an infinite domain can be handled using constraints. In section 5.2 we discuss the details of the new database schemes. Finally, in section 5.3 we establish the connection between query evaluation and quantifier elimination.

5.1 A Theory of Atomic Data Values

Let DCL be a first-order language with equality whose set of constants is the *infinite* set of data values *Dom*. The intended interpretation for DCL will be a *Herbrand* one: in the intended structure $\mathbf{D}$ constants are interpreted as themselves. We will take $Th(\mathbf{D})$ to be *the* theory of atomic data values over an infinite domain.

Definition 5.1.1 A DCL-*constraint* is a formula of the form $x = y, x \neq y, x = a$ or $x \neq a$ where x, y are variables and a is a constant.

The algorithm DCL-VarElimination of figure 5.1 achieves variable elimination for DCL-constraints. Its proof of correctness is straightforward. The same algorithm can be used for checking the consistency of a set of DCL-constraints. This can be done by eliminating all variables

Algorithm DCL-VarElimination

Input: A consistent set C of DCL-constraints in variables X. A set of variables $E \subseteq X$ to be eliminated from C.

Output: A set of DCL-constraints in variables $E \setminus X$ with solution set $Proj_{E \setminus X}(Sol(C))$.

Method:

<u>For</u> each variable $x \in E$ <u>do</u>
    <u>If</u> x appears in equation $x = t$ of C <u>then</u>
        Remove $x = t$ from C
        <u>For</u> each constraint $c \in C$ <u>do</u>
            $c := c[t/x]$
            <u>If</u> c is of the form $t = t$ <u>then</u>
                Remove c from C
            <u>EndIf</u>
        <u>EndFor</u>
    <u>Else</u> (x appears only in inequations)
        Remove all inequations containing x from C
    <u>EndIf</u>
<u>EndFor</u>
Output C

Figure 5.1: Variable Elimination for DCL-constraints

from the given constraint set. The constraint set is inconsistent if and only if the output of variable elimination contains an inequation of the form $a \neq a$ where a is a constant.

The proof of the following theorem is straightforward.

Theorem 5.1.1 *Let C be a set of DCL-constraints. We can eliminate m variables from C, using* DCL-VarElimination*, in $O(m\,|C|)$ time.*

Let us now introduce the languages diPDCL (*di*screte *P*oint and *D*ata *C*onstraint *L*anguage), diTDCL (*di*screte *T*ime and *D*ata *C*onstraint *L*anguage), dePDCL (*de*nse *P*oint and *D*ata *C*onstraint *L*anguage) and deTDCL (*de*nse *T*ime and *D*ata *C*onstraint *L*anguage). Each one of these languages (e.g., diPDCL) is the union of DCL and one of the languages of section 3.2 (e.g., diPCL). The reader can easily fill in the details since DCL shares no non-logical symbol with the languages of section 3.2. We would only like to notice that in all of the above languages the sort for data values will be denoted by $\mathcal{D}$. The intended structures for these languages will be denoted by $\mathbf{DZ}, \mathbf{DZI_Z}, \mathbf{DQ}$ and $\mathbf{DQI_Q}$. Naturally each one of these structures (e.g., $\mathbf{DZ}$) is the union of $\mathbf{D}$ and one of the structures of section 3.2 (e.g., $\mathbf{Z}$). Detailed definitions are omitted since they are straightforward.

5.2 The Hierarchy of Representation Schemes Revisited: Definitions, Semantics and Query Languages

Let us now modify the definitions of chapter 4 so that data values are also represented using constraints. The changes in the definitions are straightforward and are presented here only for completeness. $\mathcal{L}$ will again denote a many-sorted language and $\mathbf{M}_{\mathcal{L}}$ will be the intended $\mathcal{L}$-structure. We will assume that the class of $\mathcal{L}$-constraints admits variable elimination and is weakly closed under negation.

Let us first define relation schemes.

Definition 5.2.1 A *relation scheme* R is a finite subset of $\mathcal{U}$. If R is the scheme $\{A_1, \ldots, A_n\}$ it will also be written as $R(A_1, \ldots, A_n)$.

We will now define $\mathbf{M}_{\mathcal{L}}$-relational tuples and relations.

Definition 5.2.2 Let R be a relation scheme. An $\mathbf{M}_{\mathcal{L}}$*-relational tuple* t over scheme R is a mapping from R to $\bigcup_{s \in sorts(\mathcal{L})} dom(s, \mathbf{M}_{\mathcal{L}})$ such that $t(A) \in dom(sort(A), \mathbf{M}_{\mathcal{L}})$.

Definition 5.2.3 Let R be a relation scheme. An $\mathbf{M}_{\mathcal{L}}$*-relation* r over scheme R is an *unrestricted* set of $\mathbf{M}_{\mathcal{L}}$-relational tuples over R.

We will now define $\mathcal{L}$-constraint tuples and relations.

Definition 5.2.4 Let R be a relation scheme. An $\mathcal{L}$*-constraint tuple* t over scheme R is a mapping from $R \cup \{CON\}$ to $UVAR_{\mathcal{L}} \cup WFF(\mathcal{L})$ such that (i) $t(A) \in UVAR_{\mathcal{L}}^{sort(A)}$ for each $A \in R$, (ii) $t(A_i)$ is different than $t(A_j)$ for all distinct $A_i, A_j \in R$, (iii) $t(CON)$ is a conjunction of $\mathcal{L}$-constraints and (iv) the variables of $t(CON)$ are included in $\{t(A) : A \in R\}$.

If t is an $\mathcal{L}$-constraint tuple then $t(CON)$ is called the *local condition* of the tuple t while $t(R)$ is called the *proper part* of t.

Definition 5.2.5 Let R be a relation scheme. An $\mathcal{L}$*-constraint relation* r over R is a *finite* set of $\mathcal{L}$-constraint tuples over R.

Finally, we define indefinite $\mathcal{L}$-constraint tuples and relations.

Definition 5.2.6 Let R be a relation scheme. An *indefinite* $\mathcal{L}$*-constraint tuple* t over scheme R is a mapping from $R \cup \{CON\}$ to $UVAR_{\mathcal{L}} \cup WFF(\mathcal{L})$ such that (i) $t(A) \in UVAR_{\mathcal{L}}^{sort(A)}$ for each $A \in R$, (ii) $t(A_i)$ is different than $t(A_j)$ for all distinct A_i, $A_j \in R$, (iii) $t(CON)$ is a conjunction of $\mathcal{L}$-constraints and (iv) the free variables of $t(CON)$ are included in $\{t(A) : A \in R\} \cup EVAR_{\mathcal{L}}$.

For indefinite tuples, the notions of local condition and proper part are defined as for definite ones.

Definition 5.2.7 Let R be a relation scheme. An *indefinite* $\mathcal{L}$*-constraint relation* over scheme R is a *finite* set of indefinite $\mathcal{L}$-constraint tuples over R. Each indefinite $\mathcal{L}$-constraint relation r is associated with a boolean combination of $\mathcal{L}$-constraints $G(r)$. $G(r)$ is called the *global condition* of r.

The definitions of database schemes and databases remain as in section 4.2.

Let us now discuss the changes required to the semantics of section 4.2. The notions of e-valuation and u-valuation remain the same. The same is true for the result of applying an e-valuation to an indefinite $\mathcal{L}$-constraint relation. The result of applying a u-valuation of $\mathbf{M}_{\mathcal{L}}$ to the proper part of a tuple is now defined as follows. If t is an $\mathcal{L}$-constraint tuple on scheme R and u is a u-valuation in $\mathbf{M}_{\mathcal{L}}$ then $u(t)$ is an $\mathbf{M}_{\mathcal{L}}$-tuple over R such that for each $A \in R$, $u(t)(A) = u(t(A))$. The functions *points*, *rep* and *sem* can now be defined as in 4.2.

Finally let us consider query languages for the new scheme. Let $\widetilde{R}$ be a database scheme. The language $\mathcal{L}^+(\widetilde{R})$ can now be defined by extending $\mathcal{L}$ in the following way. For each scheme $R(C_1, \ldots, C_m)$ in $\widetilde{R}$ an m-place predicate symbol R of sort

$$sort(C_1) \times \cdots \times sort(C_m)$$

is introduced. These will be called *database predicate symbols.* $\mathcal{L}^+(\widetilde{R})$ will simply be written as $\mathcal{L}^+$ when the database predicate vocabulary is of no importance.

The symbols of $\mathcal{L}^+(\widetilde{R})$ are interpreted by a many-sorted $\mathcal{L}^+(\widetilde{R})$-structure $\mathbf{M}_{\mathcal{L}^+} = (\mathbf{M}_{\mathcal{L}}, \widetilde{r})$ as follows:

- To the symbols of $\mathcal{L}^+(\widetilde{R})$ which are also symbols of $\mathcal{L}$, $\mathbf{M}_{\mathcal{L}^+}$ makes the same assignments as the structure $\mathbf{M}_{\mathcal{L}}$.
- To the finite sequence $\widetilde{p}$ of database predicate symbols, $\mathbf{M}_{\mathcal{L}^+}$ assigns the sequence of $\mathbf{M}_{\mathcal{L}}$-relations $\widetilde{r}$.

The (modal) relational calculus with $\mathcal{L}$-constraints can now be defined as in section 4.3. Since regular attributes do not interfere with variable attributes, it is easy to modify the definitions for the algebraic query languages of sections 4.4 and 4.5. The rest of the development of chapter 4 can now be carried out in the new framework. Details are omitted since they are straightforward.

In the rest of this dissertation we will consider $\mathcal{L}$-constraint databases and indefinite $\mathcal{L}$-constraint databases where $\mathcal{L}$ can be any of the languages diPCL, diTCL, diPDCL, diTDCL, dePCL, deTCL, dePDCL or deTDCL. We will also refer to these models collectively as *temporal constraint databases* and *indefinite temporal constraint databases.*

5.3 Query Evaluation as Quantifier Elimination

In this section we provide an alternative (proof-theoretic) semantics for query evaluation over indefinite $\mathcal{L}$-constraint databases. We show that query evaluation over indefinite $\mathcal{L}$-constraint databases can be viewed as quantifier elimination in the theory of the structure $\mathbf{M}_{\mathcal{L}}$. This technique was originally presented in [KKR90] for $\mathcal{L}$-constraint databases. Quantifier elimination is always possible in our framework since the class of $\mathcal{L}$-constraints admits variable elimination and is weakly closed under negation. The importance of our results will become apparent in chapter 7 where we study the complexity of query evaluation in temporal constraint databases.

Notation 5.3.1 We will use $\overline{x}, \overline{y}, \overline{z}, \ldots$ to represent vectors of variables of $\mathcal{L}$ and $\overline{a}, \overline{b}, \overline{c}, \ldots$ to represent vectors of elements of the set $\cup_{s \in sorts(\mathcal{L})} dom(s, \mathbf{M}_{\mathcal{L}})$. We will also use $\overline{X}, \overline{Y}, \overline{Z}, \ldots$ to represent vectors of attributes of relations. For example,

$$\{R(\overline{X}), \overline{x}/\overline{s} : \ \phi(\overline{x})\}$$

will represent an expression in relational calculus with $\mathcal{L}$-constraints. In this case s_i is assumed to be an element of $sorts(\mathcal{L})$ and $s_i = sort(X_i)$ for all $i = 1, \ldots, |\overline{x}|$.

The following definitions are from [KKR90].

Definition 5.3.1 Let $r = \{t_1, \ldots, t_m\}$ be an $\mathcal{L}$-constraint relation over scheme $R(A_1, \ldots, A_n)$. The formula of $\mathcal{L}$ *corresponding* to r is the disjunction

$$t_1(CON)[\overline{x}/t_1(R)] \ \vee \ \cdots \ \vee \ t_m(CON)[\overline{x}/t_m(R)]$$

where $\overline{x}$ is an n-vector of new variables from $VAR_{\mathcal{L}}$ such that $sort(x_i) = sort(A_i)$ for all $i = 1, \ldots, n$.

The formula of $\mathcal{L}$ corresponding to an indefinite $\mathcal{L}$-constraint relation can be defined similarly.

Example 5.3.1 Consider the following deTDCL-constraint relation ON_SERVICE.

ON_SERVICE

VEH	CITY	TIME	CON
v_1	c_1	i_1	$v_1 = V1,\ c_1 = Athens,\ 9 \leq i_{1L},\ i_{1R} \leq 15$
v_2	c_2	i_2	$v_2 = V2,\ c_2 = Athens,\ 10 \leq i_{2L},\ i_{2R} \leq 14$

The deTDCL formula corresponding to ON_SERVICE is:

$$(v = V1 \ \wedge\ c = Athens \ \wedge\ 9 \leq i_L \ \wedge\ i_R \leq 15) \ \vee\ (v = V2 \ \wedge\ c = Athens \ \wedge\ 10 \leq i_L \ \wedge\ i_R \leq 14)$$

where v, c are new variables of sort $\mathcal{D}$, and i is a new variable of sort $\mathcal{I}_{\mathcal{Q}}$.

Definition 5.3.2 Let $\widetilde{r}$ be an $\mathcal{L}$-constraint database over scheme $\widetilde{R} = (R_1, \ldots, R_n)$ and ϕ be a $\mathcal{L}^+(\widetilde{R})$ formula. The formula of $\mathcal{L}$ *corresponding* to ϕ and $\widetilde{r}$ can be obtained from ϕ in the following way. If R_i is a database predicate and $\overline{\alpha}$ is a vector of terms of $\mathcal{L}^+$ then substitute every subformula $R_i(\overline{\alpha})$ of ϕ by the formula $\psi_i[\overline{\alpha}/\overline{x}]$ where ψ_i is the formula of $\mathcal{L}$ corresponding to relation r_i of $\widetilde{r}$.

If $\widetilde{r}$ is an indefinite $\mathcal{L}$-constraint database then the definition of the formula of $\mathcal{L}$ corresponding to ϕ and $\widetilde{r}$ is similar.

Example 5.3.2 Consider the deTDCL$^+$ formula

$$\phi(x) \equiv (\exists t/\mathcal{I}_{\mathcal{Q}})(ON\_SERVICE(x, Athens, t) \ \wedge\ t_L \leq 12 \ \wedge\ t_R \leq 13)$$

and the deTCL-constraint relation ON_SERVICE of the previous example. The deTDCL formula corresponding to ϕ and ON_SERVICE is the following:

$$(\exists t/\mathcal{I}_{\mathcal{Q}})(((x = V1 \ \wedge\ Athens = Athens \ \wedge\ 9 \leq t_L \ \wedge\ t_R \leq 15) \ \vee$$
$$(x = V2 \ \wedge\ Athens = Athens \ \wedge\ 10 \leq t_L \ \wedge\ t_R \leq 14)) \wedge\ t_L \leq 12 \ \wedge\ t_R \leq 13)$$

The following theorem is from [KKR90].

Theorem 5.3.1 *Let $\widetilde{r}$ be an $\mathcal{L}$-constraint database over $\widetilde{R}$ and f be the expression*

$$\{R(\overline{X}), \overline{x}/\overline{s} : \ \phi(\overline{x})\}$$

over $\widetilde{R}$ in relational calculus with $\mathcal{L}$-constraints. Then

$$f(\widetilde{r}) = \{ \ \overline{a} : \ \mathbf{M}_{\mathcal{L}} \models \psi[\overline{x} \leftarrow \overline{a}] \ \}$$

where ψ is the formula of $\mathcal{L}$ corresponding to ϕ and $\widetilde{r}$.

Proof: Use induction on the structure of the formula ϕ. ∎

The above theorem implies that queries in relational calculus with $\mathcal{L}$-constraints over $\mathcal{L}$-constraint databases can be evaluated in closed form. All we have to do is eliminate quantifiers from formula ψ of $\mathcal{L}$. This is possible since the class of $\mathcal{L}$-constraints is weakly closed under negation and admits variable elimination. This is established formally by proposition 6.1.2 of the next chapter. The resulting formula in DNF can be turned into an $\mathcal{L}$-constraint relation which is the answer to the query f (definition 5.3.1).

Example 5.3.3 Consider query

$$\{ANS(V),\ x/\mathcal{D}:\ (\exists t/\mathcal{I_Q})(ON\_SERVICE(x, Athens, t)\ \wedge\ t_L \leq 12\ \wedge\ t_R \leq 13)\}$$

over deTCL-constraint relation ON_SERVICE of example 5.3.1.

This query can be answered by eliminating quantifier $(\exists t)$ from the deTDCL formula

$$(\exists t/\mathcal{I_Q})(((x = V1\ \wedge\ Athens = Athens\ \wedge\ 9 \leq t_L\ \wedge\ t_R \leq 15) \vee$$
$$(x = V2\ \wedge\ Athens = Athens\ \wedge\ 10 \leq t_L\ \wedge\ t_R \leq 14))\wedge\ t_L \leq 12\ \wedge\ t_R \leq 13)$$

The answer to the query is given by the resulting quantifier-free formula $x = V1\ \vee\ x = V2$.

The following theorem extend the above result of [KKR90] to the case of indefinite $\mathcal{L}$-constraint databases and queries in relational calculus with $\mathcal{L}$-constraints.

Theorem 5.3.2 *Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database over $\widetilde{R}$. Let $G(\overline{\omega})$ be the global condition of $\widetilde{r}$ where $\overline{\omega}$ is the vector of e-variables. Let f be the expression*

$$\{R(\overline{X}), \overline{x}/\overline{s}:\ \phi(\overline{x})\}$$

over $\widetilde{R}$ in relational calculus with $\mathcal{L}$-constraints. Then

$$f(\widetilde{r}) = \{\ \{\ \overline{a}:\ \mathbf{M}_{\mathcal{L}} \models \psi(\overline{x}, \overline{\omega})[\overline{x} \leftarrow \overline{a}, \overline{\omega} \leftarrow \overline{\omega_0}]\ \}\ \ :\mathbf{M}_{\mathcal{L}} \models G[\overline{\omega} \leftarrow \overline{\omega_0}]\ \}$$

where $\psi(\overline{x}, \overline{\omega})$ is the formula of $\mathcal{L}$ which corresponds to ϕ and $\widetilde{r}$.

Proof: Straightforward. Use the previous theorem. ■

The following theorems consider the case of indefinite $\mathcal{L}$-constraint databases and queries in modal relational calculus with $\mathcal{L}$-constraints.

Theorem 5.3.3 *Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database over $\widetilde{R}$. Let $G(\overline{\omega})$ be the global condition of $\widetilde{r}$ where $\overline{\omega}$ is a vector of e-variables of sort $\overline{s}'$. Let f be the expression*

$$\{R(\overline{X}), \overline{x}/\overline{s}:\ \Diamond\phi(\overline{x})\}$$

over $\widetilde{R}$ in modal relational calculus with $\mathcal{L}$-constraints. Then

$$f(\widetilde{r}) = \{\ \{\ \overline{a}:\ \mathbf{M}_{\mathcal{L}} \models (\exists \overline{z}/\overline{s}')(G[\overline{z}/\overline{\omega}]\ \wedge\ \psi'(\overline{x}, \overline{z}))[\overline{x} \leftarrow \overline{a}]\ \}\ \}$$

where $\psi'(\overline{x}, \overline{z})$ is a formula of $\mathcal{L}$ which is the same as the formula which corresponds to ϕ and $\widetilde{r}$, except that $\overline{z}$ has been substituted for $\overline{\omega}$.

Proof: The following equivalences prove the theorem.

$\overline{a} \in f(\widetilde{r})$ iff
there exists $\widetilde{r}' \in sem(\widetilde{r})$ such that $(\mathbf{M}_{\mathcal{L}},\ \widetilde{r}') \models \phi[\overline{x} \leftarrow \overline{a}]$ iff
there exists valuation $v = (\overline{\omega} \leftarrow \overline{b}) \in Val^{e}_{\mathbf{M}_{\mathcal{L}}}$ such that $\mathbf{M}_{\mathcal{L}} \models G[v]$ and
$(\mathbf{M}_{\mathcal{L}},\ points(v(\widetilde{r})) \models \phi[\overline{x} \leftarrow \overline{a}]$ iff
there exists $\overline{b}$ such that $\mathbf{M}_{\mathcal{L}} \models G[\overline{\omega} \leftarrow \overline{b}]$ and $\mathbf{M}_{\mathcal{L}} \models \psi''[\overline{x} \leftarrow \overline{a}]$ iff
(assuming that $\psi''(\overline{x})$ is the formula of $\mathcal{L}$ corresponding to ϕ and $v(\widetilde{r})$)
there exists $\overline{b}$ such that $\mathbf{M}_{\mathcal{L}} \models G[\overline{\omega} \leftarrow \overline{b}]$ and $\mathbf{M}_{\mathcal{L}} \models \psi'[(\overline{x}, \overline{z}) \leftarrow (\overline{a}, \overline{b})]$ iff
$\mathbf{M}_{\mathcal{L}} \models (\exists \overline{z}/\overline{s}')(G[\overline{z}/\overline{\omega}]\ \wedge\ \psi'(\overline{x}, \overline{z}))[\overline{x} \leftarrow \overline{a}]$.

■

Theorem 5.3.4 *Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database over $\widetilde{R}$. Let $G(\overline{\omega})$ be the global condition of $\widetilde{r}$ where $\overline{\omega}$ is a vector of e-variables of sort $\overline{s}'$. Let f be the expression*

$$\{R(\overline{X}), \overline{x}/\overline{s} : \ \Box\phi(\overline{x})\}$$

over $\widetilde{R}$ in modal relational calculus with $\mathcal{L}$-constraints. Then

$$f(\widetilde{r}) = \{ \ \{ \ \overline{a} : \ \mathbf{M}_{\mathcal{L}} \models (\forall \overline{z}/\overline{s}')(G[\overline{z}/\overline{\omega}] \ \supset \ \psi'(\overline{x},\overline{z}))[\overline{x} \leftarrow \overline{a}] \ \} \ \}$$

where $\psi'(\overline{x},\overline{z})$ is a formula of $\mathcal{L}$ which is the same as the formula which corresponds to ϕ and $\widetilde{r}$, except that $\overline{z}$ has been substituted for $\overline{\omega}$.

Proof: Similar to the proof of the previous theorem. ■

The above theorems imply that queries in modal relational calculus with $\mathcal{L}$-constraints over indefinite $\mathcal{L}$-constraint databases can be evaluated in closed form.

5.4 Conclusion

We presented a simple modification of the three database schemes discussed in chapter 4 where regular attributes are no longer present. This does not result in any loss of expressive power because atomic data values could still be represented by equality constraints over the data domain. In the new schemes there are *no unsafe formulas*: all queries in modal relational calculus with $\mathcal{L}$-constraints can be evaluated bottom-up in closed form. This was demonstrated by transforming query evaluation over indefinite $\mathcal{L}$-constraint databases into quantifier elimination in the theory of the structure $\mathbf{M}_{\mathcal{L}}$. In the next chapter we discuss the concept of quantifier elimination in more detail.

Chapter 6

Quantifier Elimination for Theories of Data and Time

In this chapter we present quantifier elimination procedures for the theories of data and time introduced in sections 3.2 and 5.1. Initially we discuss naive quantifier elimination algorithms based on standard techniques developed by logicians [End72, CK73]. These algorithms give us information which is used later on for developing more efficient algorithms.

The most important contributions of this chapter are the following:

- We show that elimination of quantifiers in diPC and dePC can be performed in PSPACE (sections 6.4 and 6.5). The decision problem for these two theories is PSPACE-complete.
- The decision problem for sentences of diPC or dePC with k alternations of quantifiers starting with an existential quantifier is Σ_k^p-complete.

Since diPC is a subtheory of Presburger arithmetic and dePC is a subtheory of real addition with order these results must be of independent interest to the theoretical computer science community. Previous results on the same line of research include [FR74, FR75, FR79, FG77, RL78].

This chapter is organized as follows. The next section gives some preliminary definitions and results. In section 6.2 we present naive algorithms based on standard quantifier elimination techniques. Then we develop more efficient quantifier elimination algorithms in sections 6.3, 6.4, 6.5, and 6.6.

6.1 Preliminaries

Let us start by defining our terminology.

Definition 6.1.1 Let Th be a theory in some language $\mathcal{L}$. *Th admits elimination of quantifiers* iff for every formula ϕ there is a disjunction ϕ' of conjunctions of $\mathcal{L}$-constraints such that $Th \models \phi \equiv \phi'$.

The above notion of quantifier-elimination is stronger than the traditional one [End72, CK73] where ϕ' is simply required to be quantifier-free. We require ϕ' to be in the above form because we do not want to deal with negations of $\mathcal{L}$-constraints.

The following proposition shows that if the class of $\mathcal{L}$-constraints is weakly closed under negation then we can determine whether Th admits elimination of quantifiers by considering only conjunctions of $\mathcal{L}$-constraints.

Proposition 6.1.1 *Let Th be a theory in some language $\mathcal{L}$. Th admits elimination of quantifiers if*

1. *the class of $\mathcal{L}$-constraints is weakly closed under negation, and*
2. *for every formula $(\exists x)\theta$ where θ is a conjunction of $\mathcal{L}$-constraints, there is a disjunction θ' of conjunctions of $\mathcal{L}$-constraints such that $Th \models (\exists x)\theta \equiv \theta'$.*

Proof: The proof follows [CK73]. We will show by induction that every formula ϕ of $\mathcal{L}$ is equivalent to a disjunction of conjunctions of $\mathcal{L}$-constraints.

For the base case, let ϕ be an atomic formula. Then the result is trivial because atomic formulas of $\mathcal{L}$ are included in the class of $\mathcal{L}$-constraints (by definition).

For the inductive step, we have to consider the following cases:

- ϕ is $\neg\psi$ and ψ is equivalent to a disjunction of conjunctions of $\mathcal{L}$-constraints. Then the result follows from De Morgan's laws, distributive laws and the fact that the class of $\mathcal{L}$-constraints is weakly closed under negation.
- ϕ is $\psi_1 \vee \psi_2$ and ψ_1, ψ_2 are equivalent to a disjunction of conjunctions of $\mathcal{L}$-constraints. In this case the result is obvious.
- ϕ is $(\exists x)\psi$ and ψ is equivalent to a disjunction $\theta_1 \vee \cdots \vee \theta_m$ where all θ_i's are conjunctions of $\mathcal{L}$-constraints. In this case ϕ is equivalent to

$$(\exists x)\theta_1 \vee \cdots \vee (\exists x)\theta_m$$

and the result follows from the second assumption of the theorem.

■

The following proposition establishes an obvious relationship between variable elimination and quantifier-elimination.

Proposition 6.1.2 *Let Th be a theory in some language $\mathcal{L}$. Th admits elimination of quantifiers if and only if the class of $\mathcal{L}$-constraints admits variable elimination and is weakly closed under negation.*

We will now develop quantifier elimination algorithms for all the theories of data and time considered in sections 3.2 and 5.1.

6.2 Naive Quantifier Elimination Algorithms

We will first consider the theory DC introduced in section 5.1.

Proposition 6.2.1 *The theory DC admits elimination of quantifiers.*

Proof: Let us assume that ϕ is a formula of DCL. We can find a quantifier-free formula ϕ' equivalent to ϕ in the following way:

1. Compute the PNF $(Qx_1)\cdots(Qx_m)\psi(x_1,\ldots,x_m)$ of ϕ.

2. Eliminate quantifier (Qx_m) as follows. Let Q be $\exists$ and $\theta_1 \vee \cdots \vee \theta_k$ be the DNF of $\psi(x_1, \ldots, x_m)$. Use algorithm DCLVarElimination of section 5.1 to eliminate variable x_m from each θ_i to obtain θ_i'. Then simplify θ_i' by substituting atomic formulas of the form $a = a$ by $true$ and $a \neq a$ by $false$ (where a is a constant or a variable). If θ_i' is empty then set $\theta_i' = true$. The resulting expression is $\theta_i' \vee \cdots \vee \theta_k'$.

   Now let us assume Q is $\forall$ and $\theta_1 \vee \cdots \vee \theta_k$ is the DNF of $\neg\psi(x_1, \ldots, x_m)$. Eliminate variable x_m from each θ_i to compute θ_i' as above. The resulting expression is $\neg(\theta_i' \vee \cdots \vee \theta_k')$.

3. Repeat step 2 to eliminate quantifiers $(Qx_{m-1}), \ldots, (Qx_1)$ and obtain a quantifier-free formula in DNF.

4. Simplify the above DNF formula by performing the following substitutions repeatedly.

   - Substitute $true \wedge \alpha$ by α.
   - Substitute $true \vee \alpha$ by $true$.
   - Substitute $false \wedge \alpha$ by $false$.
   - Substitute $false \vee \alpha$ by α.

The correctness of the above procedure can be verified easily. Similar proofs can be found in standard logic textbooks [End72, CK73]. ■

The following theorem estimates the complexity of the quantifier elimination algorithm of the previous proposition.

Theorem 6.2.1 *Let ϕ be a DCL formula with K variables and M quantifiers. The quantifier elimination algorithm of proposition 6.2.1 computes a quantifier-free DNF formula equivalent to ϕ in $O(M(K^2 + |\phi|\, K)2^{K^2+|\phi|K})$ time.*

Proof: Let l be the number of constants in ϕ. Obviously, $l \leq |\phi|$. In a conjunction of data constraints we can have at most $K(K-1)/2$ equations of the form $x_i = x_j$, K equations of the form $x_i = c$, $K(K-1)/2$ inequations of the form $x_i \neq x_j$ and lK inequations of the form $x_i \neq c$ (where x_i, x_j are variables and c is a constant). Thus the total number of constraints in a conjunction can be at most $K^2 + lK$. Then there can be at most 2^{K^2+lK} different disjuncts in any DNF formula formed by the algorithm of proposition 6.2.1. The time needed for eliminating a variable is $O(K^2 + lK)$. Finally, eliminating M quantifiers takes $O(M(K^2 + |\phi|\, K)2^{K^2+|\phi|K})$ time. ■

Let us now turn to theories of time.

Proposition 6.2.2 *The theories diPC and dePC admit elimination of quantifiers.*

Proof: Let us assume that ϕ is a formula of diPCL or dePCL. We can find a quantifier-free formula ϕ' equivalent to ϕ in the following way:

1. Compute the PNF $(Qt_1) \cdots (Qt_m)\psi(t_1, \ldots, t_m)$ of ϕ.

2. Eliminate quantifier (Qt_m) as follows. Let Q be $\exists$ and $\theta_1 \vee \cdots \vee \theta_k$ be the DNF of $\psi(t_1, \ldots, t_m)$. In the case of a dePCL formula, eliminate variable t_m from each θ_i to compute θ_i' using algorithm PCL-VarElimination of section 3.5. In the case of a diPCL formula, first transform every strict inequality into a weak one and then apply PCLVarElimination. The resulting expression is $\theta_1' \vee \cdots \vee \theta_k'$.

   Now let us assume Q is $\forall$ and $\theta_1 \vee \cdots \vee \theta_k$ is the DNF of $\neg\psi(t_1, \ldots, t_m)$. Eliminate variable t_m from each θ_i to compute θ_i' as above. The resulting expression is $\neg(\theta_i' \vee \cdots \vee \theta_k')$.

3. Repeat step 2 to eliminate quantifiers $(Qt_{m-1}), \ldots, (Qt_1)$ to obtain a quantifier-free formula in DNF.

4. Finally simplify the above DNF formula as in proposition 6.2.1.

The correctness of the above procedure can be verified easily. For diPC a similar method appears in [Rab77]. ■

The following theorem estimates the complexity of the quantifier elimination algorithm of the previous proposition.

Theorem 6.2.2 *Let ϕ be a diPCL or dePCL formula with K variables and M quantifiers. The quantifier elimination algorithm of proposition 6.2.2 computes a quantifier-free DNF formula equivalent to ϕ in $O(MK^2\,|\phi|^{(K(K+1))^M})$ time.*

Proof: The formula $\psi(t_1, \ldots, t_m)$ can have at most $K(K+1)$ types of constraints (lemma 3.5.1) and at most $|\psi|$ constraints of each type. Therefore the DNF form of ψ can contain $O(|\psi|^{K(K+1)})$ disjuncts and each disjunct can contain at most $K(K+1)$ constraints. Finally, after M quantifier eliminations the resulting formula can have $O(|\phi|^{(K(K+1))^M})$ disjuncts. The time to eliminate a variable from a disjunct is $O(K^2)$ (theorem 3.5.2). The result follows easily. ■

Proposition 6.2.3 *The theories diTC and deTC admit elimination of quantifiers.*

Proof: Let us assume that ϕ is a formula of diTCL. We can find a quantifier-free formula equivalent to ϕ in the following way:

1. Compute the diPCL formula $itp(\phi)$ as defined in section 3.2.3.

2. Perform quantifier elimination in formula $itp(\phi)$ using the method of proposition 6.2.2. Let the result be the diPCL formula $ipt(\phi)^*$ in DNF.

3. The result is the diTCL formula $pti(itp(\phi)^*)$ in DNF.

The correctness of the above procedure can be verified easily. Let us assume that the free variables of ϕ are $p_1, \ldots, p_n$ of sort $\mathcal{Z}$ and $i_1, \ldots, i_m$ of sort $\mathcal{I}_{\mathcal{Z}}$. The following equivalences prove the result:

$$\begin{array}{lll}
\mathbf{ZI_Z} & \models & \phi[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, i_1 \leftarrow i_1^0, \ldots, i_m \leftarrow i_m^0] \hfill \text{iff} \\
\mathbf{Z} & \models & itp(\phi)[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, B_{i_1} \leftarrow l(i_1^0), \ldots, B_{i_m} \leftarrow l(i_m^0), \\
 & & \quad E_{i_1} \leftarrow r(i_1^0), \ldots, E_{i_m} \leftarrow r(i_m^0)] \hfill \text{iff} \\
\mathbf{Z} & \models & itp(\phi)^*[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, B_{i_1} \leftarrow l(i_1^0), \ldots, B_{i_m} \leftarrow l(i_m^0), \\
 & & \quad E_{i_1} \leftarrow r(i_1^0), \ldots, E_{i_m} \leftarrow r(i_m^0)] \hfill \text{iff} \\
\mathbf{ZI_Z} & \models & pti(itp(\phi)^*)[p_1 \leftarrow p_1^0, \ldots, p_n \leftarrow p_n^0, i_1 \leftarrow i_1^0, \ldots, i_m \leftarrow i_m^0]
\end{array}$$

The first "iff" follows from proposition 3.2.1, the second one from proposition 6.2.2 and the third one from the definition of translation function itp. The quantifier elimination procedure for deTC is similar. ■

[Lad88] has previously proposed a very similar quantifier elimination algorithm for a less expressive first-order theory of intervals based on Allen's relations.

The following theorem estimates the complexity of the quantifier elimination algorithm of the previous proposition.

Theorem 6.2.3 *Let ϕ be a diTCL (resp. deTCL) formula with K variables of sort $\mathcal{Z}$ or $\mathcal{I}_{\mathcal{Z}}$ (resp. $\mathcal{Q}$ or $\mathcal{I}_{\mathcal{Q}}$), and M quantifiers of sort $\mathcal{Z}$ or $\mathcal{I}_{\mathcal{Z}}$ (resp. $\mathcal{Q}$ or $\mathcal{I}_{\mathcal{Q}}$). The quantifier elimination algorithm of proposition 6.2.3 computes a quantifier-free DNF formula equivalent to ϕ in*

$$O(MK^2\ (2\,|\phi|)^{(2K(2K+1))^M})$$

time.

Proof: The transformations of proposition 6.2.3 can at most double the number of variables of sort $\mathcal{Z}$ (resp. $\mathcal{Q}$) and the size of ϕ. Then the result follows from theorem 6.2.2. ■

Let us now turn to theories of data and time.

Proposition 6.2.4 *The theories diPDC and dePDC admit elimination of quantifiers.*

Proof: We can easily devise a quantifier elimination algorithm for diPDC by combining the algorithms for DC and diPC given in the previous propositions.

Let us assume that ϕ is a formula of diPDCL. We can find a quantifier-free formula ϕ' equivalent to ϕ in the following way:

1. Compute the PNF $(Qz_1/s_1)\cdots(Qz_m/s_m)\psi(z_1,\ldots,z_m)$ of ϕ.
2. Eliminate quantifier (Qz_m/s_m) as follows. Let Q be $\exists$ and

   $$(\sigma_1\ \wedge\ \theta_1)\ \vee\ \cdots\ \vee\ (\sigma_k\ \wedge\ \theta_k)$$

   be the DNF of $\psi(z_1,\ldots,z_m)$ where σ_i is a formula of DCL and θ_i is a formula of diPCL for all $i=1,\ldots,k$.

   If $s_m = \mathcal{D}$ then eliminate variable z_m from each σ_i to compute σ_i' as in proposition 6.2.1. The resulting expression is

   $$(\sigma_1'\ \wedge\ \theta_1)\ \vee\ \cdots\ \vee\ (\sigma_k'\ \wedge\ \theta_k).$$

   If $s_m = \mathcal{Z}$ then eliminate variable z_m from each θ_i to compute θ_i' as in proposition 6.2.2. The resulting expression is

   $$(\sigma_1\ \wedge\ \theta_1')\ \vee\ \cdots\ \vee\ (\sigma_k\ \wedge\ \theta_k').$$

   If Q is $\forall$ then proceed analogously.
3. Repeat step 2 to eliminate quantifiers $(Qz_{m-1}/s_{m-1}),\ldots,(Qz_1/s_1)$ to obtain a quantifier-free formula in DNF.
4. Finally simplify the above DNF formula as in proposition 6.2.1.

The correctness of the above procedure can be verified easily. The quantifier elimination algorithm for dePDC is similar. ■

The following theorem estimates the complexity of the quantifier elimination algorithm of the previous proposition. Its proof is an easy consequence of theorems 6.2.1 and 6.2.2.

Theorem 6.2.4 *Let ϕ be a diPDCL (resp. dePDCL) formula with K variables of sort $\mathcal{D}$, L variables of sort $\mathcal{Z}$ (resp. $\mathcal{Q}$), M quantifiers of sort $\mathcal{D}$, and N quantifiers of sort $\mathcal{Z}$ (resp. $\mathcal{Q}$). The quantifier elimination algorithm of proposition 6.2.4 computes a quantifier-free DNF formula equivalent to ϕ in*

$$O((M(K^2+|\phi|\,K)+NL^2)\ 2^{K^2+|\phi|K}\ \ |\phi|^{(L(L+1))^N})$$

time.

Proposition 6.2.5 *The theories diTDC and deTDC admit elimination of quantifiers.*

Proof: Let us assume that ϕ is a formula of diTDCL (resp. deTDCL). We can find a quantifier-free formula equivalent to ϕ in the following way:

1. Compute the diPDCL (resp. dePDCL) formula $itp(\phi)$ by extending the definition of translation function *itp* to diTDCL (resp. deTDCL) formulas in the obvious way.

2. Perform quantifier elimination in formula $itp(\phi)$ using the method of proposition 6.2.4. Let the result be the diPDCL (resp. dePDCL) formula $itp(\phi)^*$ in DNF.

3. Compute the diTDCL (resp. deTDCL) formula $pti(itp(\phi)^*)$ in DNF by extending the definition of translation function *pti* to diPDCL (resp. dePDCL) formulas in the obvious way.

The correctness of the above procedure can be verified easily. ■

The following theorem estimates the complexity of the quantifier elimination algorithm of the previous proposition.

Theorem 6.2.5 *Let ϕ be a diTDCL (resp. deTDCL) formula with K variables of sort $\mathcal{D}$, L variables of sort $\mathcal{Z}$ or $\mathcal{I}_{\mathcal{Z}}$ (resp. $\mathcal{Q}$ or $\mathcal{I}_{\mathcal{Q}}$), M quantifiers of sort $\mathcal{D}$, and N quantifiers of sort $\mathcal{Z}$ or $\mathcal{I}_{\mathcal{Z}}$ (resp. $\mathcal{Q}$ or $\mathcal{I}_{\mathcal{Q}}$). The quantifier elimination algorithm of proposition 6.2.5 computes a quantifier-free DNF formula equivalent to ϕ in*

$$O((M(K^2 + |\phi|\,K) + NL^2)\;2^{K^2+|\phi|K}\;(2\,|\phi|)^{(2L(2L+1))^N})$$

time.

Proof: The transformations of proposition 6.2.5 can at most double the number of variables of sort $\mathcal{Z}$ (resp. $\mathcal{Q}$) and the size of ϕ. Then the result follows from theorem 6.2.4. ■

A quantifier elimination algorithm for a theory Th is also a decision procedure for Th. If ϕ is a sentence of Th and ϕ' is a quantifier-free formula equivalent to ϕ then ϕ' is a boolean combination of formulas $true$ and $false$ and can be decided very easily. Therefore the following theorem is a simple consequence of the previous propositions.

Theorem 6.2.6 *The theories DC, diPC, dePC, diTC, deTC, diPDC, dePDC, diTDC and deTDC are decidable. The decision procedures for these theories have been described in propositions 6.2.1, 6.2.2, 6.2.3, 6.2.4 and 6.2.5. Upper bounds on the complexity of these procedures are given by theorems 6.2.1, 6.2.2, 6.2.3, 6.2.4 and 6.2.5.*

The quantifier elimination algorithms of this section are simple but inefficient. The problem with these algorithms is the transformation into DNF; this creates an exponential blow-up in time and space complexity. We will now turn to more efficient algorithms that do not require transformations into DNF.

6.3 Quantifier Elimination For the Theory DC Revisited

We discuss an efficient quantifier elimination algorithm for the theory DC. The algorithm was originally presented in [KKR90]. It will also be described here only for completeness.

We will first consider sentences of DCL. Let us assume that we want to decide the truth of sentence $(Q\overline{x})\phi(\overline{x})$. We partition the space $Dom^{|\overline{x}|}$ into a finite number of *regions* such that the truth value of ϕ is the same for all *points* in the same region. Then we determine the truth value of $(Q\overline{x})\phi(\overline{x})$ by determining the truth value of ϕ on this *finite* number of regions.

Remark 6.3.1 The following definitions are given with respect to a DCL formula ϕ. The set D_ϕ is the set of elements of *Dom* which appear in ϕ. The special symbol $\circ \notin Dom$ will stand for some unspecified element of *Dom* different than all elements of *Dom* which appear in ϕ.

At first we define the concept of *data configuration (d-configuration)*. This is essentially the notion of e-configuration of [KKR90]. A d-configuration of size n is a unique representation of a region of Dom^n.

Definition 6.3.1 A d-configuration $\delta = (\overline{e}, \overline{v})$ of size n consists of an equivalence relation $\overline{e}$ on $\{1, \ldots, n\}$ and a sequence $\overline{v} = \{v_1, \ldots, v_n\}$ such that:

1. v_i is in $D_\phi \cup \{\circ\}$ for all i such that $0 \leq i \leq n$.
2. If $i \; \overline{e} \; j$ then $v_i = v_j$.
3. If $v_i = v_j \neq \circ$ then $i \; \overline{e} \; j$.

The empty d-configuration will be denoted by $\delta^\emptyset$.

We will now define the concept of a formula corresponding to a d-configuration. This concept will be useful for dealing with open DCL formulas.

Definition 6.3.2 Let $\delta = (\overline{e}, \overline{v})$ be a d-configuration of size n. The formula with free variables $\overline{x} = (x_1, \ldots, x_n)$ corresponding to δ is the conjunction of the following DCL formulas:

1. $x_i = x_j$ if $i \; \overline{e} \; j$.
2. $x_i \neq x_j$ if i and j are in different equivalence classes of $\overline{e}$.
3. $x_i = v_i$ if $v_i \neq \circ$.
4. $x_i \neq v$ (for all $v \in D_\phi$) if $v_i = \circ$.

This formula will be denoted by $\Phi(\delta)$.

We now define the concept of an *extension* of a d-configuration.

Definition 6.3.3 Let $\delta = (\overline{e}, \overline{v})$ be a d-configuration of size n. A d-configuration $\delta' = (\overline{e}', \overline{v}')$ of size $n+1$ is called an extension of δ iff

1. For all i and j such that $1 \leq i, j \leq n$, $i \; \overline{e} \; j$ iff $i \; \overline{e}' \; j$.
2. $\overline{v}' = (v_1, \ldots, v_n, v'_{n+1})$.

Figure 6.1 presents the decision procedure DC-Eval which is based on the above idea. The correctness of this procedure is shown in section 4 of [KKR90]. If we have a sentence ϕ of *DC* then $\mathbf{D} \models \phi$ iff DC-Eval$(\phi, \delta^\emptyset)$ returns 1. The reader can consult figure 6.1 and [KKR90] for more details.

Remark 6.3.2 In the subsequent complexity analysis the models of computation used are deterministic and non-deterministic Turing machines which have a read-only input tape, a fixed number of read/write work tapes and a write-only output tape where the head can never move left. The *time* of a computation is its length. The *space* of a computation is the number of cells visited on the *work tapes*. Precise definitions of these notions and the associated time and space complexity classes can be found in the standard literature [Joh90, HU79, GJ79].

Algorithm DC-EVAL(ϕ, δ)
Inputs:

- A DCL formula ϕ.
- A d-configuration $\delta = (\overline{e}, \overline{v})$.

Output: 1 iff $\mathbf{D} \models \phi$. Otherwise 0.

Method:

1. If ϕ is an atomic formula $x_i = a$ (where x_i is a variable and $a \in Dom$) then return 1 if $v_i = a$. Otherwise return 0.
2. If ϕ is an atomic formula $x_i \neq a$ (where x_i is a variable and $a \in Dom$) then return 1 if $v_i \neq a$. Otherwise return 0.
3. If ϕ is an atomic formula $x_i = x_j$ (where x_i and x_j are variables) then return 1 if $i \; \overline{e} \; j$. Otherwise return 0.
4. If ϕ is an atomic formula $x_i \neq x_j$ (where x_i and x_j are variables) then return 1 if $\neg(i \; \overline{e} \; j)$. Otherwise return 0.
5. If ϕ is $\phi_1 \vee \phi_2$ and DC-EVAL(ϕ_1, δ) returns 1 then return 1, else return the result of DC-EVAL(ϕ_2, δ).
6. If ϕ is $\neg\phi'$ and the result of DC-EVAL(ϕ', δ) is 1 then return 0, else return 1.
7. If ϕ is $(\exists x)\phi'$ then for every extension δ' of δ call DC-EVAL(ϕ', δ'). If the result of one of these calls is 1 then return 1 else return 0.

Figure 6.1: The algorithm DC-EVAL

If $i > 1$ then $\log i$ will denote $\lceil \log_2 i \rceil$. By convention we assume that $\log 1 = 1$.

Formulas will be represented by the standard binary encoding. Integers are written in binary. Rationals are written as fractions of two integers. Variables are chosen from v_0, v_1, v_{10} etc. (i.e., subscripts are written in binary). Similarly, the elements of Dom are d_0, d_1, d_{10} etc. Finally, the size of all integer constants and elements of Dom in a formula ϕ is assumed to be less than $\log|\phi|$.

The following theorem characterizes the complexity of the decision problem for DC. The theorem is implicit in [KKR90] who consider only *data complexity.*

Theorem 6.3.1 *Let ϕ be a DCL sentence. The problem of deciding whether $\mathbf{D} \models \phi$ is PSPACE-complete.*

Proof: *Lower bound.* PSPACE-hardness follows from a straightforward reduction from QBF [Sto77].

Upper bound. The recursive algorithm DC-EVAL of figure 6.1 can be used to decide whether $\mathbf{D} \models \phi$. DC-EVAL can be implemented by a deterministic Turing machine which uses a stack for storing the activation record of each recursive call. The activation record of each call will store the value of δ. For every d-configuration $\delta = (\overline{e}, \overline{v})$ created by the algorithm, $\overline{e}$ can be stored as a sequence of equivalence classes using $O(K \log K)$ bits where K is the number of quantifiers of ϕ.[1] Similarly $\overline{v}$ can be stored using $K \log|\phi|$ bits. Thus δ needs at most $O(K \log K \log|\phi|)$ bits. The depth of the recursion is at most $|\phi|$. The space needed for keeping track of returned values is constant. Therefore the total space requirements of this algorithm is $O(K\,|\phi| \log|\phi| \log K)$. ■

Let us now consider sentences with a fixed number of alternations of quantifiers.

Definition 6.3.4 Let $\mathcal{L}$ be a first-order language and k be a fixed natural number. A $\exists_k$ (resp. $\forall_k$) formula of $\mathcal{L}$ is a formula in prenex normal form with k alternations of quantifiers beginning with an existential (resp. universal) quantifier.

According to the above definition, a $\exists_k$ sentence of DCL has the form

$$(\exists \overline{x_1})(\forall \overline{x_2}) \cdots (Q_k \overline{x_k}) \psi(\overline{x_1}, \ldots, \overline{x_k}) \tag{6.1}$$

where ψ is quantifier-free. The quantifier Q_k is existential if k is odd and universal if k is even.

Theorem 6.3.2 *Let ϕ be a $\exists_k$ sentence of DCL. The problem of deciding whether $\mathbf{D} \models \phi$ is Σ_k^p-complete.*

Proof: *Lower bound.* Σ_k^p-hardness follows from a straightforward reduction from $\mathrm{QBF}_k^{\exists}$ (the corresponding problem for quantified boolean formulas with a fixed number of alternations of quantifiers) [Sto77].

Upper bound. Let ϕ be the sentence (6.1) above. Then $\mathbf{D} \models \phi$ if and only if

[there exists a d-configuration δ_1 of size $|\overline{x_1}|$]

...

[for all/there exists a d-configuration δ_k of size $|\overline{x_1}| + \cdots + |\overline{x_k}|$]
δ_{i+1} is an extension of δ_i for all $i = 1, \ldots, k-1$, and ψ is true when evaluating $\overline{x_1}, \ldots, \overline{x_k}$ according to δ_k.

The last condition can be evaluated in time polynomial in $|\phi|$. The sizes of $\delta_1, \ldots, \delta_k$ are also polynomial in $|\phi|$. Therefore the result follows.

[1] We need at most $K \log K$ bits to store all elements of the set $\{1, \ldots, K\}$ plus $K+1$ bits for separating elements and equivalence classes.

■

A corresponding Π_k^p bound can be established for $\forall_k$ sentences of DCL.

Let us now consider quantifier elimination for open formulas of DCL. Let us assume we have a formula $\phi(\bar{t})$ of DCL such that $\bar{t}$ is the vector of all the free variables of ϕ. A quantifier free formula ϕ' equivalent to ϕ can be found in the following way. First, we partition the space $Dom^{|\bar{t}|}$ into *regions* with the following properties:

1. Every region can be represented by a conjunction of DCL-constraints.

2. The truth value of the sentence $\phi(\bar{\tau})$ is the same for all points $\bar{\tau}$ in the same region.

For every region with the above properties, we use algorithm DC-EVAL to determine whether ϕ evaluates to true for all points in the region. If this check succeeds, the conjunction of constraints representing the region becomes a disjunct in the DNF form of ϕ'.

The following theorem of [KKR90] summarizes the above discussion.

Theorem 6.3.3 *Let ϕ be a DCL formula. A quantifier-free formula equivalent to ϕ in DNF can be computed in PSPACE.*

6.4 Quantifier Elimination For the Theory diPC Revisited

First, we present a decision procedure for sentences of diPCL. Then we use this procedure for developing a quantifier elimination algorithm for arbitrary diPCL formulas.

Our starting point will be the quantifier elimination technique from [FR75]. The main point of this technique is that "given a particular theory, one gives an elimination of quantifiers procedure, analyzes it to see how *large* constants can grow, and then uses this analysis (...) to limit quantifiers to range over finite sets instead of an infinite domain" [FR75].

We start with a definition from [FR75].

Definition 6.4.1 Let i be an integer. We will say that i is limited by the positive integer L, denoted by $i \preceq L$, iff $|i| \leq L$.

As usual, we will also write $i \succeq j$ when $j \preceq i$. The following definitions introduce additional notation.

Definition 6.4.2 Let ϕ be diPCL formula. Then $(\forall z \preceq L)\phi$ is a shorthand for the formula

$$(\forall z)(z \geq -L \ \wedge\ z \leq L \ \supset\ \phi).$$

Similarly $(\exists z \preceq L)\phi$ is a shorthand for the formula

$$(\exists z)(z \geq -L \ \wedge\ z \leq L \ \wedge\ \phi)$$

Definition 6.4.3 Let ϕ be a diPCL formula. The expression $maxabs(\phi)$ will denote the maximum absolute value of the integers which appear in ϕ.

The following lemmas are at the heart of the elimination of quantifiers.

Lemma 6.4.1 *Let us assume that $\bar{t}$ is a vector of variables, y is a variable, $\phi(\bar{t}, y)$ is a diPCL formula with K quantifiers and $\bar{\tau}$ is a vector of elements of $\mathbb{Z}$ limited by W. Then*

$$\mathbf{Z} \models (Qy)\phi(\bar{\tau}, y) \ \text{ iff }\ \mathbf{Z} \models (Qy \preceq 2^K(maxabs(\phi) + 1) + W + 2)\phi(\bar{\tau}, y).$$

If $\bar{t} = ()$, the lemma holds with $W = 0$.

Proof: Let us eliminate quantifiers from ϕ using the procedure of proposition 6.2.2. After the first quantifier is eliminated, the integer constants in the resulting formula are limited by $2maxabs(\phi)+1$. With a simple inductive argument, we can show that after all K quantifiers are eliminated the integer constants in the resulting formula $\phi'(\overline{t}, y)$ are limited by $2^K maxabs(\phi) + 2^K - 1$ or (more compactly) by $2^K(maxabs(\phi) + 1)$.

Now let us consider the formula $\phi'(\overline{\tau}, y)$. Every point constraint in this formula can be written in the form $y \sim c$ where $\sim$ is $<, =$ or $>$ and c is limited by $2^K(maxabs(\phi)+1)+W+1$. The truth value of $\phi'(\overline{\tau}, y)$ is the same for all y such that $y \succeq 2^K(maxabs(\phi) + 1) + W + 2$. Therefore the truth value of $(Qy)\phi'(\overline{\alpha}, \overline{\tau}, y)$ can be determined by simply determining the truth value of $\phi'(\overline{\tau}, y)$ for every integer limited by $2^K(maxabs(\phi) + 1) + W + 2$. ■

Lemma 6.4.2 *Let us assume that ϕ is a diPCL sentence in PNF. Let $(Q_1 t_1), \ldots, (Q_K t_K)$ be all the quantifiers of ϕ in left to right order of appearance. Then*

$$\mathbf{Z} \models \phi \ \text{ iff } \ \mathbf{Z} \models \phi'$$

where ϕ' is the same as ϕ except that $(Q_i t_i)$ is substituted by

$$(Q_i t_i \preceq 2^{K+i-2}(maxabs(\phi) + 1) + i + 1)$$

for all $i = 1, \ldots, K$.

Proof: The sentence ϕ can be written as $(Q_1 t_1)\psi_1(t_1)$ where ψ_1 is a formula with $K-1$ quantifiers. Thus lemma 6.4.1 gives us

$$\mathbf{Z} \models (Q_1 t_1)\psi_1(t_1) \ \text{ iff } \ \mathbf{Z} \models (Q_1 t_1 \preceq 2^{K-1}(maxabs(\phi) + 1) + 2)\psi_1(t_1).$$

A similar inductive argument gives the result. ■

Figure 6.2 presents the decision procedure diPC-Eval for the theory diPC. This procedure makes use of lemma 6.4.2 to restrict quantifiers to range over finite sets of integers. If we have a sentence ϕ of diPC then $\mathbf{Z} \models \phi$ iff diPC-Eval$(\phi, Val, 0)$ returns 1 (see figure 6.2 for more details).

The following theorem gives the exact complexity of the decision problem for theory diPC.

Theorem 6.4.1 *Let ϕ be a diPCL sentence. The problem of deciding whether $\mathbf{Z} \models \phi$ is PSPACE-complete.*

Proof: *Lower bound.* PSPACE-hardness follows from a straightfoward reduction from QBF [Sto77].

Upper bound. We first transform ϕ into a formula ϕ' in PNF. The length of the resulting formula is $|\phi'| \leq |\phi| \log |\phi|$. Then the recursive algorithm diPC-Eval of figure 6.2 can be used to decide whether $\mathbf{Z} \models \phi'$. The correctness of diPC-Eval follows from lemma 6.4.2.

diPC-Eval can be implemented by a deterministic Turing machine which uses a stack for storing the activation record of each recursive call. The activation record of each call will store the values of Val and I. Array Val needs $O(K(2K - 1 + \log(maxabs(\phi) + 1)))$ bits where K is the number of quantifiers of ϕ. Counter I needs at most $\log K$ bits. Thus the total for I and Val is $O(K^2 \log(maxabs(\phi))$ bits. The depth of the recursion is at most $|\phi|$. The space needed for keeping track of returned values is constant. Therefore the total space requirements of this algorithm is $O(K^2\, |\phi| \log(maxabs(\phi)))$. ■

It might be interesting to compare the above result with the following well-known theorems.

Algorithm $\text{DIPC-EVAL}(\phi, Val, I)$

Inputs:

- A diPCL formula ϕ in PNF with I free variables.
- An array Val of $K \geq I$ elements. The first I elements of Val contain the current values of the free variables of ϕ. The rest contain the value "undefined". When the procedure is called for the first time, all elements of Val contain the value "undefined".

Output: 1 iff $\mathbf{Z} \models \phi$. Otherwise 0.

Method:

1. If ϕ is an atomic formula $x_j - x_i < c_0$ (where x_j, x_i are variables and c_0 is an integer constant) then return 1 if $Val(j) - Val(i) < c_0$. Otherwise return 0.
2. If ϕ is an atomic formula $x_j - x_i = c_0$ (where x_j, x_i are variables and c_0 is an integer constant) then return 1 if $Val(j) - Val(i) = c_0$. Otherwise return 0.
3. If ϕ is an atomic formula $x_j - x_i \leq c_0$ (where x_j, x_i are variables and c_0 is an integer constant) then return 1 if $Val(j) - Val(i) \leq c_0$. Otherwise return 0.
4. If ϕ is $\phi_1 \lor \phi_2$ and $\text{DIPC-EVAL}(\phi_1, Val, I)$ returns 1 then return 1, else return the result of $\text{DIPC-EVAL}(\phi_2, Val, I)$.
5. If ϕ is $\neg\phi'$ and the result of $\text{DIPC-EVAL}(\phi', Val, I)$ is 1 then return 0, else return 1.
6. If ϕ is $(\exists t)\phi'$ then for every integer $i \preceq 2^{K+I-1}(maxabs(\phi)+1)+I+2$, set $Val(I+1)$ equal to i and call $\text{DIPC-EVAL}(\phi', Val, I+1)$. If the result of one of these calls is 1 then return 1 else return 0.

Figure 6.2: The algorithm DIPC-EVAL

Theorem 6.4.2 *[FR75] There exists a constant $c > 0$ such that a deterministic Turing machine can decide a sentence of length n in the first order theory of integer addition with order (Presburger arithmetic) in $O(2^{2^{cn}})$ space.*

Theorem 6.4.3 *[FR74] There exists a constant $d > 0$ such that any non-deterministic Turing machine which decides the first order theory of integer addition with order requires for almost every n time $2^{2^{dn}}$ to decide some sentences of length n.*

Theorem 6.4.4 *Let ϕ be a $\exists_k$ sentence of diPCL. The problem of deciding whether $\mathbf{Z} \models \phi$ is Σ_k^p-complete.*

Proof: *Lower bound.* Σ_k^p-hardness follows from a straightforward reduction from $\mathrm{QBF}_k^{\exists}$ [Sto77].

Upper bound. Let ϕ be the sentence

$$(\exists \overline{t_1})(\forall \overline{t_2}) \cdots (Q_k \overline{t_k}) \psi(\overline{t_1}, \ldots, \overline{t_k})$$

where ψ is quantifier-free. Then $\mathbf{Z} \models \phi$ if and only if

[there exists a vector $\overline{\tau_1}$ of integers limited by $2^{K+|\overline{t_1}|-2}(maxabs(\phi)+1) + |\overline{t_1}| + 1$]

...

[for all/there exists a vector τ_k of integers limited by
$2^{K+(|\overline{t_1}|+\cdots+|\overline{t_k}|)-2}(maxabs(\phi)+1) + (|\overline{t_1}| + \cdots + |\overline{t_k}|) + 1$]

ψ is true when evaluating $\overline{t_1}, \ldots, \overline{t_k}$ according to $\overline{\tau_1}, \ldots, \overline{\tau_k}$.

The last condition can be evaluated in time polynomial in $|\phi|$. The sizes of $\overline{\tau_1}, \ldots, \overline{\tau_k}$ are also polynomial in $|\phi|$. Therefore the result follows.

■

A corresponding Π_k^p bound can be established for $\forall_k$ sentences of diPCL.

It might be interesting to compare the above result with the following result of [RL78].

Theorem 6.4.5 *There exist constants $d, e > 0$ such that a deterministic Turing machine can decide a sentence with length $n > 4$ and at most K alternations of quantifiers, in the first order theory of integer addition with order, in $O(2^{dn^{K+4}})$ space and $O(2^{2^{en^{K+4}}})$ time.*

6.4.1 Open Formulas

We will now present a quantifier elimination algorithm for open formulas of diPCL. Let us assume we have a formula $\phi(\overline{t})$ of diPCL such that $\overline{t}$ is the vector of all the free variables of ϕ. A quantifier free formula ϕ' equivalent to ϕ can be found in the following way. At first, using the techniques presented above, we estimate how large integer constants can grow in the constraints of a quantifier-free formula equivalent to ϕ. Then we use this information to construct a finite partition of the space $\mathbb{Z}^{|\overline{t}|}$ into *regions* with the following properties:

1. Every region can be represented by a conjunction of diPCL-constraints.

2. The truth value of the sentence $\phi(\overline{\tau})$ is the same for all points $\overline{\tau}$ in the same region.

Therefore we can check whether *all* points in a region satisfy $\phi(\overline{t})$ by picking a *single* point in the region and checking whether $\mathbf{Z} \models \phi(\overline{\tau})$ is true. The latter check can be done using the above algorithm DIPC-EVAL. Every conjunction of constraints representing a region for which this check succeeds, becomes a disjunct in the DNF of ϕ'.

Let us now introduce the machinery required for presenting our method. We first define a special kind of BICN (BICN were introduced in section 3.3).

Definition 6.4.4 Let Z be a set such that $Z \subseteq \mathbb{Z}$ and $Z \neq \emptyset$. An *integer BICN with bounds from the set* Z is a consistent BICN with all edges labeled with $[c, c], (c, \infty)$ or $(-\infty, c)$ where $c \in Z$.

If an integer BICN has $n+1$ nodes (including the 0-th one which corresponds to variable $x_0 = 0$), we will say that it is of *size* n.

Let us now define the concept of a formula corresponding to an integer BICN N.

Definition 6.4.5 Let N be an integer BICN of size n. The formula with free variables $\overline{x} = (x_1, \ldots, x_n)$ corresponding to N is the diPCL formula $\bigwedge_{\theta \in Constraints(M)} \theta$ where M is the minimal network equivalent to N. This formula will be denoted by $\Phi(N)$.

Notation 6.4.1 If ϕ is a diPCL formula with K quantifiers and vector of free variables $\overline{t}$ then Z_ϕ will denote the set of integers

$$\{i : \; i \in \mathbb{Z} \text{ and } |i| \leq \left|\overline{t}\right| 2^K(maxabs(\phi) + 1)\}.$$

The following lemma tells us how to partition the answer space into regions.

Lemma 6.4.3 *Let $\phi(\overline{t})$ be a diPCL formula. Then there is a quantifier-free formula $\phi'(\overline{t})$ equivalent to ϕ with the following properties:*

1. *ϕ' is in DNF*
2. *Every disjunct of ϕ' is the diPCL formula corresponding to an integer BICN N of size $\left|\overline{t}\right|$ with bounds from the set Z_ϕ.*

Proof: Let $\phi_1(\overline{t})$ be the quantifier-free formula

$$\psi_1(\overline{t}) \;\vee\; \cdots \;\vee\; \psi_m(\overline{t})$$

equivalent to ϕ obtained by the algorithm of proposition 6.2.2. Let us also assume that the only relation symbol appearing in ψ_1 is $\leq$. As in the proof of lemma 6.4.1, integer constants in ϕ_1 are limited by $2^K(maxabs(\phi) + 1)$.

Let $N_1, \ldots, N_{m_1}$, $(m_1 \leq m)$ be the minimal BICN corresponding to the consistent ψ_i's $(i = 1, \ldots, m)$. Let us create the formula

$$\phi_2(\overline{t}) \equiv \psi'_1(\overline{t}) \;\vee\; \cdots \;\vee\; \psi'_{m_1}(\overline{t})$$

where

$$\psi'_i = \bigwedge_{\eta \in Constraints(N_i)} \eta, \quad \text{for all } i = 1, \ldots, m_1.$$

Every integer r, such that $t_k - t_l \leq r$ (or $t_k - t_l = r$) is a conjunct in some ψ'_i, is the length of the shortest (*simple*) path connecting nodes k and l in the minimal distance graph for N_i [DMP91]. The number of edges in a simple path is $\left|\overline{t}\right|$ and the label of each edge is limited by $2^K(maxabs(\phi) + 1)$. Therefore every integer in ϕ_2 is limited by $\left|\overline{t}\right| 2^K(maxabs(\phi) + 1)$.

We can now obtain a formula equivalent to ϕ_2 in the desired form as follows:

1. From every ψ'_i construct a conjuction of disjunctions ψ''_i in the following way.
   (a) For every pair of variables t_k, t_l such that $t_k - t_l = r_1$ is in ψ'_i, add $t_k - t_l = r_1$ to ψ''_i.

(b) For every pair of variables t_k, t_l such that $t_k - t_l \leq r_1$ and $t_k - t_l \geq r_2$ is in ψ_i', add

$$\bigvee_{r_1 \leq r \leq r_2} t_k - t_l = r$$

to ψ_i''.

(c) For every pair of variables such that only $t_k - t_l \leq r_1$ is in ψ_i', add conjunct

$$t_k - t_l < -\Lambda \ \vee \bigvee_{-\Lambda \leq r \leq r_1} t_k - t_l = r$$

to ψ_i'' where $\Lambda = \left|\bar{t}\right| 2^K (maxabs(\phi) + 1)$.

(d) For every pair of variables such that only $t_k - t_l \geq r_2$ is in ψ_i', add conjunct

$$\bigvee_{r_2 \leq r \leq \Lambda} t_k - t_l = r \ \vee \ t_k - t_l > \Lambda$$

to ψ_i'' where Λ is as above.

(e) For every pair of variables t_k, t_l such that ψ_i' contains no constraint involving t_k and t_l, add conjunct

$$t_k - t_l < -\Lambda \ \vee \bigvee_{r_1 \leq r \leq r_2} t_k - t_l = r \ \vee \ t_k - t_l > \Lambda$$

to ψ_i'' where Λ is as above.

2. Transform every $\psi_i''(\bar{t})$ in DNF. Let $\theta_1(\bar{t}) \ \vee \ \cdots \ \vee \ \theta_n(\bar{t})$ be the resulting formula.

3. Let $M_1, \ldots, M_{n_1}$, $(n_1 \leq n)$ be the minimal BICN corresponding to consistent θ_i's $(i = 1, \ldots, n)$. The wanted formula $\phi'(\bar{t})$ is

$$\theta_1'(\bar{t}) \ \vee \ \cdots \ \vee \ \theta_{n_1}'(\bar{t})$$

where

$$\theta_i' = \bigwedge_{\eta \in Constraints(M_i)} \eta, \quad \text{for all } i = 1, \ldots, n_1.$$

■

The following lemmas give some important properties of integer BICN.

Lemma 6.4.4 *Let $\overline{\tau}$ be an element of $\mathbb{Z}^n$. For any set $Z \subseteq \mathbb{Z}$, there exists a unique integer BICN N with bounds from Z such that $\mathbf{Z} \models \Phi(N)(\overline{\tau})$.*

Proof: Existence is obvious. For the uniqueness part, we simply observe that $N_1 \neq N_2$ implies that $\Phi(N_1) \ \wedge \ \Phi(N_2)$ is inconsistent. ■

The following lemma allows us to verify the truth of a diPCL formula over a region of the answer space by verifying its truth over a *point* in this region.

Lemma 6.4.5 *Let $\phi(\bar{t})$ be a diPCL formula and N be an integer BICN with bounds from the set Z_ϕ. Then $\mathbf{Z} \models \Phi(N) \supset \phi$ iff $\mathbf{Z} \models \phi(\overline{\tau})$ is true for an arbitrary $\overline{\tau}$ such that $\mathbf{Z} \models \Phi(N)(\overline{\tau})$.*

Proof: The "only if" part is trivial so we consider the "if" part. Let us assume that there is $\overline{\tau} \in \mathbb{Z}^{|\overline{t}|}$ such that $\mathbf{Z} \models \Phi(N)(\overline{\tau})$ and $\mathbf{Z} \models \psi(\overline{\tau})$. Let

$$\phi'(\overline{t}) \equiv \theta_1'(\overline{t})) \ \vee \ \cdots \ \vee \ \theta_n'(\overline{t})$$

be the diPCL formula equivalent to ϕ computed as in lemma 6.4.3. Then there exists a single disjunct $\theta_i(\overline{t})$ of ϕ' such that $\mathbf{Z} \models \theta_i(\overline{\tau})$. But then θ_i must be $\Phi(N)$ from lemma 6.4.4. Therefore $\mathbf{Z} \models \forall(\Phi(N) \ \supset \ \phi)$ since $\Phi(N)$ is a disjunct of ϕ'. ■

Now we are ready to demonstrate the main result of this section.

Theorem 6.4.6 *Let ϕ be a diPCL formula. A quantifier-free formula equivalent to ϕ in DNF can be computed in PSPACE.*

Proof: Let us assume that ϕ has K quantifiers and $\overline{t}$ is the vector of all its free variables. We can compute a quantifier-free formula equivalent to ϕ in DNF as follows. We generate one by one all integer BICN of size $|\overline{t}|$ with bounds from Z_ϕ. For every such BICN N, we find a solution $\overline{\tau}$ of N and check whether $\mathbf{Z} \models \phi(\overline{\tau})$ using the above algorithm diPC-Eval. If this check succeeds, the diPCL formula corresponding to N becomes a disjunct of the resulting quantifier-free formula. The correctness of this procedure follows from lemmas 6.4.3 and 6.4.5.

Finding a solution of a BICN can be done in the following way. We first compute the minimal network M equivalent to N. Then we find a solution of M using *backtrack-free* search as follows [DMP91]. We initially assign the value 0 to t_0. Then we assign to t_1 any value which satisfies the constraints involving t_1 and t_0. We proceed in the same fashion with t_2, t_3 and so on.

Let us now show that the above procedure can be implemented by a deterministic Turing machine in PSPACE.[2] The integer bounds in M will be limited by $|\overline{t}|^2\, 2^K(maxabs(\phi)+1)$. Therefore the space for storing a BICN and its minimal network is $O(K\,|\overline{t}|^2 \log|\overline{t}| \log(maxabs(\phi)))$. A solution $\overline{\tau}$ of M can be chosen so that τ_i is an integer limited by $|\overline{t}|^3\, 2^K(maxabs(\phi)+1) + |\overline{t}|^2$. Every τ_i can be stored in $O(\log|\overline{t}| + K + \log(maxabs(\phi)+1)+1)$ bits. Thus we can store a solution in $O(K\,|\overline{t}| \log|\overline{t}| \log(maxabs(\phi)))$ bits.

When we call diPC-Eval with argument $\phi(\tau)$ the space used is $O(K^3\,|\phi| \log|\overline{t}| \log(maxabs(\phi)))$ bits. This follows from theorem 6.4.1 and the fact

$$maxabs(\phi(\tau)) \leq 2^{K+2}\,|t|^3\, maxabs(\phi).$$

Therefore the procedure has space complexity $O(K^3\,|\overline{t}|^2 \log|\overline{t}|\,|\phi| \log(maxabs(\phi)))$.

■

We can now combine the machinery of section 3.2.3 and the above techniques to achieve efficient quantifier elimination for the theory diTC. The following results are straightforward.

Theorem 6.4.7 *Let ϕ be a diTCL sentence. The problem of deciding whether $\mathbf{ZI_Z} \models \phi$ is PSPACE-complete.*

Theorem 6.4.8 *Let ϕ be $\exists_k$ sentence of diTCL. The problem of deciding whether $\mathbf{ZI_Z} \models \phi$ is Σ_k^p-complete.*

Theorem 6.4.9 *Let ϕ be a diTCL formula. A quantifier-free formula equivalent to ϕ in DNF can be computed in PSPACE.*

[2] The size of the output formula can be exponential in K and $|t|$. But the number of cells visited on the output tape does not count towards the space complexity of a Turing machine (remark 6.3.2).

6.5 Quantifier Elimination For the Theory dePC Revisited

As in the previous section, we will first develop a decision procedure for sentences of dePCL. Then we will use this procedure for developing a quantifier elimination algorithm for arbitrary dePCL formulas.

At first we show that we can confine our attention to formulas of dePCL involving *only integer* constants. A similar result appears in [Alu91] in the context of checking emptiness of the language of a timed automaton.

Notation 6.5.1 Let ϕ be a formula of dePCL and $r \in \mathbb{Q}$. Then ϕ_r will denote the formula which is obtained from ϕ by replacing each rational constant c by $r \cdot c$.

Lemma 6.5.1 *Let ϕ be a formula of dePCL with free variables $\overline{t}$. If $r \in \mathbb{Q}$, $r > 0$ and $\overline{\tau} \in \mathbb{Q}^n$ then*

$$\mathbf{Q} \models \phi[\overline{t} \leftarrow \overline{\tau}] \text{ iff } \mathbf{Q} \models \phi_r[\overline{t} \leftarrow r \cdot \overline{\tau}].$$

Proof: Use induction on the structure of ϕ. ■

The following definition will be used heavily in the subsequent discussion.

Definition 6.5.1 A quantifier Qz is limited by the fraction A/B, denoted by $Qz \preceq A/B$, if, instead of ranging over all rational numbers, it ranges over all numbers whose numerator is limited by A and whose denominator *is* B.

Lemma 6.5.2 *Let ϕ be the following dePCL sentence in PNF*

$$(Q_1 t_1) \cdots (Q_K t_K) \psi(t_1, \ldots, t_K).$$

Let us also assume that only integer constants are involved in ϕ. Then $\mathbf{Q} \models \phi$ iff $\mathbf{Q} \models \phi'$ where ϕ' is the same as ϕ except that $(Q_i t_i)$ is substituted by

$$(Q_i \preceq \frac{2^{K+3i-2}(maxabs(\phi)+1)}{2^i})$$

for all $i = 1, \ldots, K$.

Proof: We will use induction on the order of appearance of the quantifier in ϕ.

Base case, $i = 1$. Let us eliminate quantifiers $(Q_2 t_2), \ldots, (Q_K t_K)$ from ϕ using the procedure of proposition 6.2.2. The integer constants in the resulting formula $(Q_1 t_1)\phi'(t_1)$ are limited by $2^{K-1}(maxabs(\phi)+1)$ (this can be shown as in lemma 6.4.1). These integers partition $\mathbb{Q}$ into a set S of $4 \cdot 2^{K-1}(maxabs(\phi)+1)+3$ intervals of the form $(-\infty, r)$ or $[r, r]$ or $(r, r+1)$ or (r, ∞) where $r \preceq 2^{K-1}(maxabs(\phi)+1)$. The truth value of $\phi'(t_1)$ remains the same for all t_1 in the same interval of S. Therefore we can determine the truth value of $(Qt_1)\phi'(t_1)$ in $\mathbf{Q}$ by determining the truth value of $\phi'(t_1)$ while t_1 ranges over a finite set of representatives, one for each interval of S. The set

$$S_{rep} = \{\ (t_1+t_2)/2,\ (t_1+t_2)/2+1 :\ t_1, t_2 \in \mathbb{Z} \text{ and } t_1, t_2 \preceq 2^{K-1}(maxabs(\phi)+1)\ \}$$

is an appropriate set of such representatives. The elements of S_{rep} are rationals limited by $2^{K+1}(maxabs(\phi)+1)/2$.

Inductive step. Let us now assume that the lemma holds for quantifiers $(Q_1t_1), \ldots, (Q_it_i)$. We will show that the lemma holds for $(Q_{i+1}t_{i+1})$ as well. From the induction hypothesis we have $\mathbf{Q} \models \phi$ if and only if $\mathbf{Q} \models \phi_1$ where ϕ_1 is

$$(Q_1t_1 \preceq \frac{2^{K+1}(maxabs(\phi)+1)}{2}) \cdots (Q_it_i \preceq \frac{2^{K+3i-2}(maxabs(\phi)+1)}{2^i})(Q_{i+1}t_{i+1}) \cdots (Q_Kt_K)\psi(t_1, \ldots, t_K).$$

Let us eliminate quantifiers $(Q_{i+2}t_{i+2}), \cdots, (Q_Kt_K)$ from ϕ_1 using the procedure of proposition 6.2.2 to arrive at the following formula ϕ_2:

$$(Q_1t_1 \preceq \frac{2^{K+1}(maxabs(\phi)+1)}{2}) \cdots (Q_it_i \preceq \frac{2^{K+3i-2}(maxabs(\phi)+1)}{2^i})(Q_{i+1}t_{i+1})\psi_1(t_1, \ldots, t_{i+1}).$$

The truth value of ϕ_2 depends on the truth values of all formulas

$$(Q_{i+1}t_{i+1})\psi_1(\tau_1, \ldots, \tau_i, t_{i+1})$$

where

$$\tau_j \leq \frac{2^{K+3j-2}(maxabs(\phi)+1)}{2^j}, \quad j = 1, \ldots, i.$$

The atomic formulas of $(Q_{i+1}t_{i+1})\psi_1(\tau_1, \ldots, \tau_i, t_{i+1})$ are of the form $t_{i+1} \sim r$ or $t_{i+1} \sim \tau_j + r$ where $1 \leq j \leq i$, $\sim$ is $<, \leq$ or $=$, τ_j is a rational limited as above and r is an integer limited by $2^{K-(i+1)}(maxabs(\phi)+1)$.

Let us now observe that every rational limited by $2^{K+3j-2}(maxabs(\phi)+1)/2^j$, where $1 \leq j \leq i-1$, is included in the set of rationals limited by $2^{K+3i-2}(maxabs(\phi)+1)/2^i$. Therefore $\tau_j + r$ is a rational limited by $2^{K+3i-1}(maxabs(\phi)+1)/2^i$.

Using an argument similar to the one given for the base case, we can see that the quantifier $(Q_{i+1}t_{i+1})$ can be limited to range over rationals that are the average of two rationals limited by $2^{K+3i-1}(maxabs(\phi)+1)/2^i$, or are one smaller or one larger than all such averages. As a result, $(Q_{i+1}t_{i+1})$ can be limited by $2^{K+3(i+1)-2}(maxabs(\phi)+1)/2^{i+1}$. The result follows. ■

Definition 6.5.2 If ϕ is a dePCL formula then $maxabs(\phi)$ will denote the maximum absolute value of the integers which appear in ϕ as numerators or denominators. If ϕ involves only integer constants then $maxabs(\phi)$ will denote the maximum absolute value of the integers which appear in ϕ.

We are now ready to prove the basic result of this section.

Theorem 6.5.1 *Let ϕ be a dePCL sentence. The problem of deciding whether $\mathbf{Q} \models \phi$ is PSPACE-complete.*

Proof: *Lower bound.* PSPACE-hardness follows from a straightfoward reduction from QBF [Sto77].

Upper bound. An algorithm for this problem can proceed as follows. First, we transform ϕ into formula ϕ' which involves only integer constants. This can be done by multiplying every fraction p/q of ϕ by the product of all denominators of fractions in ϕ. Therefore every integer in ϕ' will be limited by $maxabs(\phi)^{|\phi|+1}$. We can conclude that

$$|\phi'| \leq |\phi|\,(|\phi|+1)\log(maxabs(\phi)) \leq 2\,|\phi|^2\log(maxabs(\phi))$$

and

$$maxabs(\phi') \leq maxabs(\phi)^{|\phi|+1}.$$

The next step is to transform ϕ' into a formula ϕ'' in PNF. Then we can devise an algorithm DEPC-EVAL which decides ϕ''. This algorithm will make use of lemma 6.5.2 to limit quantifiers over finite sets of rationals. DEPC-EVAL is similar to DIPC-EVAL so its details can be omitted.

The length of ϕ'' will be

$$|\phi''| \leq 8\,|\phi|^2 \log|\phi| \log(maxabs(\phi)).$$

The maximum numerator of a fraction stored in Val is $2^{4K-2}(maxabs(\phi'')+1)$; the maximum denominator is 2^K. Thus array Val needs $O(K^2\log(maxabs(\phi'')))$ space. The counter I needs $\log K$ bits. The depth of the recursion in DEPC-EVAL is $|\phi|$. Thus the total space needed is

$$O(K^2\,|\phi|^2 \log|\phi| \log(maxabs(\phi))).$$

■

It might be interesting to compare the above result with the following theorems. The first one considers a theory which is less expressive than dePC. The other two deal with the full first-order theory of real addition with order.

Theorem 6.5.2 *[FG77] There exists a constant $c > 0$ such that a deterministic Turing machine can decide a sentence of length n and quantifier depth K in the first order theory of rational order in space $O(c(K\log(K+n)))$.*

Theorem 6.5.3 *[FR75] There exists a constant $c > 0$ such that a deterministic Turing machine can decide a sentence of length n in the first order theory of real addition with order in $O(2^{cn})$ space.*

Theorem 6.5.4 *[FR74] There exists a constant $d > 0$ such that any non-deterministic Turing machine which decides the first order theory of real addition (even without order) requires for almost every n time 2^{dn} to decide some sentences of length n.*

The next theorem follows easily from the above discussion. It is also a consequence of the fact that deciding a formula of the same form in the first-order theory of real addition with order is also Σ_k^p-complete [Son85].

Theorem 6.5.5 *Let ϕ be $\exists_k$ sentence of dePCL. The problem of deciding whether $\mathbf{Q} \models \phi$ is Σ_k^p-complete.*

A corresponding Π_k^p bound can be established for $\forall_k$ sentences of dePCL.

6.5.1 Open Formulas

We will now present a quantifier elimination algorithm for open formulas of dePCL. We will proceed as in the case of diPCL formulas. The following lemma will allow us to concentrate on dePCL formulas involving only integer constants.

Lemma 6.5.3 *Let ϕ be a formula of dePCL and ϕ' be a quantifier-free formula equivalent to ϕ. If $r \in \mathbb{Q}$ and $r > 0$ then $\phi' \equiv \psi_{r^{-1}}$ where ψ is a quantifier-free formula equivalent to ϕ_r.*

Proof: Let $\overline{x}$ be the vector of variables in ϕ'. If $\overline{\chi} \in \mathbb{Q}^{|\overline{x}|}$ then the following equivalences prove the lemma (with help from lemma 6.5.1):

$\mathbf{Q} \models \phi'[\overline{x} \leftarrow \overline{\chi}]$ iff $\mathbf{Q} \models \phi[\overline{x} \leftarrow \overline{\chi}]$ iff $\mathbf{Q} \models \phi_r[\overline{x} \leftarrow r \cdot \overline{\chi}]$ iff $\mathbf{Q} \models \psi[\overline{x} \leftarrow r \cdot \overline{\chi}]$ iff $\mathbf{Q} \models \psi_{r^{-1}}[\overline{x} \leftarrow \overline{\chi}]$.

■

Now assume we are given a formula ϕ of dePCL. We can find a quantifier-free formula equivalent to ϕ as follows. First we transform ϕ into a formula ϕ_r which has only integer constants by multiplying every fraction by an appropriate integer r. Then we find a quantifier-free formula ψ equivalent to ϕ_r. Finally, we compute $\psi_{r^{-1}}$ which is a quantifier-free formula equivalent to ϕ.

Let us then assume that $\phi(\overline{t})$ is a formula of dePCL which involves only integer constants, and $\overline{t}$ are all the free variables of ϕ. A quantifier free formula ϕ' equivalent to ϕ can be found in the following way. At first, we estimate how large integer constants can grow in the constraints of the answer. Then we use this information to construct a finite partition of the space $\mathbb{Q}^{|\overline{t}|}$ into *regions* with the following properties:

1. Every region can be represented by a conjunction of dePCL-constraints involving only integer constants.
2. The truth value of the sentence $\phi(\overline{\tau})$ is the same for all points $\overline{\tau}$ in the same region.

Therefore we can check whether *all* points in a region satisfy $\phi(\overline{t})$ by picking a *single* point $\overline{\tau}$ in the region and checking whether $\mathbf{Q} \models \phi(\overline{\tau})$ is true. The latter check can be done using the above algorithm dePC-Eval. Every conjunction of constraints representing a region for which this check succeeds, becomes a disjunct in the DNF form of ϕ'.

Let us now introduce the machinery required for presenting our method. We first define rational BICN.

Definition 6.5.3 A *rational BICN with bounds from the set* $Z \subseteq \mathbb{Z}$ is a consistent BICN with all edges labeled with $[c,c], (c,d), (c,\infty)$ or $(-\infty,c)$ where $c < d$ and $c, d \in Z$.

The *size* of a rational BICN is defined as for integer BICN.

Let us now define the concept of a formula corresponding to a rational BICN N.

Definition 6.5.4 Let N be a rational BICN of size n. The formula with free variables $\overline{x} = (x_1, \ldots, x_n)$ corresponding to N is the dePCL formula $\bigwedge_{\theta \in Constraints(M)} \theta$ where M is the minimal network equivalent to N. This formula will be denoted by $\Phi(N)$.

Notation 6.5.2 If ϕ is a dePCL formula with K quantifiers and vector of free variables $\overline{t}$ then Z_ϕ will denote the set of integers

$$\{i : \ i \in \mathbb{Z} \text{ and } |i| \leq |\overline{t}|\, 2^K maxabs(\phi)\}.$$

The following lemmas help to establish the main result. The reader can easily notice the similarity with the development of section 6.4.1. Lemma 6.5.4 tells us how to partition the answer space into regions. Subsequent lemmas give properties of rational BICN. The proofs of some of the lemmas are omitted.

Lemma 6.5.4 *Let $\phi(\overline{t})$ be a dePCL formula involving only integer constants. Then there is a quantifier-free formula $\phi'(\overline{t})$ equivalent to ϕ with the following properties:*

1. *ϕ' is in DNF*
2. *Every disjunct of ϕ' is the dePCL formula corresponding to a rational BICN N of size $|\overline{t}|$ with bounds from the set Z_ϕ.*

Proof: First, we transform ϕ into an equivalent quantifier-free formula ϕ_1 using the algorithm of proposition 6.2.2. Integer constants in ϕ_1 will be limited by $2^K maxabs(\phi)$. Then we proceed as in lemma 6.4.3. ■

Lemma 6.5.5 *Let $\overline{\tau}$ be an element of $\mathbb{Q}^n$. For any set $Z \subseteq \mathbb{Z}$, there exists a unique rational BICN N with bounds from Z such that $\mathbf{Q} \models \Phi(N)(\overline{\tau})$.*

Lemma 6.5.6 *Let $\phi(\overline{t})$ be a dePCL formula involving only integer constants. If N is a rational BICN with bounds from the set Z_ϕ then $\mathbf{Q} \models \Phi(N) \supset \phi$ iff $\mathbf{Q} \models \phi(\overline{\tau})$ for an arbitrary $\overline{\tau}$ such that $\mathbf{Q} \models \Phi(N)(\overline{\tau})$.*

The following theorem gives the main result of this section.

Theorem 6.5.6 *Let ϕ be a dePCL formula. A quantifier-free formula equivalent to ϕ in DNF can be computed in PSPACE.*

Proof: Let us assume that ϕ has K quantifiers and $\overline{t}$ is the vector of all its free variables. We can find a quantifier-free formula equivalent to ϕ as follows. First, we transform ϕ into a formula ϕ'' in PNF which involves only integer constants. This can be done as in theorem 6.5.1: we multiply every fraction of ϕ by the product P of all denominators of fractions in ϕ.

Secondly, we generate, one by one, all rational BICN of size $|\overline{t}|$ with bounds from

$$Z_{\phi''} = \{i : \ i \in \mathbb{Z} \text{ and } |i| \leq |\overline{t}|\, 2^K maxabs(\phi)^{|\phi|+1}\}$$

(since $maxabs(\phi'') \leq maxabs(\phi)^{|\phi|+1}$). For each BICN N, we find a solution $\overline{\tau}$ of N (using the minimal network M) and check whether $\mathbf{Q} \models \phi''(\overline{\tau})$ using algorithm DEPC-EVAL of theorem 6.5.1. If this check succeeds then we divide each constant of $\Phi(N)$ by P and use the result to form a disjunct of the returned formula. The correctness of this procedure follows from the previous lemmas.

Let us now show that the above algorithm can be implemented by a deterministic Turing machine in PSPACE. The integer bounds in M will be limited by $|\overline{t}|^2\, 2^K maxabs(\phi)^{|\phi|+1}$. Therefore the space for storing a BICN and its minimal network is

$$O(K\, |\overline{t}|^2 \log |\overline{t}|\, |\phi| \log(maxabs(\phi))).$$

A solution $\overline{\tau}$ of a BICN can be chosen so that every τ_i is a fraction $A/2$ where A is an integer limited by $|\overline{t}|^2\, K 2^{K+1} maxabs(\phi)^{|\phi|+1}$. Every τ_i can be stored in $O(K \log K \log |\overline{t}|\, |\phi| \log(maxabs(\phi)))$ bits. Thus we can store a solution in

$$O(|\overline{t}| \log |\overline{t}|\, K \log K\, |\phi| \log(maxabs(\phi)))$$

bits. The sentence $\phi''(\overline{\tau})$ can be transformed into one with only integer bounds by multiplying all rationals by 2. In the resulting formula $\phi'''(\overline{\tau})$,

$$maxabs(\phi'''(\overline{\tau})) \leq |\overline{t}|^2\, K 2^{K+3} maxabs(\phi)^{|\phi|+1}.$$

Therefore theorem 6.5.1 implies that DEPC-EVAL needs $O(K^3\, |\overline{t}|\, |\phi| \log(maxabs(\phi)))$ space. Finally, taking into account that

$$|\phi'''(\overline{\tau})| \leq 2\, |\phi''| \leq 16\, |\phi|^2 \log |\phi| \log(maxabs(\phi)),$$

we can conclude that the total space is

$$O(K^3 \left|\bar{t}\right| \log \left|\bar{t}\right| |\phi|^2 \log |\phi| \log(maxabs(\phi))).$$

■

We can now combine the machinery of section 3.2.3 and the above techniques to achieve efficient quantifier elimination for the theory deTC. The following results are obvious.

Theorem 6.5.7 *Let ϕ be a deTCL sentence. The problem of deciding whether $\mathbf{QI_Q} \models \phi$ is PSPACE-complete.*

Theorem 6.5.8 *Let ϕ be $\exists_k$ sentence of deTCL. The problem of deciding whether $\mathbf{QI_Q} \models \phi$ is Σ_k^p-complete.*

A corresponding Π_k^p bound can be established for $\forall_k$ sentences of deTCL.

Theorem 6.5.9 *Let ϕ be a deTCL formula. A quantifier-free formula equivalent to ϕ in DNF can be computed in PSPACE.*

6.6 Alternative Quantifier Elimination Algorithms For Theories of Data and Time

The machinery of the previous sections can now be extended to the theories diPDC, dePDC, diTDC and deTDC in the obvious way. For example, we can combine algorithms DC-Eval and diPC-Eval to obtain an algorithm diPDC-Eval for theory diPDC. The following theorems summarize the basic results about theories of data and time. The proofs of the theorems are omitted.

Theorem 6.6.1 *Let $\mathcal{L}$ be diPDCL, dePDCL, diTDCL or deTDCL and ϕ be a sentence of $\mathcal{L}$. The problem of deciding whether $\mathbf{M}_{\mathcal{L}} \models \phi$ is PSPACE-complete.*

Theorem 6.6.2 *Let $\mathcal{L}$ be diPDCL, dePDCL, diTDCL or deTDCL and ϕ be a $\exists_k$ sentence of $\mathcal{L}$. The problem of deciding whether $\mathbf{M}_{\mathcal{L}} \models \phi$ is Σ_k^p-complete.*

A corresponding Π_k^p bound can be established for $\forall_k$ sentences of $\mathcal{L}$.

Theorem 6.6.3 *Let $\mathcal{L}$ be diPDCL, dePDCL, diTDCL or deTDCL and ϕ be a formula of $\mathcal{L}$. A quantifier-free formula equivalent to ϕ in DNF can be computed in PSPACE.*

6.7 Conclusion

In this chapter we presented efficient quantifier elimination and decision algorithms for the theory of discrete time diPC and the theory of dense time dePC. The theory diPC is a special case of the theory of Presburger arithmetic. The theory dePC is a special case of the theory of real addition with order. The most important result of this chapter is that diPC and dePC can be decided in PSPACE, while real addition with order needs single exponential space and Presburger arithmetic needs double exponential space. The PSPACE bounds for these two theories imply similar bounds for the rest of the theories of data and time considered in this dissertation.

Chapter 7

Complexity of Query Evaluation in Temporal Constraint Databases

In this chapter we exploit the results obtained in chapter 6 for calculating precise bounds for query evaluation in temporal constraint databases and indefinite temporal constraint databases.

We will measure the complexity of query evaluation using the notions of data complexity and combined complexity introduced in [Var82]. When we consider *combined complexity*, we measure the complexity of query evaluation as a function of all the parameters involved: the size of the database, the size of the schema and the size of the query. When we consider *data complexity*, we measure the complexity of query evaluation as a function of the database size only; the query and the schema are considered *fixed*.

The most important results of this chapter can be summarized as follows:

- Evaluating queries in relational calculus with diPCL- or dePCL-constraints over diPCL- or dePCL-constraint databases can be done with LOGSPACE data complexity and PSPACE-complete combined complexity.

- Evaluating $\Box$-queries in modal relational calculus with diPCL- or dePCL-constraints over indefinite diPCL- or dePCL-constraint databases can be done with co-NP-complete data complexity and and PSPACE-complete combined complexity. The same problem for $\Diamond$-queries has NP-complete data complexity and PSPACE-complete combined complexity. If we consider $\Box$-queries with k alternations of quantifiers then the combined complexity drops to Π^p_{k+1}-complete.

These results demonstrate that the complexity of query evaluation *does not change* when we move from relational calculus queries over relational databases to relational calculus with temporal constraints over temporal constraint databases. The same result is true when we consider indefinite information.

This chapter is organized as follows. In sections 7.1 and 7.2 we analyze the complexity of evaluating queries in relational calculus with temporal constraints over temporal constraint databases. Finally, in section 7.3 we study the complexity of evaluating queries in modal relational calculus with temporal constraints over indefinite temporal constraint databases.

7.1 DCL-, diPCL- and diPDCL-Constraint Databases

We will start with yes/no queries in relational calculus with DCL-constraints. The following theorems are from [KKR90].[1]

Theorem 7.1.1 *Let $\tilde{r}$ be a DCL-constraint database and f be a yes/no query in relational calculus with DCL-constraints. Then the problem of deciding whether $f(\tilde{r}) = yes$ can be solved with LOGSPACE data complexity and PSPACE-complete combined complexity.*

Theorem 7.1.2 *Let $\tilde{r}$ be a DCL-constraint database and f be a $\exists_k$ yes/no expression of relational calculus with DCL-constraints. The problem of deciding whether $f(\tilde{r}) = yes$ has Σ_k^p-complete combined complexity.*

A corresponding Π_k^p bound can be established for $\forall_k$ yes/no expressions.

Theorem 7.1.3 *Let $\tilde{r}$ be a DCL-constraint database and f be a query in relational calculus with DCL-constraints. A DCL-constraint relation r such that $r = f(\tilde{r})$ can be computed with LOGSPACE data complexity.*

Let us now turn to diPCL- and diPDCL-constraint databases.

Theorem 7.1.4 *Let $\tilde{r}$ be a diPCL-constraint (resp. diPDCL-constraint) database and f be a yes/no query in relational calculus with diPCL-constraints (resp. diPDCL-constraints). The problem of deciding whether $f(\tilde{r}) = yes$ can be solved with LOGSPACE data complexity and PSPACE-complete combined complexity.*

Proof: We will only consider the diPCL case. The proof for the other case is similar. To evaluate $f(\tilde{r})$ we can proceed as follows. First, we transform the diPCL$^+$ formula of f into PNF. Secondly, we form the formula ϕ of diPCL which corresponds to f and $\tilde{r}$. Then we use the quantifier elimination algorithm DIPC-EVAL to decide whether $\mathbf{Z} \models \phi$.

For combined complexity the upper bound follows from the space estimates of theorem 6.4.1 and the lower bound from an easy reduction from QBF. For data complexity the result can be deduced from the space estimates of theorem 6.4.1 and the following observations:

- The transformation to PNF can be done within the claimed space bound because f is fixed.
- The depth of the recursion in DIPC-EVAL is fixed.
- We do not really need to create the diPCL formula corresponding to the query and the database. Instead, we can switch to the database whenever the query contains a database predicate symbol [KKR90].

For data complexity the LOGSPACE upper bound can also be achieved if, instead of DIPC-EVAL, we use the decision procedure of [FR75] for the first-order theory of real addition with order. ■

Let us now consider queries with a fixed level of quantifier alternation.

Theorem 7.1.5 *Let $\tilde{r}$ be a diPCL-constraint (resp. diPDCL-constraint) database and f be a $\exists_k$ yes/no expression of relational calculus with diPCL-constraints (resp. diPDCL-constraints). The problem of deciding whether $f(\tilde{r}) = yes$ has Σ_k^p-complete combined complexity.*

[1] [KKR90] consider only data complexity. The results for combined complexity follow easily from their analysis.

Proof: Similar to the proof of theorem 7.1.4. ■

A corresponding Π_k^p bound can be established for $\forall_k$ yes/no expressions of relational calculus with diPCL-constraints (resp. diPDCL-constraints).

Let us now consider queries with free variables.

Theorem 7.1.6 *Let $\widetilde{r}$ be a diPCL-constraint (resp. diPDCL-constraint) database and f be a query in relational calculus with diPCL-constraints (resp. diPDCL-constraints). A diPCL-constraint (resp. diPDCL-constraint) relation r such that $r = f(\widetilde{r})$ can be computed with LOGSPACE data complexity.*

Proof: We give the proof for the case of diPDCL-constraint databases. Let f be $\{R(\overline{X}), \overline{x}/\overline{s} : \phi(\overline{x})\}$. First, we compute the diPDCL formula $\phi'(\overline{x})$ which corresponds to f and $\widetilde{r}$. Then we use the algorithm described in theorem 6.4.6 to find a quantifier-free formula ϕ'' in DNF which is equivalent to ϕ'. Finally, each disjunct of ϕ'' becomes the local condition of a tuple of $f(\widetilde{r})$. Theorem 6.4.6 shows that this computation can be carried out with LOGSPACE data complexity. ■

The same complexity bounds can be achieved for the case of diTCL- and diTDCL-constraint databases by using the above results and the translations of section 3.2.3. We do not study these cases in detail since they are straightforward.

7.2 dePCL-Constraint Databases

Let us now turn to dePCL-constraint databases. We will first consider yes-no queries in relational calculus with dePCL-constraints.

Theorem 7.2.1 *Let $\widetilde{r}$ be a dePCL-constraint database and f be a yes/no query in relational calculus with dePCL-constraints. Then the problem of deciding whether $f(\widetilde{r}) = yes$ has PSPACE-complete combined complexity.*

Proof: To evaluate $f(\widetilde{r})$ we proceed as follows. First, we tranform the dePCL$^+$ formula of f into PNF. Secondly, we form the formula ϕ of dePCL which corresponds to f and $\widetilde{r}$. Then we use the quantifier elimination algorithm dePC-Eval to decide whether $\mathbf{Q} \models \phi$. Theorem 6.5.1 shows that this computation can be carried out in PSPACE. The lower bound follows easily from a reduction from QBF [Sto77]. ■

Let us now consider queries with a fixed level of quantifier alternation.

Theorem 7.2.2 *Let $\widetilde{r}$ be a dePCL-constraint database and f be a $\exists_k$ yes/no expression of relational calculus with dePCL-constraints. The problem of deciding whether $f(\widetilde{r}) = yes$ has Σ_k^p-complete combined complexity.*

Proof: Similar to the proof of theorem 7.2.1. ■

A corresponding Π_k^p bound can be established for $\forall_k$ yes/no expressions of relational calculus with dePCL-constraints.

For data complexity we cannot achieve a LOGSPACE bound using the decision procedure for dePC developed in section 6.5. The problem is that we might need at most $O(|\phi| \log |\phi|)$ space for storing the product of all denominators of fractions in some dePCL formula ϕ. However, under this complexity measure, the decision procedure of [FR75] for the first order theory of real addition with order can be implemented in LOGSPACE. Therefore, we have the following result.

Theorem 7.2.3 *Let $\widetilde{r}$ be a dePCL-constraint database and f be a yes/no query in relational calculus with dePCL-constraints. Then the problem of deciding whether $f(\widetilde{r}) = yes$ has LOGSPACE data complexity.*

We will now consider queries with free variables. Let us assume we are given dePCL-constraint database $\widetilde{r}$ and query $\{R(\overline{T}), \overline{t}/\mathcal{Q} : \ \phi(\overline{t})\}$ in relational calculus with dePCL-constraints. At first, using lemma 7.2.1 below, we estimate how large rational constants can grow in the constraints of the answer. Secondly, we use this information to construct a finite partition of the space $\mathbb{Q}^{|\overline{t}|}$ into *regions* with the following properties:

1. Every region can be represented by a conjunction of dePCL-constraints.
2. The truth value of the sentence $\phi(\overline{\tau})$ is the same for all points $\overline{\tau}$ in the same region.

Then, using the algorithm of [FR75], we can check whether *all* points in a region satisfy $\phi(\overline{t})$ by picking a *single* point in the region and checking whether $\mathbf{Q} \models \phi(\overline{\tau})$ is true.

We start with a definition from [FR75]. In this definition the meaning of term "limited" is different than the one in definition 6.5.1.

Definition 7.2.1 Let r be a rational number. We will say that r is *limited* by the positive integer L, denoted by $r \preceq L$, iff there exist integers a and b such that $r = a/b$ and $|a| \leq L$ and $|b| \leq L$.

The following lemma estimates how large rational constants can grow in the quantifier-free formula equivalent to a given dePCL formula ϕ.

Lemma 7.2.1 *Let $\phi(\overline{t})$ be a dePCL formula with K quantifiers and $\phi'(\overline{t})$ be the quantifier free formula equivalent to $\phi(\overline{t})$ which is computed by the algorithm of proposition 6.2.2. Every rational constant of ϕ' is limited by*

$$2^{2^K(\log(maxabs(\phi))+1)}.$$

Proof: If r_1, r_2 are rationals such that $r_1, r_2 \preceq L$ then $r_1 + r_2 \preceq 2L^2$. Thus after the first quantifier of ϕ is eliminated the rational constants of the resulting formula are limited by $2maxabs(\phi)^2$. The result follows by a simple inductive argument. ■

Let us now introduce the machinery required for presenting our method. We first define the concept of a rational BICN and its dePCL formula. These definitions are different than definitions 6.5.3 and 6.5.4.

Definition 7.2.2 Let Q be a set such that $Q \subseteq \mathbb{Q}$. A *rational BICN with bounds from the set Q* is a consistent BICN with all edges labeled with $[c, c], (c, d), (c, \infty)$ or $(-\infty, c)$ where $c < d$ and $c, d \in Q$.

Definition 7.2.3 Let N be a rational BICN of size n. The formula with free variables $\overline{x} = (x_1, \ldots, x_n)$ corresponding to N is the dePCL formula $\bigwedge_{\theta \in Constraints(M)} \theta$ where M is the minimal network equivalent to N. This formula will be denoted by $\Phi(N)$.

Notation 7.2.1 If ϕ is a dePCL formula with K quantifiers and vector of free variables $\overline{t}$ then Q_ϕ will denote the set of rationals

$$\{r : \ r \in \mathbb{Q} \text{ and } |r| \leq |\overline{t}|\, 2^{2^K(\log(maxabs(\phi))+1)}\}.$$

The following lemmas help to establish the main result. The similarity with the development of section 6.5.1 is obvious. Lemma 7.2.2 tells us how to partition the answer space into regions. Subsequent lemmas give properties of rational BICN. The proofs of the lemmas and the subsequent main results are omitted.

Lemma 7.2.2 *If $\phi(\bar{t})$ is a dePCL formula then there is a quantifier-free formula $\phi'(\bar{t})$ equivalent to ϕ with the following properties:*

1. *ϕ' is in DNF*
2. *Every disjunct of ϕ' is the dePCL formula corresponding to a rational BICN N of size $|\bar{t}|$ with bounds from the set Q_ϕ.*

Lemma 7.2.3 *Let $\overline{\tau}$ be an element of $\mathbb{Q}^n$. For any set $Q \subseteq \mathbb{Q}$, there exists a unique rational BICN N with bounds from Q such that $\mathbf{Q} \models \Phi(N)(\overline{\tau})$.*

Lemma 7.2.4 *Let $\phi(\bar{t})$ be a dePCL formula and N be a rational BICN with bounds from the set Q_ϕ. Then $\mathbf{Q} \models \Phi(N) \supset \phi$ iff $\mathbf{Q} \models \phi(\overline{\tau})$ is true for an arbitrary $\overline{\tau}$ such that $\mathbf{Q} \models \Phi(N)(\overline{\tau})$.*

The following theorem gives the main result of this section.

Theorem 7.2.4 *Let $\tilde{r}$ be a dePCL-constraint database and f be a query in relational calculus with dePCL-constraints. A dePCL-constraint relation r such that $r = f(\tilde{r})$ can be computed with LOGSPACE data complexity.*

Proof: Let f be $\{R(\overline{T}), \bar{t}/\mathcal{Q} : \phi(\bar{t})\}$. Let us compute the dePCL formula $\phi'(\bar{t})$ corresponding to f and $\tilde{r}$. We can now find a quantifier-free formula equivalent to ϕ' in DNF as follows. We generate one by one all rational BICN of size $|\bar{t}|$ with bounds from Q_ϕ. For every such BICN N, we find a solution $\overline{\tau}$ of N and check whether $\mathbf{Q} \models \phi(\overline{\tau})$ using the decision procedure of [FR75] for the first-order theory of real addition with order. If this check succeeds then we form a tuple of r from the formula $\Phi(N)$ corresponding to N. The correctness of this procedure follows from the above lemmas. The entire computation can be carried out with LOGSPACE data complexity. ■

We can now easily obtain the following results which deal with dePDCL-constraint databases.

Theorem 7.2.5 *Let $\tilde{r}$ be a dePDCL-constraint database and f be a yes/no query in relational calculus with dePDCL-constraints. Then the problem of deciding whether $f(\tilde{r}) = yes$ has LOGSPACE data complexity and PSPACE-complete combined complexity.*

Theorem 7.2.6 *Let $\tilde{r}$ be a dePDCL-constraint database and f be a $\exists_k$ yes/no expression of relational calculus with dePDCL-constraints. The problem of deciding whether $f(\tilde{r}) = yes$ has Σ_k^p-complete combined complexity.*

Theorem 7.2.7 *Let $\tilde{r}$ be a dePDCL-constraint database and f be a query in relational calculus with dePDCL-constraints. A dePDCL-constraint relation r such that $r = f(\tilde{r})$ can be computed with LOGSPACE data complexity.*

The same complexity bounds can be achieved for the case of deTCL- and deTDCL-constraint databases by using the above results and the translations of section 3.2.3. We will again omit these cases.

7.3 Indefinite Temporal Constraint Databases

Let us now consider the problem of query evaluation in indefinite temporal constraint databases. We will only consider queries in modal relational calculus with temporal constraints. This case is more interesting and it is also the one usually studied in the literature [Var86, Gra89, AKG91]. Queries in relational calculus with temporal constraints over indefinite temporal constraint databases can be studied using theorem 5.3.2 and the methodology of sections 7.1 and 7.2. The only thing worth noticing here is that the number of e-variables is *not* fixed so the results will not follow the pattern of the results of the previous sections.

The following theorems demonstrate that evaluation of queries in modal relational calculus with temporal constraints over indefinite temporal constraint databases is probably intractable (given our current understanding of complexity classes [Joh90]).

Theorem 7.3.1 *Let $\mathcal{L}$ be DCL, diPCL, dePCL, diPDCL or dePDCL. Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database and f be a yes/no $\Diamond$-query in modal relational calculus with $\mathcal{L}$-constraints. Then the problem of deciding whether $f(\widetilde{r}) = yes$ is NP-complete for data complexity and PSPACE-complete for combined complexity.*

Proof: *Upper bound.* Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database, $\overline{\omega}$ be the vector of e-variables of sort $\overline{s}$ in $\widetilde{r}$, $G(\overline{\omega})$ be the global condition of $\widetilde{r}$ and f be the query $\{: \Diamond\phi\}$. Theorem 5.3.3 tells us that $f(\widetilde{r}) = yes$ iff

$$\mathbf{M}_{\mathcal{L}} \models (\exists\overline{\omega}/\overline{s})(G(\overline{\omega}) \ \wedge \ \psi(\overline{\omega})) \tag{7.1}$$

where ψ is the formula of $\mathcal{L}$ corresponding to ϕ and $\widetilde{r}$. For combined complexity, the upper bound follows immediately from theorems 6.3.1, 6.4.1, 6.5.1 and 6.6.1.

Let us now consider data complexity. For indefinite DCL-constraint databases, [KKR90] implies that

$$\mathbf{D} \models (\exists\overline{\lambda}/\mathcal{D})(G(\overline{\lambda}) \ \wedge \ \psi(\overline{\lambda}))$$

if and only if there exists a d-configuration δ in variables $\overline{\lambda}$ such that $G \wedge \psi$ is true when variables $\overline{\lambda}$ are interpreted according to δ. Therefore a non-deterministic Turing machine can guess a d-configuration δ (in polynomial time) and then verify the truth of $G \ \wedge \ \psi$ using the LOGSPACE algorithm of section 6.3. Therefore the whole computation can be performed in polynomial time.

For indefinite diPCL-constraint databases, if $\overline{\omega} = (\omega_1, \ldots, \omega_N)$ then lemma 6.4.2 tells us that (7.1) is equivalent to

$$\begin{array}{rc}
\mathbf{Z} \models & (\exists\omega_1 \preceq 2^{N+K-1}(maxabs(G \ \wedge \ \phi) + 1) + 2) \\
& (\exists\omega_2 \preceq 2^{N+K}(maxabs(G \ \wedge \ \phi) + 1) + 3) \\
& \cdots \\
& (\exists\omega_N \preceq 2^{2N+K-2}(maxabs(G \ \wedge \ \phi) + 1) + N + 1) \\
& (G(\omega_1, \ldots, \omega_N) \ \wedge \ \psi(\omega_1, \ldots, \omega_N)).
\end{array}$$

Therefore a non-deterministic Turing machine can guess values $\beta_1, \ldots, \beta_N$ for $\omega_1, \ldots, \omega_N$ in the ranges specified above, and then verify whether

$$\mathbf{Z} \models G(\beta_1, \ldots, \beta_N) \ \wedge \ \psi(\beta_1, \ldots, \beta_N).$$

The guessing step can be done in polynomial time because the sizes of $\beta_1, \ldots, \beta_N$ do not exceed $(2N + K - 1)\log(maxabs(G \ \wedge \ \psi))$ bits. The verification step can also be done in polynomial time (theorem 7.1.4). Therefore the whole computation can be performed in polynomial time by a non-deterministic Turing machine. The proofs for dePCL, diPDCL and dePDCL are similar.

Lower bounds. The PSPACE lower bound is obvious. For any language $\mathcal{L}$ given above, the problem of query evaluation in relational calculus with $\mathcal{L}$-constraints over $\mathcal{L}$-constraint databases has PSPACE-complete combined complexity (theorems 7.1.1, 7.1.4, 7.2.1 and 7.2.5).

Let us then consider data complexity. For the case of indefinite DCL-constraint databases, theorem 5.2(2) of [AKG91] shows that there exists a formula ϕ of DCL$^+$ such that the query $\{: \Diamond\neg\phi\}$ has NP-hard data complexity on indefinite DCL-constraint databases restricted as follows:[2]

- The database contains only e-variables and constants.
- All e-variables are distinct.
- All conditions (local and global) are *true*.

For the case of diPDCL-constraint (resp. dePDCL-constraint) databases, theorem 3.3 of [vdM92] shows that there exists a conjunctive diPDCL$^+$ sentence ϕ such that the query $\{: \Diamond\neg\phi\}$ has NP-hard data complexity on diPDCL-constraint (resp. dePDCL-constraint) databases restricted as follows:

- Data are represented in their standard form (i.e., as in chapter 4).
- The constraints are only qualitative $<$-constraints.
- The database contains only e-variables and constants.
- Every relation has only two attributes (one attribute of sort $\mathcal{D}$ and one attribute of sort $\mathcal{Z}$ (resp. $\mathcal{Q}$)) and local conditions *true*.

■

The following theorem is an easy consequence of the previous one.

Theorem 7.3.2 *Let $\mathcal{L}$ be DCL, diPCL, dePCL, diPDCL or dePDCL. Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database and f be a yes/no $\Box$-query in modal relational calculus with $\mathcal{L}$-constraints. Then the problem of deciding whether $f(\widetilde{r}) = yes$ is co-NP complete for data complexity and PSPACE-complete for combined complexity.*

Proof: Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database and f be the query $\{: \Box\phi\}$. Then $f(\widetilde{r}) = yes$ iff $f'(\widetilde{r}) = no$ where $f' = \{: \Diamond\neg\phi\}$. The results follow easily from theorem 7.3.1. ■

Let us now consider queries with a fixed alternation of quantifiers.

Theorem 7.3.3 *Let $\mathcal{L}$ be DCL, diPCL, dePCL, diPDCL or dePDCL. Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database and f be a $\exists_k$ yes/no $\Box$-query in modal relational calculus with $\mathcal{L}$-constraints. The problem of deciding whether $f(\widetilde{r}) = yes$ is Π^p_{k+1}-complete for combined complexity.*

Proof: *Upper bounds.* Let $\widetilde{r}$ be an indefinite $\mathcal{L}$-constraint database, $\overline{\omega}$ be the vector of e-variables of sort $\overline{s}$ in $\widetilde{r}$, $G(\overline{\omega})$ be the global condition of $\widetilde{r}$ and f be the query $\{: \Box\phi\}$. Theorem 5.3.4 tells us that $f(\widetilde{r}) = yes$ iff

$$\mathbf{M}_{\mathcal{L}} \models (\forall\overline{\omega}/\overline{s})(G(\overline{\omega}) \supset \psi(\overline{\omega}))$$

where ψ is the formula $\mathcal{L}$ corresponding to ϕ and $\widetilde{r}$. Thus we have to decide a $\forall_{k+1}$ formula of $\mathcal{L}$. The upper bounds now follow from theorems 6.3.2, 6.4.4, 6.5.5 and 6.6.2.

[2] In the terminology of [AKG91], the database is restricted to consist of *Codd tables* only.

Lower bounds. The lower bounds follow easily from reductions from $\mathrm{QBF}^{\forall}_{k+1}$. ■

Corresponding Σ^p_{k+1} bounds can be established for $\forall_k$ $\diamond$-queries in modal relational calculus with $\mathcal{L}$-constraints.

The same complexity bounds can be achieved for the case of indefinite diTCL- or diTDCL-constraint databases by using the above results and the translations of section 3.2.3.

7.4 Conclusion

In this chapter we established precise bounds for query evaluation in (indefinite) temporal constraint databases. Our results are summarized in the following table.

Complexity type	Types of databases and queries				
	TCDB		ITCDB		
	RC+TC	$\exists_k$ RC+TC	$\diamond$+RC+TC	$\Box$+RC+TC	$\exists_k$ $\Box$+RC+TC
Data complexity	LOGSPACE	—	NP	co-NP	—
Combined complexity	PSPACE	Σ^p_k	PSPACE	PSPACE	Π^p_{k+1}

In this table

- TCDB and ITCDB stand for temporal constraint databases and indefinite temporal constraint databases,
- RC+TC stands for relational calculus with temporal constraints,
- $\exists_k$ RC+TC stands for $\exists_k$ expressions of relational calculus with temporal constraints,
- $\diamond$+RC+TC stands for $\diamond$-expressions of modal relational calculus with temporal constraints,
- $\Box$+RC+TC stands for $\Box$-expressions of modal relational calculus with temporal constraints,
- $\exists_k$ $\Box$+RC+TC stands for $\exists_k$ $\Box$-expressions of modal relational calculus with temporal constraints, an
- a table entry $\mathcal{C}$, where $\mathcal{C}$ is a complexity class, means that the corresponding query answering problem is complete in the class $\mathcal{C}$.

Taking now into account the results of [CH82, Var82, Var86, AKG91] we can conclude: from a complexity-theoretic perspective, query evaluation in temporal constraint databases and query evalution in standard relational databases have the *same data and combined complexity*. This is a very encouraging theoretical result which needs to be complemented by implementations of temporal constraint databases.

This chapter finishes the development of the constraint database schemes and their realization in temporal constraint databases. In the next chapter we will extend the class of dePCL-constraints to include inequations (e.g., $x_1 \neq x_2$) and disjunctions of inequations (e.g., $x_1 \neq x_2 \vee x_5 - x_3 \neq 6$). This is an interesting class of temporal constraints where most of the basic problems studied in chapter 3 remain tractable.

Chapter 8

Temporal Constraints with $\neq$

In this and the following chapter we extend the class of dePCL-constraints to include inequations (e.g., $x_1 \neq x_2$, $x_1 - x_3 \neq 4$) or disjunctions of inequations (e.g., $x_1 \neq 5 \vee x_3 - x_4 \neq 7$). Inequations are not encountered in practice as often as inequalities. This is probably the reason why, in the published literature, inequations have received less attention than inequalities.

Inequations of the form $t_1 \neq t_2$ have been considered by [VK86, vB90a, GS93, GSS93] in the context of PA networks. Only [Mei91b] has studied inequations of the form $t \neq r$ (r a real constant) in the context of point networks with *almost-single-interval domains*. To the best of our knowledge, only [Kou92] considered inequations of the form $t_1 - t_2 \neq r$ and disjunctions of such inequations.[1] This chapter is largely based on [Kou92] while chapter 9 extends those results in several directions.

The basic properties of disjunctions of linear inequations have been initially studied by [LM89] who considered *generalized linear constraints* i.e., linear inequalities and disjunctions of linear inequations over the reals. [LM89] observed that disjunctions of inequations can be treated independently of one another for deciding consistency. An important consequence of this fact is that one can decide in polynomial time whether a set of generalized linear constraints (or dePCL-constraints) is consistent. Let us stress that a similar result cannot be achieved for the case of diPCL-constraints. If the class of diPCL-constraints is extended to include inequations of the form $x \neq y$ (where x and y are integer variables) then the consistency problem for this extended class becomes NP-complete (trivial reduction from 3-colorability [Cha77]).

The contributions of this chapter can be summarized as follows:

- We apply the results of [LM89] to the case of dePCL-constraints. In particular, we develop algorithms for deciding consistency (section 8.2) and computing a canonical form (section 8.3).
- We develop a variable elimination algorithm for dePCL-constraints (section 8.4). This algorithm can also be applied to the more expressive class of generalized linear constraints. The same algorithm has been developed independently in [Imb93].
- We demonstrate that there is a set C of dePCL-constraints such that the following property holds. If m variables are eliminated from C, the resulting constraint set cannot be

[1] Inequations of the form $t_1 - t_2 \neq r$ or $t \neq r'$ (r, r' are real constants) can be represented in the inequality constraint networks of [DMP91] and the generalized temporal constraint networks of [Mei91b]. However no particular attention has been paid to this class of constraints.

represented by less than $O(2^m)$ dePCL-constraints. In chapter 9 we identify subclasses of dePCL-constraints for which this undesirable property no longer holds.

This chapter is organized as follows. Section 8.1 gives definitions and examples. Sections 8.2 and 8.3 discuss consistency checking and canonical forms for dePCL-constraints. Finally, section 8.4 presents our results on variable elimination.

8.1 Preliminaries

Let us start by redefining the class of dePCL-constraints.

Definition 8.1.1 A dePCL-*constraint* is a dePCL formula of any of the following forms:

- *An equality or inequality:* $t \sim r$ where t is a term, r is a constant and $\sim$ is $=$, $<$, $>$, $\leq$ or $\geq$.
- *A disjunction of inequations*: $t_1 \neq r_1 \ \vee \ \cdots \ \vee \ t_n \neq r_n$ where $t_1, \cdots, t_n$ are terms and $r_1, \cdots, r_n$ are constants.

Notation 8.1.1 If c is a disjunction of inequations then $\overline{c}$ will denote the conjunction of equations obtained by negating c. If c is a conjunction of equations then $\overline{c}$ will denote the disjunction of inequations obtained by negating c.

If C is a set of disjunctions of inequations in n variables then the solution set of C is $\mathbb{Q}^n \setminus Sol(\{\overline{c} : c \in C\})$. Let $C = C_e \cup C_i \cup C_d$ be a set of dePCL-constraints in n variables where C_e is a set of equalities, C_i is a set of inequalities and C_d is a set of disjunctions of inequations. The solution set of C is a convex polyhedron in $\mathbb{Q}^n$ defined by $C_e \cup C_i$ minus the set of affines spaces defined by C_d.

As we said in the introduction of this chapter, disjunctions of inequations are useful in several situations.

Example 8.1.1 The following are some simple statements which can be formalized using disjunctions of inequations:

- Jobs $J1$ and $J2$ should not both start at the time job J starts:[2] $J_s \neq J1_s \ \vee \ J_s \neq J2_s$
- It took robots A and B at most five minutes to finish job J; however, one of the robots worked for a shorter time:
  $A_e - A_s \leq 5, \ B_e - B_s \leq 5, \ A_e - A_s \neq 5 \ \vee \ B_e - B_s \neq 5$
- Interval i is not equal to interval j: $i_s \neq j_s \ \vee \ i_e \neq j_e$

Example 8.1.2 Let us now reconsider the story of example 4.8.5. "David and Murray arrived at the accident scene together. Murray left at least five minutes later than Bill. Bill left at least three minutes later than David. John and Bill did not leave the scene together."

The indefinite dePCL-constraint relation SCENE represents the information in the above story. The e-variables d_s, d_e, m_s, m_e, b_s, b_e and j_s, j_e denote the endpoints of the time intervals David, Murray, Bill and John were at the accident scene. In this case the statement John and Bill did not leave the scene together" is represented by the inequation $b_e \neq j_e$.

[2] In this and the following examples i_s and i_e denote the left and right endpoint of an interval i respectively.

SCENE

PERSON	FROM	TO	CON
David	d_s	d_e	$true$
Murray	m_s	m_e	$true$
Bill	b_s	b_e	$true$
John	j_s	j_e	$true$

$$G: \ d_s = m_e \ \wedge \ m_e - b_e \geq 5 \ \wedge \ b_e - d_e \geq 3 \ \wedge \ b_e \neq j_e.$$

Let us now assume that the query "What are the times that John, David and Murray were possibly at the scene?" is posed. In modal relational calculus with dePCL-constraints this query is expressed by

$$\{TIMES(JS,\ JE,\ DS,\ DE,\ MS,\ ME),\ J_s/\mathcal{Q}, J_e/\mathcal{Q}, D_s/\mathcal{Q}, D_e/\mathcal{Q}, M_s/\mathcal{Q}, M_e/\mathcal{Q} : \\ \Diamond(SCENE(John, J_s, J_e) \wedge SCENE(David, D_s, D_e) \wedge SCENE(Murray, M_s, M_e))\}.$$

The equivalent expression in modal $\mathcal{L}$-constraint algebra is $POSS(CP)$ where CP is the cartesian product of expressions

$$\theta_{JE}^{TO}(\theta_{JS}^{FROM}(\pi_{FROM,TO}(\sigma_{NAME=John}(SCENE)))),$$

$$\theta_{DE}^{TO}(\theta_{DS}^{FROM}(\pi_{FROM,TO}(\sigma_{NAME=David}(SCENE))))$$

and

$$\theta_{ME}^{TO}(\theta_{MS}^{FROM}(\pi_{FROM,TO}(\sigma_{NAME=Murray}(SCENE)))).$$

When we evaluate operation POSS, we have to eliminate all e-variables from the dePCL formula

$$J_s = j_s \ \wedge \ J_e = j_e \ \wedge \ D_s = d_s \ \wedge \ D_e = d_e \ \wedge \ M_s = m_s \ \wedge \ M_e = m_e \ \wedge \\ d_l = m_e \ \wedge \ m_e - b_e \geq 5 \ \wedge \ b_e - d_e \geq 3 \ \wedge \ b_e \neq j_e$$

where $J_s,\ J_e,\ D_s,\ D_e,\ M_s,\ M_e$ are u-variables introduced for tranforming relation SCENE into the form used in the definitions of section 4.2.

The above query will be answered by returning the following dePCL-constraint relation.

TIMES

JS	JE	DS	DE	MS	ME	CON
J_s	J_e	D_s	D_e	M_s	M_e	$D_s < D_e,\ M_s < M_e,\ J_s < J_e,\ D_s = M_s,$ $M_e - D_e \geq 8,\ J_e - D_e \neq 3 \ \vee \ J_e - M_e \neq -5$

The above computation is more efficient than the computation performed in example 4.8.5. We do not have to transform in DNF before eliminating e-variables and the resulting relation has only one tuple.

The last example is important. It shows that an inequation can give rise to a disjunction of inequations when variable elimination is performed. This fact motivated much of the work presented in this and the following chapter.

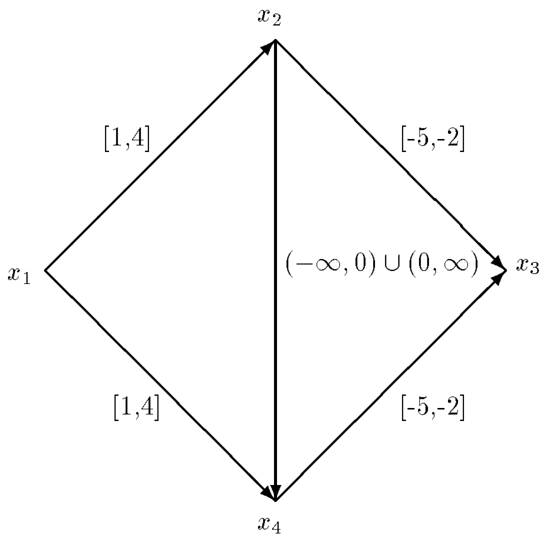

Figure 8.1: A binary constraint network

Binary Constraint Networks

We can extend the binary inequality constraint networks of section 3.3 to accomodate inequations (but *not* disjunctions of inequations). Let C be a set of inequalities and inequations in variables $x_1, \ldots, x_n$. The *binary constraint network* (BCN) associated with C is a labeled directed graph $G = (V, E)$ where $V = \{1, \ldots, n\}$. Node i represents variable x_i and edge (i, j) represents the binary constraints involving x_i and x_j.

Let us assume that the set of constraints c_{ij} on $x_j - x_i$ is

$$\{x_j - x_i \leq d_{ij},\ x_j - x_i \geq -d_{ji},\ x_j - x_i \neq r_{ji}^1, \ldots, x_j - x_i \neq r_{ji}^{h_{ji}}\}$$

where $-d_{ji} < r_{ji}^1 < \cdots < r_{ji}^{h_{ji}} < d_{ij}$. Then the corresponding BCN N will have an edge $i \rightarrow j$ labeled by the *almost-convex interval*

$$N_{ij} = [-d_{ji}, r_{ji}^1) \cup (r_{ji}^1, r_{ji}^2) \cup \cdots \cup (r_{ji}^{h_{ji}-1}, r_{ji}^{h_{ji}}) \cup (r_{ji}^{h_{ji}}, d_{ij}].$$

Interval N_{ij} can also be open from the right or left depending on the form of the inequalities in c_{ij}. The h_{ij} values $r_{ji}^1, \ldots, r_{ji}^{h_{ji}}$ will be called the *"holes"* of interval N_{ij}. We define a function *holes* such that, for each interval N_{ij}, $holes(N_{ij}) = \{r_{ji}^1, \ldots, r_{ji}^{h_{ji}}\}$. Unary constraints will again be represented as binary constraints with the introduction of a special variable $x_0 = 0$.

Example 8.1.3 The BCN of figure 8.1 represents the constraints

$$1 \leq x_2 - x_1 \leq 4,\ 2 \leq x_2 - x_3 \leq 5,\ 1 \leq x_4 - x_1 \leq 4,\ 2 \leq x_4 - x_3 \leq 5,\ x_4 - x_2 \neq 0$$

Given an interval I, $conv(I)$ will denote the *convex hull* of I i.e., the minimal (in the set-theoretic sense) convex interval which includes I. Formally,

$$conv([a, r_1) \cup (r_1, r_2) \cup \cdots \cup (r_{k-1}, r_k) \cup (r_k, b]) = [a, b]$$

and $conv(I) = I$ if I is convex. If N is a BCN then $conv(N)$ denotes the BCN (in fact, BICN) which is obtained from N by substituting each interval N_{ij} by $conv(N_{ij})$.

For the case of BCN, the operations of composition and intersection of almost-convex intervals are defined as for BICN (see section 3.3). The following proposition is straightforward.

Proposition 8.1.1 The class of *almost-convex intervals* over $\mathbb{Q}$ is closed under composition and intersection.

Let us now define the function *Constraints* for BCN. If N is a BCN with n nodes then $Constraints(N)$ is a set of constraints in variables $x_1, \ldots, x_n$ which is defined as follows:

$$Constraints(N) = Constraints(conv(N)) \cup \{x_j - x_i \neq r^g_{ji} \mid r^g_{ji} \in holes(N_{ij}),\ i < j\}$$

where

$$\begin{aligned}
Constraints(conv(N)) \quad = \quad & \{x_j - x_i \leq d_{ij},\ x_j - x_i \geq -d_{ji} \mid N_{ij} = [-d_{ji}, d_{ij}],\ i < j\} \cup \\
& \{x_j - x_i < d_{ij},\ x_j - x_i \geq -d_{ji} \mid N_{ij} = [-d_{ji}, d_{ij}),\ d_{ij} \neq \infty,\ i < j\} \cup \\
& \{x_j - x_i \leq d_{ij},\ x_j - x_i < -d_{ji} \mid N_{ij} = (-d_{ji}, d_{ij}],\ d_{ji} \neq \infty,\ i < j\} \cup \\
& \{x_j - x_i < d_{ij},\ x_j - x_i < -d_{ji} \mid N_{ij} = (-d_{ji}, d_{ij}),\ d_{ij}, d_{ji} \neq \infty,\ i < j\} \cup \\
& \{x_j - x_i \geq -d_{ji} \mid N_{ij} = [-d_{ji}, \infty),\ i < j\} \cup \\
& \{x_j - x_i \leq d_{ij}, \mid N_{ij} = (-\infty, d_{ij}],\ i < j\} \cup \\
& \{x_j - x_i < -d_{ji} \mid N_{ij} = (-d_{ji}, \infty),\ i < j\} \cup \\
& \{x_j - x_i < d_{ij}, \mid N_{ij} = (-\infty, d_{ij}),\ i < j\}.
\end{aligned}$$

The notions of solution, consistency, equivalence and minimality are defined as for BICN.

8.2 Consistency of dePCL-Constraints

In section 3.4 we studied consistency of dePCL-inequalities in some depth. We will now consider dePCL-inequations and disjunctions of dePCL-inequations.

The following lemma of [LM89] shows that disjunctions of inequations can be treated independently of one another for deciding consistency. This result is very important since it allows one to avoid the combinatorial explosion resulting from translating inequations into disjunctions of strict inequalities.

Lemma 8.2.1 *(Independence of disjunctions of inequations.) Let C_i be a set of inequalities and C_d be a set of disjunctions of inequations. The set $C_i \cup C_d$ is consistent if and only if C_i is consistent and each disjunction of inequations in C_d is consistent with the inequalities in C_i.*

Proof: To verify the lemma notice that a set of inequalities in k variables defines a polyhedral set P in $\mathbb{Q}^k$ while the complement of a disjunction of inequations c defines a hyperplane H_c of dimension less than the dimension of P. Now assume that C_p is consistent and each constraint c in C_d is consistent with the constraints in C_p. Therefore $H_c \not\subseteq P$. We can now conclude that the set of constraints C is consistent since it would be impossible to achieve inconsistency: we cannot cover a set of dimension d (i.e., P) with a finite set of objects of smaller dimension (i.e., the hyperplanes H_c). The other direction of the lemma is obvious. ■

Let us first consider binary dePCL-constraints. Figure 8.2 gives an algorithm for deciding whether a BICN is consistent. The algorithm BCN-Consistency propagates constraints locally

Algorithm BCN-CONSISTENCY
Input: A BCN N.
Output: YES or NO depending on whether N is consistent or not.

Method:
1. For $k, i, j = 1$ to n do
2. $\quad N_{ij} := N_{ij} \oplus (conv(N_{ik}) \otimes conv(N_{kj}))$
3. $\quad$ If $N_{ij} = \emptyset$ then Return NO
4. EndFor
5. Return YES

Figure 8.2: Deciding consistency of BCNs

in almost the same way as 3-consistency algorithms [Mac77]. But it does not take into account inequations while propagating a constraint (this is done by composing the convex hulls of the almost-convex intervals). This results in a network which is equivalent to the one obtained by running the modified Floyd-Warshall algorithm of [KL91] on $conv(N)$. Inequations are taken into account only during the intersection of the new constraints with the old ones. This is enough for deciding consistency as we show below.

Theorem 8.2.1 *Algorithm* BCN-CONSISTENCY *is correct i.e., it returns YES (resp. NO) iff the input network N is consistent (resp. inconsistent).*

Proof: Lemma 8.2.1 tells us that we can only have the following two kinds of inconsistency:

1. The inequalities of N are inconsistent. This kind of inconsistency is captured by BCN-CONSISTENCY since BCN-CONSISTENCY imitates algorithm Floyd-Warshall of [KL91] on $conv(N)$.

2. There exists an interval N_{ij} with a "hole" r and $x_j - x_i \neq r$ contradicts the inequalities in N. This is possible only if the equation $x_j - x_i = r$ is implied by the inequalities in N. Then, for some value of k, the interval $[r, r]$ is generated by composing $conv(N_{ik}) \otimes conv(N_{kj})$. This results in an empty interval N_{ij} when the intersection is computed. Thus this case is captured by BCN-CONSISTENCY as well.

■

Let us now consider the running time of algorithm BCN-CONSISTENCY. The parameter n denotes the number of nodes in the input network N. The parameter m denotes the maximum number of "holes" in any interval of N i.e., $m = \max_{1 \leq i,j \leq n} h_{ij}$.

Theorem 8.2.2 *Algorithm* BCN-CONSISTENCY *takes $O(n^3 m)$ time.*

Proof: The composition operation in line 2 of BCN-CONSISTENCY takes constant time and returns a convex interval. If the almost-convex interval N_{ij} is implemented as a pair

$$([-d_{ji}, d_{ij}], \{r_{ij}^1, \ldots, r_{ij}^{h_{ji}}\})$$

then the intersection operation can be done in $O(m)$ time. Therefore the running time of BCN-CONSISTENCY is $O(n^3 m)$. ■

Let us now turn to the full class of dePCL-constraints. The algorithm CONSISTENCY, shown in figure 8.3, detects whether a set of dePCL-constraints C is consistent.

Algorithm CONSISTENCY
Input: A set of constraints $C = C_e \cup C_i \cup C_d$ where C_e is a set of equalities, C_i is a set of inequalities and C_d is a set of disjunctions of inequations.

Output: YES or NO depending on whether C is consistent or not.

Method:
Step 1: Check whether $C_e \cup C_i$ is consistent
Let N be the BCN for $C_e \cup C_i$
For $k, i, j = 1$ to n do
    $N_{ij} := N_{ij} \oplus (N_{ik} \otimes N_{kj})$
    If $N_{ij} = \emptyset$ then Return NO
EndFor

Step 2: Check whether there exists $c \in C_d$ which contradicts $C_e \cup C_i$
For each constraint c in C_d do
    $Inc := true$
    For each equality $x_i - x_j = r$ in $\overline{c}$ do
        If $N_{ij} \neq [r, r]$ then
            $Inc := false$; Break
        EndIf
    EndFor
    If Inc then Return NO
EndFor
Return YES

Figure 8.3: Deciding consistency of a set of dePCL-constraints

Definition 8.2.1 The *size* of a dePCL-constraint is the number of disjuncts in its disjunctive normal form. For example, $||x_1 - x_2 \leq 3|| = 1$ and $||x_1 \neq x_2 \ \vee \ x_1 \neq x_3|| = 2$. The size of a constraint set is the sum of the sizes of its elements. The size of a constraint c (resp. a constraint set C) will be denoted by $||c||$ (resp. $||C||$).

The following theorem gives the complexity of algorithm CONSISTENCY.

Theorem 8.2.3 *Algorithm* CONSISTENCY *takes* $O(v(C_e \cup C_i)^3 + ||C_d||)$ *time.*

Proof: Step 1 takes $O(v(C_e \cup C_i)^3)$ time. Step 2 takes $O(||C_d||)$ time. ■

8.3 A Canonical Form for dePCL-Constraints

We will now present a canonical form for dePCL-constraints. The use of canonical forms in constraint database systems has been discussed in section 4.7.

Canonical forms for linear inequality constraints over the reals have been studied by Lassez and McAloon [LM89]. In this section we apply their methodology to the class of dePCL-constraints. A canonical form which shares some features with the one presented below has been used for qualitative BICN by [vB92, GS93]. For the more general class of dePCL-inequalities the role of a canonical form was played (implicitly) by the minimal network representation [DMP91, KL91].

The following definitions are essentially from [LM89].

Definition 8.3.1 A set of equalities is in *solved form* if it has the form

$$y_1 = t_1, \ldots, y_m = t_m$$

and the y_i's are all distinct variables which do not occur in the right hand side of any equation. The variables $y_1, \ldots, y_m$ are called the *eliminable* or *bound* variables. The remaining variables are called *parameters* of the solved form.

Definition 8.3.2 A constraint c is *redundant* in the set of constraints C if $Sol(C) = Sol(C - \{c\})$.

Definition 8.3.3 Let C be a set of inequalities. The inequality $x_i - x_j \leq r$ of C is called an *implicit equality* if $C \models x_i - x_j = r$.

Definition 8.3.4 A disjunction of inequations c is called C_i*-precise* (or simply *precise* when C_i is understood) if it is not implied by the inequalities in C_i, and the affine closure of $Sol(C_i) \cap Sol(\overline{c})$ is $Sol(\overline{c})$. The latter requirement implies that c is given in its smallest possible dimension [LM89].

Definition 8.3.5 A disjunction of inequations is in *solved form* if its complement is in solved form.

Definition 8.3.6 A set $C = C_e \cup C_i \cup C_d$ of constraints in variables $y_1, \ldots, y_m, x_1, \ldots, x_k$ is in *canonical form* if

- it does not contain redundant constraints
- C_e is a set of equalities in variables $y_1, \ldots, y_m, x_1, \ldots, x_k$ in solved form such that $y_1, \ldots, y_m$ are the bound variables and $x_1, \ldots, x_k$ are the parameters.
- C_i is a set of inequalities in variables $x_1, \ldots, x_k$ which does not contain any implicit equalities, and

Algorithm CANONICALFORM

Input: A set of dePCL-constraints $C = C_e \cup C_i \cup C_d$ where C_e is a set of equalities, C_i is a set of inequalities and C_d is a set of disjunctions of inequations.

Output: INCONSISTENT if C is inconsistent, otherwise a constraint set equivalent to C in canonical form.

Method:
Step 1: Detect possible inconsistencies in $C_e \cup C_i$.
$N := NET_m(C_e \cup C_i)$.
If N contains an inconsistency then
    Return INCONSISTENT
EndIf

Step 2: Find all implicit/explicit equalities of $C_e \cup C_i$ and return them in solved form.
$C_e^1 := Equalities(N)$.

Step 3: Eliminate all bound variables of C_e^1 from C_i and C_d.
$C_i^1 := \emptyset$; $C_d^1 := \emptyset$
For every equation $y = x + r$ in C_e^1 do
    For every $c \in C_i$ do
        If y appears in c then
            Substitute $x + r$ for y in c
            Simplify c and add it to C_i^1
        EndIf
    EndFor
    For every $c \in C_d$ do
        If y appears in c then
            Substitute $x + r$ for y in c; simplify c
            If c evaluates to $0 \neq 0$ then
                Return INCONSISTENT
            Else
                Add c to C_d^1
            EndIf
        EndIf
    EndFor
EndFor

Step 4: Remove redundant inequalities.
Call REDUNDANTINEQUALITIES(C_i^1)

Figure 8.4: The canonical form algorithm (steps 1-4)

Algorithm CANONICALFORM (continued)

Step 5: Determine which disjunctions of inequations are not implied by the inequalities in C_i^1. Transform these constraints into precise ones.

Initialize C_d^2 to $\emptyset$
<u>For</u> each $c \in C_d^1$ <u>do</u>
    Negate c to obtain a set of equalities E
    $N := NET_m(E \cup C_i^1)$
    <u>If</u> there is an inconsistency in N <u>then</u>
        Discard c (c is a tautology or it is already implied by C_i^1)
    <u>Else</u>
        $E' := Equalities(N)$
        Negate the equalities of E' to obtain a precise constraint c.
        Add c to C_d^2.
    <u>EndIf</u>
<u>EndFor</u>

Step 6: Remove redundancy from C_d^2.
Call REDUNDANTINEQUATIONS(C_d^2)

Step 7: Return the normal form of C.
Return (C_e^1, C_i^1, C_d^2)

Figure 8.5: The canonical form algorithm (steps 5 - 7)

- C_d is a set of C_i-precise disjunctions of inequations in variables $x_1, \ldots, x_k$ in solved form.

Figures 8.4 and 8.5 present an algorithm for deciding consistency and computing the canonical form of a set of dePCL-constraints. We now discuss this algorithm in detail and calculate its time complexity. Our results are presented without proofs. The interested reader can consult [LM89] for proofs for the case of generalized linear inequality constraints.

In step 1 CANONICALFORM discovers whether there is an inconsistency in the constraints $C_e \cup C_i$. This step takes $O(v(C_e \cup C_i)^3)$ time.

Step 2 computes the solved form of all explicit and implicit equalities in the constraints. This is done by algorithm EQUALITIES which is given in figure 8.6. EQUALITIES takes as input a BCN N and returns the set of equalities in $Constraints(N)$ in solved form. Variables which are related through equalities are partitioned into equivalence classes. These equivalence classes are stored as a collection of disjoint sets S. S can be implemented using linked lists. Each member of S is a list of variables sorted in ascending order of index. The first For statement creates the collection S. The call to algorithm *Insert* inserts x_i and x_j in the appropriate element (equivalence class) of S. *Insert* is not presented since it is straightforward. The second For statement generates the equations. From among the variables in an equivalence class, we always choose as a parameter the variable with the lowest index. This policy is necessary since we want equivalent sets of constraints to have identical canonical forms (theorem 8.3.2). If n is the number of variables in N then the first and the second For statements take time $O(n^3)$ and $O(n)$ respectively. Therefore step 2 takes time $O(v(C_e \cup C_i)^3)$ and generates at most $\left|C_e^1\right| = O(v(C_e \cup C_i))$ equalities.

Step 3 eliminates all bound variables of C_e^1 from $C_i \cup C_d$. This step takes

$$O(v(C_e \cup C_i)(|C_i| + |C_d| \, v(C_d)))$$

Algorithm EQUALITIES

Input: A binary constraint network N.
Output: The set of equalities in $Constraints(N)$ in solved form.

Method:
$S := \emptyset$
<u>For</u> each pair of nodes (x_i, x_j) of N such that $x_j > x_i$ <u>do</u>
    <u>If</u> arcs $x_j \rightarrow x_i$ and $x_i \rightarrow x_j$ are labeled with r <u>then</u>
        $Insert(\{x_i, x_j\}, S)$
    <u>EndIf</u>
<u>EndFor</u>

$C := \emptyset$
<u>For</u> each $s \in S$ <u>do</u>
    Let v_0 be the first element of s
    <u>For</u> each $v \in S$ such that $v \neq v_0$ <u>do</u>
        Let $v_0 \rightarrow v$ be labeled with r
        Add $v = v_0 + r$ to C
    <u>EndFor</u>
<u>EndFor</u>
Return C

Figure 8.6: The algorithm EQUALITIES

time. Step 4 calls algorithm REDUNDANTINEQUALITIES, shown in figure 8.7, to remove redundant inequalities. This step takes $O(|C_i|\, v(C_i)^3)$ time.

Step 5 rewrites each disjunction of inequations in its precise form in $O(|C_d|\,(v(C_i \cup C_d)^3)$ time. Step 6 calls algorithm REDUNDANTINEQUATIONS given in figure 8.8 to remove redundancy from disjunctions of inequations. Disjunctions of inequations can be treated independently of inequalities for questions of redundancy [LM89]. More formally, a precise disjunction of inequations $c_1 \in C_i \cup C_d$ is redundant if and only if there is another disjunction of inequations c_2 such that $c_2 \models c_1$. Step 6 takes $O(|C_d|^2\, v(C_d)^2)$ time.

We can now conclude that the algorithm takes

$$O(v(C)^3(|C_i| + |C_d|) + v(C_d)^2\, |C_d|^2)$$

time. Thus we have proved the following theorem.

Theorem 8.3.1 *Deciding consistency and computing the canonical form of a set of dePCL-constraints can be done in $O(v(C)^3(|C_i| + |C_d|) + v(C_d)^2\, |C_d|^2)$ time.*

Example 8.3.1 For the constraint set

$$\{x_6 - x_3 = 7,\ x_5 \leq x_1,\ x_1 \leq x_5,\ x_5 \leq x_3,\ x_3 \leq x_2,\ x_3 - x_2 \leq 4,\ x_1 \neq x_2\}$$

CANONICALFORM returns

$$\{x_5 = x_1,\ x_6 = x_3 + 7,\ x_1 - x_3 \leq 0,\ x_3 - x_2 \leq 0,\ x_2 - x_1 \neq 0\ \vee\ x_3 - x_1 \neq 0\}.$$

Algorithm REDUNDANTINEQUALITIES

Input: A set of inequalities C.
Output: A set of irredundant inequalities equivalent to C.

Method:
<u>For</u> each constraint c in C <u>do</u>
    <u>If</u> c is a tautology <u>then</u>
        Delete c
    <u>ElseIf</u> c is $x_i - x_j \leq r$ <u>then</u>
        <u>If</u> there is another constraint $c' :\ x_i - x_j \leq r'$ in C and $r' \leq r$ <u>then</u>
            Delete c
        <u>Else</u>
            $N := NET_m(C - \{c\})$
            <u>If</u> the constraints between x_i and x_j in N imply c <u>then</u>
                Delete c
            <u>EndIf</u>
        <u>EndIf</u>
    <u>EndIf</u>
<u>EndFor</u>
Return C

Figure 8.7: Removing redundant inequalities

Algorithm REDUNDANTINEQUATIONS

Input: A set of disjunctions of inequations C.
Output: A set of irredundant disjunctions of inequations equivalent to C.

Method:
<u>For</u> each constraint c_i in C <u>do</u>
    <u>For</u> each constraint $c_j \neq c_i$ in C <u>do</u>
        Find whether $\overline{c_j} \models \overline{c_i}$ as follows
        <u>For</u> each variable x_i of $\overline{c_i}$ <u>do</u>
            <u>If</u> x_i is a bound variable of $\overline{c_j}$ (i.e., if $x_i = x_k + r_{ik}$ is a conjunct of $\overline{c_j}$) <u>then</u>
                Substitute $x_k + r_{ik}$ for x_i in $\overline{c_i}$
            <u>EndIf</u>
        <u>EndFor</u>
        <u>If</u> $\overline{c_i}$ evaluates to $0 = 0$ <u>then</u>
            Delete c_i
            Break
        <u>EndIf</u>
    <u>EndFor</u>
<u>EndFor</u>

Figure 8.8: Removing redundant disjunctions of inequations

The following uniqueness theorem is a consequence of [LM89]. One of the observations underlying the proof of the theorem is that there is a fixed order of variables and this order is taken into account consistently when equations or inequations are transformed into solved form (e.g., in algorithm EQUALITIES the variable with the lowest index is always chosen to be a parameter).

Theorem 8.3.2 *For equivalent sets of constraints in the same set of variables, the algorithm* CANONICALFORM *outputs identical canonical forms.*

8.4 Variable Elimination for dePCL-constraints

For dePCL-constraints, Fourier's algorithm can be extended to cope with disjunctions of inequations. The extension is based on the following observation. Consider the constraints $x \leq r_u, x \geq r_l$ and $x \neq r \ \vee \ \phi$ where ϕ is a disjunction of inequations or *false*. Without loss of generality we can assume that x does not appear in ϕ.[3] Let us eliminate x from the first two constraints to obtain $r_l \leq r_u$. Now consider a solution v of $r_l \leq r_u$. If v satisfies ϕ then it can be extended to a solution of the original constraints. If v does not satisfy ϕ then it can be extended to a solution of the original constraints if and only if r, r_l and r_u do not share the same value. As a result, the constraint $\neg\phi \ \supset \ r \neq r_u \ \vee \ r \neq r_l$ or equivalently $\phi \ \vee \ r \neq r_u \ \vee \ r \neq r_l$ must be included in the resulting set of constraints not involving x. Note that this constraint is also a consequence of the original constraints. Therefore, if the variable to be eliminated is x, our algorithm reduces to the following statement: For all triples $x \leq r_u, x \geq r_l$ and $x \neq r \ \vee \ \phi$ of input constraints, add $r_l \leq r_u$ and $\phi \ \vee \ r \neq r_u \ \vee \ r \neq r_l$ in the output constraints. Thus as in lemma 8.2.1, disjunctions of inequations can be processed *independently* of one another when variable elimination is performed.

Example 8.4.1 Let us assume we want to eliminate variable x_1 from the set of constraints

$$x_3 \leq x_1, \ x_5 \leq x_1, \ x_1 \leq x_2, \ x_5 \neq x_1, \ x_4 \neq x_1.$$

Eliminating x_1 gives the inequalities

$$x_3 \leq x_2, \ x_5 \leq x_2$$

and the disjunctions of inequations

$$x_5 \neq x_3 \ \vee \ x_5 \neq x_2, \ x_5 \neq x_5 \ \vee \ x_5 \neq x_2, \ x_4 \neq x_3 \ \vee \ x_4 \neq x_2, \ x_4 \neq x_5 \ \vee \ x_4 \neq x_2.$$

The latter set is equivalent to

$$x_5 \neq x_2, \ x_4 \neq x_3 \ \vee \ x_4 \neq x_2.$$

Figure 8.9 gives our variable elimination algorithm. For simplicity, the algorithm assumes that one-variable inequalities are given as two-variable inequalities (with the introduction of an auxiliary variable x_0 which is understood to be equal to zero). The same algorithm can be applied to the case of generalized linear constraints [LM89]. Independently this algorithm has also been proposed in [Imb93]. The following proposition shows that the algorithm is correct. For the case of generalized linear constraints the proof is similar [Imb93].

[3] The constraint $x \neq r \ \vee \ \phi$ is equivalent to $\neg(x = r \ \wedge \ \neg\phi)$ where $\neg\phi$ is a conjunction of equalities. If we substitute r for x in $\neg\phi$, we obtain an equivalent constraint.

Algorithm VarElimination

Input: A consistent constraint set $C = C_e \cup C_i \cup C_d$, in variables X, where C_e is a set of equalities, C_i is a set of inequalities and C_d is a set of disjunctions of inequations. A set of variables $E \subseteq X$ to be eliminated from C.

Output: A set of constraints C' in variables $X \setminus E$ such that $Sol(C') = Proj_{X \setminus E}(Sol(C))$.

Method:
$C'_e := \emptyset;\ C'_i := \emptyset;\ C'_d := \emptyset$
For each variable $x \in E$ do
  If x is the bound variable of equation $e_i \in C_e$ then
    $C_e := C_e - \{e_i\}$
  ElseIf x is a parameter in equation $c_i :\ y_i = x + r$ of C_e then
    For each constraint c in $C_e - \{c_i\}$ do
      $C'_e := C'_e \cup \{c[y_i - r/x]\}$
    EndFor
    For each constraint c in C_i do
      $C'_i := C'_i \cup \{c[y_i - r/x]\}$
    EndFor
    For each constraint c in C_d do
      $C'_d := C'_d \cup \{c[y_i - r/x]\}$
    EndFor
    $C_e := C'_e;\ C_i := C'_i;\ C_d := C'_d;\ C'_e := \emptyset;\ C'_i := \emptyset;\ C'_d := \emptyset$
  Else (the variable x appears only in $C_i \cup C_d$)
    For each constraint $x_k - x \leq (<) r_k$ in C_i do
      For each constraint $x - x_j \leq (<) r_j$ in C_i $(j \neq k)$ do
        $C'_i := C'_i \cup \{x_k - x_j \leq (<) r_k + r_j\}$
        For each $x - x_s \neq r_s \ \vee\ \phi$ in C_d (s.t. ϕ does not involve x) do
          $C'_d := C'_d \cup \{x_k - x_s \neq r_k + r_s \ \vee\ x_j - x_s \neq r_s - r_j \ \vee\ \phi\}$
        EndFor
      EndFor
    EndFor
    Add all constraints of C_i which do not contain x to C'_i
    Add all constraints of C_d which do not contain x to C'_d
    $C_e := C'_e;\ C_i := C'_i;\ C_d := C'_d$
    $C'_e := \emptyset;\ C'_i := \emptyset;\ C'_d := \emptyset$
  EndIf
EndFor
Output $C_e \cup C_i \cup C_d$

Figure 8.9: Variable Elimination for dePCL-constraints

Theorem 8.4.1 *Let C be a set of constraints in variables $x_1, x_2, \ldots, x_n$ and C' be the set of constraints produced by algorithm* VarElimination *when x_1 is eliminated. If $(x_1^0, x_2^0, \ldots, x_n^0)$ is a solution of C then $(x_2^0, x_3^0, \ldots, x_n^0)$ is a solution of C'. Conversely, every solution $(x_2^0, x_3^0, \ldots, x_n^0)$ of C' can be extended to a solution $(x_1^0, x_2^0, \ldots, x_n^0)$ of C.*

Proof: The proof is straightforward when x_1 appears in an equality of C_e. We will therefore limit our attention to the case when X_1 appears only in $C_i \cup C_d$.

If part. Assume $(x_1^0, x_2^0, \ldots, x_n^0)$ is a solution of C. It is obvious that $(x_2^0, x_3^0, \ldots, x_n^0)$ satisfies all constraints of C' which also belong to C. Let us then turn to inequalities of C' which do not belong to C i.e., they have been generated by the algorithm. For each pair of inequalities $x_k - r_{k1} \leq x_1 \leq x_j + r_{1j}$ of C considered by the algorithm, the inequality $x_k - x_j \leq r_{k1} + r_{1j}$ is added to C'. But since $x_k^0 - r_{k1} \leq x_1^0 \leq x_j^0 + r_{1j}$ holds, $x_k^0 - x_j^0 \leq r_{k1} + r_{1j}$ holds as well. Consequently $(x_2^0, x_3^0, \ldots, x_n^0)$ satisfies all the inequality constraints of C'.

Let us now turn to disjunctions of inequations. When VarElimination considers constraint $x_1 - x_l \neq q_{1l} \ \vee \ \phi$ (ϕ does not contain x_1) and inequalities $c: \ x_k - r_{k1} \leq x_1 \leq x_j + r_{1j}$, the constraint c' is generated:

$$c': \ x_k - r_{k1} - x_l \neq q_{1l} \ \vee \ x_j + r_{1j} - x_l \neq q_{1l} \ \vee \ \phi$$

If $(x_1^0, x_2^0, \ldots, x_n^0)$ satisfies ϕ then $(x_2^0, x_3^0, \ldots, x_n^0)$ satisfies c'. If $(x_1^0, x_2^0, \ldots, x_n^0)$ does not satisfy ϕ then $x_1^0 - x_l^0 \neq q_{1l}$. Let us now assume that $x_k^0 - r_{k1} - x_l^0 = q_{1l}$ and $x_j^0 + r_{1j} - x_l^0 = q_{1l}$ (proof by contradiction). Then $x_k^0 - r_{k1} = x_l^0 + q_{1l}$ and $x_j^0 + r_{1j} = x_l^0 + q_{1l}$, which together with c imply $x_1^0 = x_l^0 + q_{1l}$. But this contradicts the fact $x_1^0 - x_l^0 \neq q_{1l}$. Thus one of the equalities assumed true does not hold, therefore c' is satisfied by $(x_2^0, x_3^0, \ldots, x_n^0)$. Consequently, all disjunctions of inequations of C' are satisfied by $(x_2^0, x_3^0, \ldots, x_n^0)$.

Only if part. Case 1: C_i contains at least one pair of inequalities where x_1 appears with different signs.

Let us assume that $(x_2^0, x_3^0, \ldots, x_n^0)$ is a solution of C'. Obviously, $(x_2^0, x_3^0, \ldots, x_n^0)$ satisfies the constraints of C which do not involve x_1. Let us now consider the remaining inequalities of C:

$$x_1 - x_j \leq r_{1j}, \ j = 1, \ldots, p \ \text{ and } \ x_k - x_1 \leq r_{k1}, \ k = 1, \ldots, t$$

In order to extend $(x_2^0, x_3^0, \ldots, x_n^0)$ to a solution of these inequalities, we need to find a rational number x_1^0 such that

$$x_1^0 - x_j^0 \leq r_{1j}, \ j = 1, \ldots, p \ \text{ and } \ x_k^0 - x_1^0 \leq r_{k1}, \ k = 1, \ldots, t$$

or equivalently

$$x_1^0 \leq x_j^0 + r_{1j}, \ j = 1, \ldots, p \ \text{ and } \ x_1^0 \geq x_k^0 - r_{k1}, \ k = 1, \ldots, t.$$

Let us assume that

$$min_{1 \leq j \leq p} \ x_j^0 + r_{1j} = x_{j_m}^0 + r_{1j_m} \ \text{ and } \ max_{1 \leq k \leq t} \ x_k^0 - r_{k1} = x_{k_m}^0 + r_{k_m 1}.$$

Thus x_1^0 must be such that $x_{k_m}^0 - r_{k_m 1} \leq x_1^0 \leq x_{j_m}^0 + r_{1j_m}$. But such a value x_1^0 exists since $x_{k_m}^0 - x_{j_m}^0 \leq r_{k_m 1} + r_{1j_m}$ is true (this holds because $x_{k_m} - x_{j_m} \leq r_{k_m 1} + r_{1j_m}$ is the inequality added to C' for $j = j_m$ and $k = k_m$ and $(x_2^0, x_3^0, \ldots, x_n^0)$ is a solution of C').

Now let us turn to disjunctions of inequations of the form $x_1 - x_v \neq q_{1v} \ \vee \ \phi$. Let us substitute the values of $x_2^0, x_3^0, \ldots, x_n^0$ for $x_2, x_3, \ldots, x_n$ in these constraints. Each constraint such that $(x_2^0, x_3^0, \ldots, x_n^0)$ satisfies ϕ will evaluate to $true$. Let us assume that the remaining constraints evaluate to $x_1 \neq x_{v_l}^0 + q_{1v_l}$, $l = 1, \ldots, s$. The value of x_1 chosen above (i.e., x_1^0) must also satisfy these constraints. We can always find such a value unless

$$x_{k_m}^0 - r_{k_m1} = x_{j_m}^0 + r_{1j_m} = x_{v_{l_w}}^0 + q_{1v_{l_w}},\ 1 \leq l_w \leq s.$$

But this is not possible since C' contains a constraint

$$x_{k_m} - r_{k_m1} \neq x_{v_{l_w}} + q_{1v_{l_w}} \ \vee\ x_{j_m}^0 + r_{1j_m} \neq x_{v_{l_w}} + q_{1v_{l_w}} \ \vee\ \phi$$

which is satisfied by $(x_2^0, x_3^0, \ldots, x_n^0)$ while ϕ is not satisfied. In other words, each solution of C' can always be extended to a solution of C.

Case 2: C_i contains only inequalities where x_1 appears with a positive or negative sign. We prove the result only for the former case. The proof for the latter case is similar.

Let us assume that $(x_2^0, x_3^0, \ldots, x_n^0)$ is a solution of C'. Obviously, $(x_2^0, x_3^0, \ldots, x_n^0)$ satisfies the constraints of C which do not involve x_1. Let us now consider the remaining constraints of C:

$$x_1 - x_j \leq r_{1j},\ j = 1, \ldots, p \ \text{ and } \ x_1 - x_v \neq q_{1v} \ \vee\ \phi,\ v = 1, \ldots, s.$$

Thus we have to find a value x_1^0 such that

$$x_1^0 \leq x_j^0 + r_{1j},\ j = 1, \ldots, p \ \text{ and } \ x_1^0 - x_v^0 \neq q_{1v} \ \vee\ \phi,\ v = 1, \ldots, s.$$

But such a value always exists. ■

Although the input to VarElimination is in canonical form, it is possible that its output is not. This can happen only if an eliminated variable is a parameter which appears only in C_i. When this variable is eliminated, C_i' does not contain implicit equalities (see [LHM89] for a proof of this) but may contain redundant constraints. It is also possible that the constraints in C_d' are not precise or irredundant.

Example 8.4.2 Let us assume that variable x_1 is eliminated from the set of constraints

$$\{x_2 \leq x_1,\ x_1 \leq x_3,\ x_2 \leq x_5,\ x_5 \leq x_3\}$$

using algorithm VarElimination. The resulting set of constraints

$$\{x_2 \leq x_3,\ x_2 \leq x_5,\ x_5 \leq x_3\}$$

and obviously $x_2 \leq x_3$ is redundant.

Example 8.4.3 Let us assume that variable x_2 is eliminated from the set of constraints

$$\{x_1 \leq x_2,\ x_2 \leq x_3,\ x_5 \leq x_1,\ x_5 \leq x_3\}$$

using algorithm VarElimination. The resulting set of constraints

$$\{x_1 \leq x_3,\ x_5 \leq x_1,\ x_5 \leq x_3\}$$

and obviously $x_5 \leq x_3$ is redundant.

Example 8.4.4 Let us assume that variable x_2 is eliminated from the set of constraints

$$\{x_3 - x_2 \leq 2,\ x_2 - x_1 \leq 3,\ x_7 \geq 5,\ x_3 \leq 0,\ x_2 \neq x_7\}.$$

The resulting constraint set is

$$\{x_3 - x_1 \leq 5,\ x_7 \geq 5,\ x_3 \leq 0,\ x_3 - x_7 \neq 2\ \vee\ x_1 - x_7 \neq -3\}.$$

The inequation $x_3 - x_7 \neq 2$ is implied by the inequality constraints.

Consequently, if the output of VarElimination must be returned in canonical form, redundancy must be eliminated from C_i', each $c \in C_d'$ must be made precise and finally redundancy must be eliminated from C_d' (in this order!). Fortunately, in the remaining cases the output of VarElimination is in canonical form as it is demonstrated by the following theorem.

Theorem 8.4.2 *Let the input to algorithm* VarElimination *be $C_e \cup C_i \cup C_d$ and E. If all variables of E are bound or parameter variables of the input set C_e then the output sets C_i and C_d are irredundant. Moreover, the former set does not contain implicit equalities and the latter consists only of precise constraints.*

Proof: Assume that x is the eliminated variable and $y_i = x + r$ is an equation of C_e where x appears. In the new set of equations C_e', y_i becomes a parameter and C_e' is in solved form. In addition, C_i' does not contain any implicit equalities. For the sake of the argument, let us assume the contrary. If C_i' contains an implicit equality, application of Fourier's elimination algorithm produces an inequality $0 \leq 0$ as a linear combination of constraints of C_i' [LHM89]. If y_i does not appear in these constraints then we have a contradiction ($0 \leq 0$ is obtained as a linear combination of constraints in C_i therefore C_i contains an implicit equality). So let us assume that y_i does appear in these constraints. Let us rewrite any constraint $x_i - y_i \leq r_i$ as $x_i - (y_i - r) \leq r_i + r$, and $y_i - x_j \leq r_j$ as $(y_i - r) - x_j \leq r_j - r$. Then substitute x for $y_i - r$ in these constraints. Now we are left with a linear combination of constraints of C_i which produces $0 \leq 0$ if all variables are eliminated by Fourier's algorithm. Thus we have a contradiction again therefore C_i' does not contain any implicit equalities.

We will now show that C_i' contains no redundant inequalities. Let us assume the contrary e.g., that the constraint $c:\ t_i - t_j \leq r_{ij}$ is redundant. In other words, $C_i' - \{t_i - t_j \leq r_{ij}\} \models t_i - t_j \leq r_{ij}$ which is equivalent to saying that the set $C_i' - \{t_i - t_j \leq r_{ij}\} \cup \{t_i - t_j > r_{ij}\}$ is inconsistent. Therefore this set contains a sequence of inequalities

$$t_1 \leq t_2 + r_{12},\ t_2 \leq t_3 + r_{23},\ \ldots,\ t_j - t_i < -r_{ij},\ \ldots,\ t_{n-1} \leq t_n + r_{n-1,n}$$

such that $t_1, \ldots, t_n$ are different variables, $t_n = t_1$ and $r_{12} + r_{23} + \cdots + r_{n-1,n} \leq 0$ [Sho81]. If y_i is not one of $t_1, \ldots, t_n$ then we have a contradiction since all the above inequalities belong to C_i, therefore C_i contains a redundant constraint. So let us assume that y_i is one of $t_1, \ldots, t_n$ and $y_i \leq x_k + r_{ik},\ x_l \leq y_i + r_{li}$ are the inequalities where it appears (if these inequalities are successive then $k = l$). Let us rewrite these equalities as

$$(y_i - r) \leq x_k + (r_{ik} - r),\ x_l \leq (y_i - r) + (r_{li} + r)$$

and then substitute x for $y_i - r$. Since all these inequalities belong to C_i, C_i contains a redundant inequality. Thus we have a contradiction again. For the case of C_d the proof is similar. ■

Remark 8.4.1 In the rest of this section, we assume that whenever a variable is eliminated, using VARELIMINATION, trivially redundant or syntactically redundant constraints are not added to C'_e or C'_i.

Let us now investigate the complexity of VARELIMINATION.

Theorem 8.4.3 *Let $C = C_e \cup C_i \cup C_d$ be a set of dePCL-constraints. If m variables appearing in C_i are eliminated using algorithm* VARELIMINATION *and the resulting set is $C' = C'_e \cup C'_i \cup C'_d$ then $|C'_e| \leq |C_e|$, $|C'_i| \leq 2(v(C_i) - m)^2$ and $|C'_d| \leq |C_d|\,(v(C_i) - 1)^{2m}$.*

Proof: When m variables are eliminated we are left with $v(C) - m$ variables. It easy to see that $|C'_e| \leq |C_e|$. Lemma 3.5.1 implies $|C_i| \leq 2(v(C_i) - m)^2$.

For the second inequality, let C_i^l (resp. C_d^l) be the set of inequalities (resp. disjunctions of inequations) produced by VARELIMINATION when the l-th variable is eliminated ($1 \leq l \leq m$). Therefore

$$\left|C_d^1\right| \leq |C_d|\,(v(C_i) - 1)^2, \quad \left|C_d^2\right| \leq \left|C_d^1\right|(v(C_i) - 2)^2 \leq |C_d|\,(v(C_i) - 1)^2(v(C_i) - 2)^2$$

and so on. Finally, $|C_d^m| \leq |C_d| \prod_{l=1}^m (v(C_i) - l)^2 \leq |C_d|\,(v(C_i) - 1)^{2m}$. ■

Corollary 8.4.1 *The algorithm* VARELIMINATION *takes time polynonial in the size of the input constraint set and exponential in the number of eliminated variables.*

This exponential time complexity is only due to the fact that the size of the output constraint set can be exponential in the number of eliminated variables. The following proposition complements the corollary.

Proposition 8.4.1 *For every m, k such that $k \geq 4m$ we can find a set of dePCL-constraints C of size $\|C\| \leq k$ and m variables of C such that when these variables are eliminated the resulting constraint set cannot be represented by fewer than 2^m disjunctions of inequations and $2m$ inequalities.*

Proof: Given m, k such that $k \geq 4m$, the set C in variables $L_1, \ldots, L_m,\ U_1^1, \ldots, U_m^1$, $U_1^2, \ldots, U_m^2, X_1, \ldots, X_m,\ Y_1, \ldots, Y_m$ is $C = \bigcup_{1 \leq i \leq m} C_i \cup C_d$ where

$$C_i = \{L_i \leq X_i,\ X_i \leq U_i^1,\ X_i \leq U_i^2\} \text{ and } C_d = \{\bigvee_{i=1}^m Y_i \neq X_i\}.$$

Let us now eliminate variables $X_1, \ldots, X_m$ to arrive at $C' = \bigcup_{1 \leq i \leq m} C'_i \cup C'_d$ where

$$C'_i = \{L_i \leq U_i^1,\ L_i \leq U_i^2\} \text{ and } C_d = \{\bigvee_{i=1}^m Y_i \neq L_i \ \vee\ Y_i \neq U_i^{j_i} :\ j_i = 1, 2\}.$$

The canonical form of C' is $C'' = \bigcup_{1 \leq i \leq m} C'_i \cup C''_d$ where

$$C''_d = \{\bigvee_{i=1}^m Y_i \neq L_i \ \vee\ U_i^{j_i} \neq L_i :\ j_i = 1, 2\}$$

if the order of the variables is the one given above. Obviously C' (and C'') contains 2^m disjunctions of inequations and $2m$ inequalities. Now assume that there exists a set of constraints A such that

$A \neq C'$, $Sol(C') = Sol(A)$ and $|A| < 2^m + 2m$ (proof by contradiction). If A' is the canonical form of A then $|A'| \leq |A|$. But equivalent sets of constraints have identical canonical forms therefore $A' = C''$ thus $|A| \geq 2^m + 2m$ which contradicts the assumption $|A| < 2^m + 2m$. ■

The following chapter demonstrates an important class of dePCL-constraints for which the above property no longer holds i.e., eliminating any number of variables results in an output set which can be described with a polynomial number of constraints.

8.5 Conclusion

In this chapter we extended the class of dePCL-constraints to include inequations and disjunctions of inequations. We developed algorithms for deciding consistency, computing a canonical form and performing variable elimination for dePCL-constraints. In the following chapter we turn to another important problem: enforcing global consistency of dePCL-constraints.

Chapter 9

Global Consistency of dePCL-Constraints

One of the most important notions found in the constraint satisfaction literature is global consistency [Fre78]. In a *globally consistent* constraint set all interesting constraints are explicitly represented. In other words, the projection of the solution set on any subset of the variables can be computed by simply collecting the constraints involving these variables. An important consequence of this property is that a solution can be found by *backtrack-free* search [Fre82].

Enforcing global consistency can take an exponential amount of time in the worst case [Fre78, Coo90]. As a result it is very important to identify cases in which *local consistency*, which presumably can be enforced in polynomial time, implies global consistency [Dec92].

In this chapter we study the problem of enforcing global consistency in sets of dePCL-constraints which contain inequations but *not* disjunctions of inequations. In other words, we study global consistency for the BCN defined in section 8.1. Our contributions can be summarized as follows:

- We identify the level of local consistency which is necessary and sufficient for achieving global consistency of a BCN (corollary 9.1.1). This result has the following interesting consequence which complements proposition 8.4.1. For the class of dePCL-constraints which can be represented by a BCN, the output of all variable elimininination operations can be described by a set of dePCL-constraints whose size is polynomial.
- We develop detailed algorithms for the global consistency problem in BCN (sections 9.1 and 9.2).
- Finally, we present an algorithm for achieving global consistency in the point algebra networks of [VK86]. This problem has been erroneously claimed solved in [vBC90].

This chapter is organized as follows. In section 9.1 we present an algorithm for enforcing global consistency of BCN. In section 9.2 we improve the algorithm of section 9.1 and show how part of it can be used for computing the minimal network equivalent to a BCN. Finally, in section 9.3 we solve the consistency problem for point algebra networks.

9.1 Global Consistency of a BCN

We will first consider enforcing global consistency in a BCN for dePCL-constraints.

Notation 9.1.1 Let C be a set of constraints in variables $x_1, \ldots, x_n$. For any i such that $1 \leq i \leq n$, $C(x_1, \ldots, x_i)$ will denote the set of constraints in C involving only variables $x_1, \ldots, x_i$.

The following definition is from [Dec92].

Definition 9.1.1 Let C be a set of constraints in variables $x_1, \ldots, x_n$ and $1 \leq i \leq n$. C is called *i-consistent* iff for every $i-1$ distinct variables $x_1, \ldots, x_{i-1}$, every valuation $u = \{x_1 \leftarrow x_1^0, \ldots, x_{i-1} \leftarrow x_{i-1}^0\}$ such that u satisfies the constraints $C(x_1, \ldots, x_{i-1})$ and every variable x_i different from $x_1, \ldots, x_{i-1}$, there exists a rational number x_i^0 such that u can be extended to a valuation $u' = u \cup \{x_i \leftarrow x_i^0\}$ which satisfies the constraints $C(x_1, \ldots, x_{i-1}, x_i)$. C is called *strong i-consistent* if it is j-consistent for every j, $1 \leq j \leq i$. C is called *globally consistent* iff it is i-consistent for every i, $1 \leq i \leq n$.

Let us present some examples illustrating the above definitions.

Example 9.1.1 The constraint set $C = \{x_2 - x_1 \leq 5,\ x_1 - x_3 \leq 2,\ x_5 - x_4 \leq 1,\ x_4 - x_6 \leq 3\}$ is 1- and 2-consistent but not 3-consistent. For example, the valuation $v = \{x_2 \leftarrow 10,\ x_3 \leftarrow 2\}$ satisfies $C(x_2, x_3) = \emptyset$ but it cannot be extended to a valuation which satisfies C.

We can enforce 3-consistency by adding the constraints $x_2 - x_3 \leq 7$ and $x_5 - x_6 \leq 4$ to C. The resulting set is 3-consistent and also globally consistent.

Example 9.1.2 The constraint set $C = \{x_2 - x_1 = 5,\ x_1 - x_4 \neq 1\}$ is 1- and 2-consistent but not 3-consistent. For example, the valuation $v = \{x_2 \leftarrow 6,\ x_4 \leftarrow 0\}$ satisfies $C(x_2, x_4) = \emptyset$ but it cannot be extended to a valuation which satisfies C.

We can enforce 3-consistency by adding the constraint $x_2 - x_4 \neq 6$ to C. The resulting set is 3-consistent and also globally consistent.

Example 9.1.3 The constraint set $C = \{x_2 - x_1 \leq 5,\ x_1 - x_3 \leq 2,\ x_2 - x_3 \leq 7,\ x_1 - x_4 \neq 1\}$ is strong 3-consistent but not 4-consistent. For example, the valuation $v = \{x_2 \leftarrow 7,\ x_3 \leftarrow 0,\ x_4 \leftarrow 1\}$ satisfies $C(x_2, x_3, x_4) = \{x_2 - x_3 \leq 7\}$ but it cannot be extended to a valuation which satisfies C.

Enforcing 4-consistency amounts to adding the *disjunction*

$$x_2 - x_4 \neq 6 \ \vee \ x_3 - x_4 \neq -1.$$

The resulting set is 4-consistent and also globally consistent.

Example 9.1.4 The constraint set

$$C = \{x_2 - x_1 \leq 5,\ x_1 - x_3 \leq 2,\ x_2 - x_3 \leq 7,\ x_5 - x_4 \leq 1,\ x_4 - x_6 \leq 3,\ x_5 - x_6 \leq 4,\ x_1 - x_4 \neq 1\}$$

is strong 3-consistent but not 4-consistent. Adding the constraint $x_2 - x_4 \neq 6 \vee x_3 - x_4 \neq -1$ (as in the previous example) is not enough. For example, the valuation $v = \{x_5 \leftarrow 2,\ x_6 \leftarrow -2,\ x_1 \leftarrow 2\}$ satisfies $C(x_5, x_6, x_1) = \{x_5 - x_6 \leq 4\}$ but it cannot be extended to a valuation which satisfies $C(x_5, x_6, x_1, x_4)$.

We can enforce 4-consistency by also adding the constraint $x_5 - x_1 \neq 0 \ \vee \ x_6 - x_1 \neq -4$ to C. Let the resulting set be C'. C' is strong 4-consistent but not 5-consistent. For example, the valuation $v = \{x_2 \leftarrow 7,\ x_3 \leftarrow 0,\ x_5 \leftarrow 2,\ x_6 \leftarrow -2\}$ satisfies $C(x_2, x_3, x_5, x_6) = \{x_2 - x_3 \leq 7,\ x_5 - x_6 \leq 4\}$ but it cannot be extended to a valuation which satisfies $C(x_2, x_3, x_5, x_6, x_1)$ (or $C(x_2, x_3, x_5, x_6, x_4)$).

We can enforce 5-consistency by adding the constraint

$$x_2 - x_3 \neq 7 \ \vee \ x_5 - x_6 \neq 4 \ \vee \ x_2 - x_5 \neq 5$$

to C'. The resulting constraint set is strong 5-consistent and also globally consistent.

Figure 9.1 presents algorithm BCN-GConsistency which enforces global consistency on its input BCN. BCN-GConsistency takes as input an BCN and returns an equivalent set of temporal constraints which is globally consistent. BCN-GConsistency's output is not a BCN because, as the above examples indicate, enforcing global consistency might result in the introduction of disjunctions of inequations which cannot be represented by a BCN.

The algorithm BCN-GConsistency essentially enforces strong 5-consistency on its input network N. As we will show shortly, this level of local consistency is enough for achieving global consistency. In step 1, BCN-GConsistency enforces strong 3-consistency on its input. This is achieved in two passes. In the first pass ($pass = 1$), BCN-GConsistency runs the modified Floyd-Warshall algorithm of [KL91] on $conv(N)$. Let N' denote the resulting BCN and $A = Constraints(N')$. Then $conv(N')$ is minimal and globally consistent [DMP91, KL91]. In the second pass ($pass = 2$), BCN-GConsistency simply propagates inequations in N'.

In step 2, BCN-GConsistency enforces 4- and 5-consistency. For each $r^g_{ik} \in holes(N'_{ki})$ (or equivalently for each inequation $x_i - x_k \neq r^g_{ik}$ of A), BCN-GConsistency explores the inequalities of A involving x_i and x_k in the following systematic way. Figure 9.2 illustrates the structure of the subnetworks of N' explored in this step. Edges labeled with $\neq$ denote non-convex intervals.

1. If there are inequalities $x_m - x_i \leq d_{im}$ and $x_i - x_l \leq d_{li}$ then step 2.1 ensures that any valuation $v = \{x_l \leftarrow x^0_l, x_m \leftarrow x^0_m, x_k \leftarrow x^0_k\}$, which satisfies $A(x_l, x_m, x_k)$, can be extended to a valuation $v' = v \cup \{x_i \leftarrow x^0_i\}$ which satisfies $A(x_l, x_m, x_k, x_i)$. This is achieved with the introduction of the inequation constraint

   $$x_m - x_k \neq d_{im} + r^g_{ik} \ \vee \ x_l - x_k \neq -d_{li} + r^g_{ik}.$$

   If there are inequalities $x_s - x_k \leq d_{ks}$ and $x_k - x_t \leq d_{tk}$ then step 2.1 also ensures that any valuation $v = \{x_s \leftarrow x^0_s, x_t \leftarrow x^0_t, x_i \leftarrow x^0_i\}$, which satisfies $A(x_s, x_t, x_i)$, can be extended to a valuation $v' = v \cup \{x_k \leftarrow x^0_k\}$ which satisfies $A(x_s, x_t, x_i, x_k)$. This is achieved with the introduction of the inequation constraint

   $$x_s - x_i \neq d_{ks} - r^g_{ik} \ \vee \ x_t - x_i \neq -d_{tk} - r^g_{ik}.$$

2. If there are inequalities $x_m - x_i \leq d_{im}$, $x_i - x_l \leq d_{li}$, $x_s - x_k \leq d_{ks}$ and $x_k - x_t \leq d_{tk}$ then step 2.2 ensures that any valuation $v = \{x_l \leftarrow x^0_l, x_m \leftarrow x^0_m, x_s \leftarrow x^0_s, x_t \leftarrow x^0_t\}$, which satisfies $A(x_l, x_m, x_s, x_t)$, can be extended to a valuation $v' = v \cup \{x_i \leftarrow x^0_i, x_k \leftarrow x^0_k\}$ which satisfies $A(x_l, x_m, x_s, x_t, x_i, x_k)$. This is achieved with the introduction of the inequation constraint

   $$x_m - x_l \neq d_{im} + d_{li} \ \vee \ x_s - x_t \neq d_{ks} + d_{tk} \ \vee \ x_m - x_s \neq r^g_{ik} + d_{im} - d_{ks}.$$

Warning. It is possible that step 2 of algorithm BCN-GConsistency introduces constraints that are not strictly necessary for enforcing global consistency. In this section, we tolerate this inefficiency so that we can concentrate on the main ideas, but also minimize the case analysis in the forthcoming proofs. In section 9.2, we will present an improved version of BCN-GConsistency.

The following lemma and theorem demonstrate the correctness of algorithm BCN-GConsistency. The proofs are rather long but easy to follow.

Lemma 9.1.1 *The first step of algorithm* BCN-GConsistency *enforces strong 3-consistency on the input network N.*

Algorithm BCN-GConsistency
Input: A consistent BCN N.
Output: A globally consistent set of constraints equivalent to N.

Method:

```
Step 1: Enforce strong 3-consistency.
For pass = 1 to 2 do
    For k, i, j = 1 to n do
        If pass = 1 then
            N_ij := N_ij ⊕ (conv(N_ik) ⊗ conv(N_kj))
        Else
            N_ij := N_ij ⊕ (N_ik ⊗ N_kj)
        EndIf
    EndFor
EndFor

Step 2: Enforce 4- and 5-consistency.
C := ∅
For i, k = 1 to n do
    For g = 1 to h_ik do

        Step 2.1
        For m, l = 1 to n do
            If N_im, N_li are right-closed then
                C := C ∪ {x_m − x_k ≠ d_im + r^g_ik ∨ x_l − x_k ≠ −d_li + r^g_ik}
            EndIf
        EndFor

        Step 2.2
        For m, l, s, t = 1 to n do
            If N_im, N_li, N_ks, N_tk are right-closed then
                C := C ∪ {x_m − x_l ≠ d_im + d_li ∨ x_s − x_t ≠ d_ks + d_tk ∨ x_m − x_s ≠ r^g_ik + d_im − d_ks}
            Endif
        EndFor

    EndFor
EndFor

Return Constraints(N) ∪ C
```

Figure 9.1: Enforcing global consistency

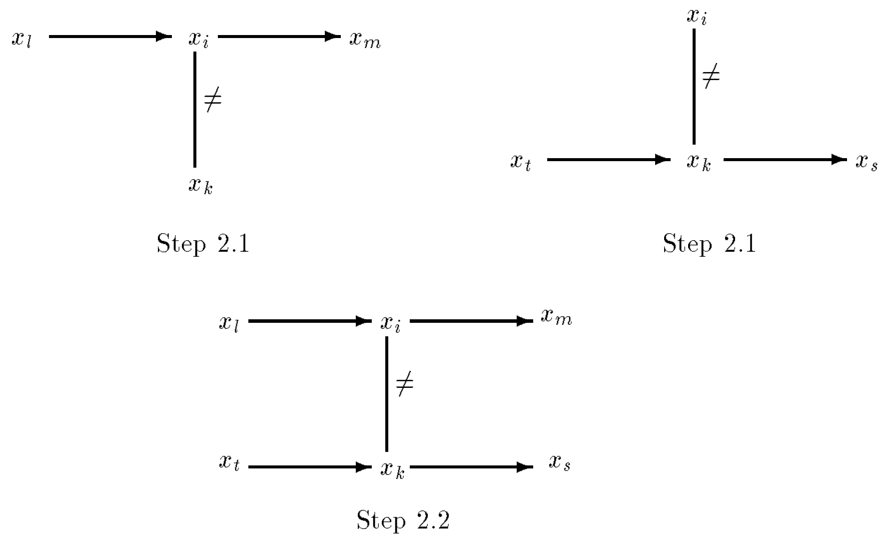

Figure 9.2: The subnetworks examined by step 2 of BCN-GConsistency

Proof: Let N' denote the network N after the first step of BCN-GConsistency has been completed. Let A be *Constraints*(N'). Since all constraints of N are binary and N is consistent, 1- and 2-consistency of A hold trivially. We will show that A is 3-consistent.

Let us take a valuation $v = \{x_1 \leftarrow x_1^0, x_2 \leftarrow x_2^0\}$ such that $A(x_1^0, x_2^0)$ is satisfiable. We will show that for every variable x_3, v can be extended to a valuation $v' = v \cup \{x_3 \leftarrow x_3^0\}$ such that $A(x_1^0, x_2^0, x_3^0)$ is satisfiable.

If $A(x_1, x_2, x_3)$ contains only inequalities, our result is immediate since $Constraints(conv(N'))$ is globally consistent [DMP91, KL91]. Let us then assume that $A(x_1, x_2, x_3)$ contains inequations, and consider $A(x_1^0, x_2^0, x_3)$.

Let I_i be the set of variables j such that $x_j - x_i \neq r$ or $x_i - x_j \neq r$ is an inequation of $A(x_1, x_2, x_3)$. Then $A(x_1^0, x_2^0, x_3)$ can be written as

$$\{x_\mu^0 - d_{3\mu} \prec_1 x_3,\ x_3 \prec_2 x_\lambda^0 + d_{\lambda 3}\} \cup \bigcup_{\zeta \in I_3} \{x_3 \neq x_\zeta^0 + r_{3\zeta}^1, \ldots, x_3 \neq x_\zeta^0 + r_{3\zeta}^{h_{3\zeta}}\}$$

where $\mu, \lambda, \zeta \in \{1, 2\}$ and $\prec_1, \prec_2 \in \{<, \leq\}$.

Since the rational numbers are dense, there is only one case which would not allow us to find a value x_3^0 such that $A(x_1^0, x_2^0, x_3^0)$ is satisfiable. This is the case when $\prec_1$ is $\leq$, $\prec_2$ is $\leq$ and there exists $\rho \in I_3$ and $\eta \in \{1, \ldots, h_{3\rho}\}$ such that

$$x_\mu^0 - d_{3\mu} = x_\lambda^0 + d_{\lambda 3} = x_\rho^0 + r_{3\rho}^\eta. \tag{9.1}$$

We will show that this case *cannot* arise.

Depending on the values of μ, λ and ρ, the following cases must be distinguished:

1. $\mu = \lambda = \rho = 1$. Then (9.1) becomes $x_1^0 - d_{31} = x_1^0 + d_{13} = x_1^0 + r_{31}^\eta$ or equivalently $-d_{31} = d_{13} = r_{31}^\eta$. This means that $N'_{13} = \emptyset$ which contradicts the assumption that N is consistent.

2. $\mu = \lambda = \rho = 1$. This case is symmetric to case 1.

3. $\mu = \rho = 1$ and $\lambda = 2$. Then (9.1) becomes $x_1^0 - d_{31} = x_2^0 + d_{23} = x_1^0 + r_{31}^\eta$ which implies $-d_{31} = r_{31}^\eta$. This is not possible since the inequality on $x_1 - x_3$ in N' is not strict.

The remaining cases are similar to case 3. Thus we have shown that A is strong 3-consistent. ∎

Theorem 9.1.1 *The algorithm* BCN-GConsistency *is correct i.e., it returns a globally consistent set of constraints equivalent to the input network.*

Proof: Given lemma 9.1.1, we only have to show that C is ν-consistent for every ν, $4 \leq \nu \leq n$.

Let us take an arbitrary valuation $v = \{x_1 \leftarrow x_1^0, \ldots, x_{\nu-1} \leftarrow x_{\nu-1}^0\}$ such that $C(x_1^0, \ldots, x_{\nu-1}^0)$ is satisfiable. We will show that for every variable x_ν, v can be extended to a valuation $v' = v \cup \{x_\nu \leftarrow x_\nu^0\}$ such that $C(x_1^0, \ldots, x_\nu^0)$ is satisfiable.

If all constraints involving x_ν and any of $x_1, \ldots, x_{\nu-1}$ are inequalities, our result is immediate since $Constraints(N')$ is globally consistent. Let us then assume that $C(x_1, \ldots, x_\nu)$ contains inequations, and consider $C(x_1^0, \ldots, x_{\nu-1}^0, x_\nu)$.

Let D_{ji} denote the number of inequation constraints involving $x_j - x_i$ in C. Let I_i be the set of natural numbers j such that $x_j - x_i \neq r \ \vee \ \phi$ or $x_i - x_j \neq r \ \vee \ \phi$ is an inequation constraint in C. Then $C(x_1^0, \ldots, x_{\nu-1}^0, x_\nu)$ can be written as

$$\{x_\mu^0 - d_{\nu\mu} \prec_1 x_\nu,\ x_\nu \prec_2 x_\lambda^0 + d_{\lambda\nu}\} \cup \bigcup_{\zeta \in I_\nu} \{x_\nu \neq x_\zeta^0 + r_{\nu\zeta}^1, \ldots, x_\nu \neq x_\zeta^0 + r_{\nu\zeta}^{D_{\nu\zeta}}\} \tag{9.2}$$

where $\mu, \lambda, \zeta \in \{1, \ldots, \nu-1\}$ and $\prec_1, \prec_2 \in \{<, \leq\}$. Since the rational numbers are dense, there is only one case which would not allow us to find a value x_ν^0 such that $C(x_1^0, \ldots, x_{\nu-1}^0, x_\nu^0)$ is satisfiable. This is the case when $\prec_1$ is $\leq$, $\prec_2$ is $\leq$ and there exists $\rho \in I_\nu$ and $\eta \in \{1, \ldots, D_{\nu\rho}\}$ such that

$$x_\mu^0 - d_{\nu\mu} = x_\lambda^0 + d_{\lambda\nu} = x_\rho^0 + r_{\nu\rho}^\eta. \tag{9.3}$$

We will show that this case *cannot* arise.

Depending on the form of the inequation constraint c from which inequation $x_\nu \neq x_\rho^0 + r_{\nu\rho}^\eta$ was generated, the following cases must be considered. Figure 9.3 illustrates the analysis by depicting the subnetworks involved in each case.

1. c is $x_\nu - x_\rho \neq r_{\nu\rho}^\eta \in Constraints(N')$ or equivalently $r_{\nu\rho}^\eta \in holes(N'_{\rho\nu})$. In this case, the constraint $x_\mu - x_\rho \neq d_{\nu\mu} + r_{\nu\rho}^\eta \ \vee \ x_\lambda - x_\rho \neq r_{\nu\rho}^\eta - d_{\lambda\nu}$ is added to C in step 2.1 of algorithm BCN-GConsistency with $g = \eta, m = \mu, l = \lambda$ and $k = \rho$. Then $x_\mu^0 - x_\rho^0 \neq d_{\nu\mu} + r_{\nu\rho}^\eta \ \vee \ x_\lambda^0 - x_\rho^0 \neq r_{\nu\rho}^\eta - d_{\lambda\nu} \in C(x_1^0, \ldots, x_{\nu-1}^0)$ thus we have a contradiction.

2. c is added to C in step 2.1 of BCN-GConsistency. Depending on the values of g, l, i, m and k we can consider the following subcases.

   (a) c is added to C in step 2.1 of BCN-GConsistency with $g = \eta, l = \xi, i = \iota, m = \nu$ and $k = \rho$. Thus c is $x_\nu - x_\rho \neq r_{\iota\rho}^\eta + d_{\iota\nu} \ \vee \ x_\xi - x_\rho \neq -d_{\xi\iota} + r_{\iota\rho}^\eta$. The constraints (9.2), (9.3) and c imply

   $$x_\xi^0 - x_\rho^0 = -d_{\xi\iota} + r_{\iota\rho}^\eta, \quad x_\mu - x_\rho = d_{\nu\mu} + r_{\nu\rho}^\eta = d_{\nu\mu} + r_{\iota\rho}^\eta + d_{\iota\nu} \tag{9.4}$$

   Now we have the following subcases:

   i. $d_{\iota\mu} = d_{\iota\nu} + d_{\nu\mu}$. Then (9.4) contradicts the constraint

   $$x_\mu - x_\rho \neq d_{\iota\mu} + r_{\iota\rho}^\eta \ \vee \ x_\xi - x_\rho \neq -d_{\xi\iota} + r_{\iota\rho}^\eta$$

   of $C(x_1, \ldots, x_{\nu-1})$ which is introduced in step 2.1 of BCN-GConsistency with $g = \eta, l = \xi, i = \iota, m = \mu$ and $k = \rho$.

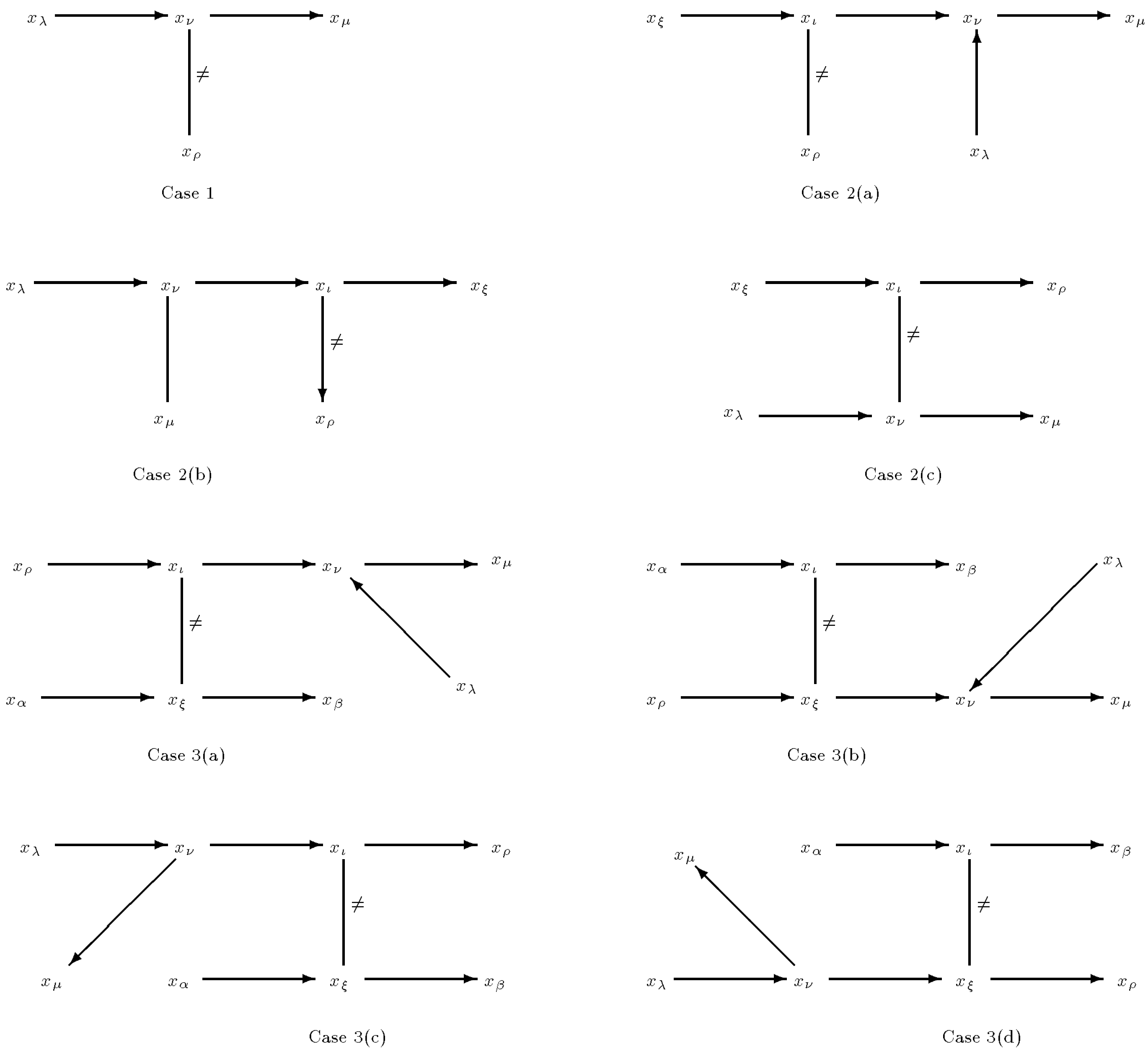

Figure 9.3: The cases examined in theorem 9.1.1

ii. $d_{\iota\mu} < d_{\iota\nu} + d_{\nu\mu}$. Then $x^0_\mu - x^0_\xi \leq d_{\xi\mu} \leq d_{\xi\iota} + d_{\iota\mu} < d_{\xi\iota} + d_{\iota\nu} + d_{\nu\mu}$. This contradicts $x^0_\mu - x^0_\xi = d_{\xi\iota} + d_{\iota\nu} + d_{\nu\mu}$ which is implied by (9.4).

(b) c is added to C in step 2.1 of BCN-GConsistency with $g = \eta, l = \nu, i = \iota, m = \xi$ and $k = \rho$. Thus c is $x_\xi - x_\rho \neq r^\eta_{\iota\rho} + d_{\iota\xi} \ \vee \ x_\nu - x_\rho \neq -d_{\nu\iota} + r^\eta_{\iota\rho}$. This case is symmetric to 2(a).

(c) c is added to C in step 2.1 BCN-GConsistency with $g = \eta, l = \xi, i = \iota, m = \rho$ and $k = \nu$. Thus c is $x_\rho - x_\nu \neq r^\eta_{\iota\nu} + d_{\iota\rho} \ \vee \ x_\xi - x_\nu \neq -d_{\xi\iota} + r^\eta_{\iota\nu}$ or equivalently $x_\nu \neq x_\rho - r^\eta_{\iota\nu} - d_{\iota\rho} \vee x_\nu \neq x_\xi + d_{\xi\iota} - r^\eta_{\iota\nu}$. The constraints (9.2) and c imply $x^0_\rho - r^\eta_{\iota\nu} - d_{\iota\rho} = x^0_\xi + d_{\xi\iota} - r^\eta_{\iota\nu} = x^0_\rho + r^\eta_{\nu\rho}$. These equalities together with (9.3) imply

$$x^0_\rho - x^0_\xi = d_{\xi\iota} + d_{\iota\rho}, \quad x^0_\mu - x^0_\lambda = d_{\nu\mu} + d_{\lambda\nu}, \quad x^0_\rho - r^\eta_{\iota\nu} - d_{\iota\rho} = x^0_\mu - d_{\nu\mu}.$$

But for $g = \eta, l = \xi, m = \rho, i = \iota, k = \nu, t = \lambda$ and $s = \mu$ the constraint

$$x_\rho - x_\xi \neq d_{\iota\rho} + d_{\xi\iota} \ \vee \ x_\mu - x_\lambda \neq d_{\nu\mu} + d_{\lambda\nu} \ \vee \ x_\rho - x_\mu \neq r^\eta_{\iota\nu} + d_{\iota\rho} - d_{\nu\mu}$$

is added to C in step 2.2 of BCN-GConsistency. This constraint also belongs to $C(x_1, \ldots, x_{\nu-1})$ thus we have a contradiction.

3. c is added to C in step 2.2 of BCN-GConsistency. Depending on the values of g, l, i, m, t, k and s we can consider the following subcases.

(a) c is added to C in step 2.2 of BCN-GConsistency with $g = \eta, l = \rho, i = \iota, m = \nu, t = \alpha, k = \xi$ and $s = \beta$. Thus c is

$$x_\nu - x_\rho \neq d_{\iota\nu} + d_{\rho\iota} \ \vee \ x_\beta - x_\alpha \neq d_{\xi\beta} + d_{\alpha\xi} \ \vee \ x_\nu - x_\beta \neq r^\eta_{\iota\xi} - d_{\xi\beta} + d_{\iota\nu}.$$

The constraints (9.2) and c imply

$$x^0_\rho + d_{\iota\nu} + d_{\rho\iota} = x^0_\beta + r^\eta_{\iota\xi} - d_{\xi\beta} + d_{\iota\nu} = x^0_\rho + r^\eta_{\nu\rho}, \quad x^0_\beta - x^0_\alpha = d_{\xi\beta} + d_{\alpha\xi}.$$

These equations together with (9.3) imply

$$x^0_\mu - x^0_\rho = d_{\iota\nu} + d_{\rho\iota} + d_{\nu\mu}, \quad x^0_\beta - x^0_\alpha = d_{\xi\beta} + d_{\alpha\xi}, \quad x^0_\mu - x^0_\beta = d_{\nu\mu} + r^\eta_{\iota\xi} - d_{\xi\beta} + d_{\iota\nu}. \tag{9.5}$$

Now we have to consider the following subcases:

i. $d_{\iota\mu} = d_{\iota\nu} + d_{\nu\mu}$. Then (9.5) contradicts the constraint

$$x_\mu - x_\rho \neq d_{\iota\mu} + d_{\rho\iota} \ \vee \ x_\beta - x_\alpha \neq d_{\xi\beta} + d_{\alpha\xi} \ \vee \ x_\mu - x_\beta \neq r^\eta_{\iota\xi} + d_{\iota\mu} - d_{\xi\beta}$$

of $C(x_1, \ldots, x_{\nu-1})$ which is added to C in step 2.2 of BCN-GConsistency with $g = \eta, l = \rho, i = \iota, m = \mu, t = \alpha, k = \xi$ and $s = \beta$.

ii. $d_{\iota\mu} < d_{\iota\nu} + d_{\nu\mu}$. Then

$$x^0_\mu - x^0_\rho \leq d_{\rho\mu} \leq d_{\rho\iota} + d_{\iota\mu} < d_{\rho\iota} + d_{\iota\nu} + d_{\nu\mu}.$$

This contradicts the first equation of (9.5).

(b) c is introduced in step 2.2 of BCN-GConsistency with $g = \eta, l = \alpha, i = \iota, m = \beta, t = \rho, k = \xi$ and $s = \nu$. Thus c is

$$x_\beta - x_\alpha \neq d_{\iota\beta} + d_{\alpha\iota} \ \vee \ x_\nu - x_\rho \neq d_{\xi\nu} + d_{\rho\xi} \ \vee \ x_\beta - x_\nu \neq r^\eta_{\iota\xi} + d_{\iota\beta} - d_{\xi\nu}.$$

This case is symmetric to case 3(a).

(c) c is introduced in step 2.2 of BCN-GConsistency with $g = \eta, l = \nu, i = \iota, m = \rho, t = \alpha, k = \xi$ and $s = \beta$. Thus c is

$$x_\rho - x_\nu \neq d_{\iota\rho} + d_{\nu\iota} \ \vee \ x_\beta - x_\alpha \neq d_{\xi\beta} + d_{\alpha\xi} \ \vee \ x_\rho - x_\beta \neq r^\eta_{\iota\xi} + d_{\iota\rho} - d_{\xi\beta}$$

or $x_\nu - x_\rho \neq -d_{\iota\rho} - d_{\nu\iota} \ \vee \ x_\beta - x_\alpha \neq d_{\xi\beta} + d_{\alpha\xi} \ \vee \ x_\rho - x_\beta \neq r^\eta_{\iota\xi} + d_{\iota\rho} - d_{\xi\beta}$. The constraints (9.2) and c imply

$$x^0_\beta - x^0_\alpha = d_{\xi\beta} + d_{\alpha\xi}, \quad x^0_\rho - x^0_\beta = r^\eta_{\iota\xi} + d_{\iota\rho} - d_{\xi\beta}, \quad r^\eta_{\nu\rho} = -d_{\iota\rho} - d_{\nu\iota}$$

The above equations together with (9.3) imply

$$x^0_\beta - x^0_\alpha = d_{\xi\beta} + d_{\alpha\xi}, \quad x^0_\rho - x^0_\beta = r^\eta_{\iota\xi} + d_{\iota\rho} - d_{\xi\beta}, \quad x^0_\rho - x^0_\lambda = d_{\lambda\nu} + d_{\iota\rho} + d_{\nu\iota}. \quad (9.6)$$

Now we have to consider the following cases:

i. $d_{\lambda\iota} = d_{\lambda\nu} + d_{\nu\iota}$. Then (9.6) contradicts the constraint

$$x_\rho - x_\lambda \neq d_{\iota\rho} + d_{\lambda\iota} \ \vee \ x_\beta - x_\alpha \neq d_{\xi\beta} + d_{\alpha\xi} \ \vee \ x_\rho - x_\beta \neq r^\eta_{\iota\xi} + d_{\iota\rho} - d_{\xi\beta}$$

of $C(x_1, \ldots, x_{\nu-1})$ which is introduced in step 2.2 of of BCN-GConsistency with $g = \eta, l = \lambda, i = \iota, m = \rho, t = \alpha, k = \xi$ and $s = \beta$.

ii. $d_{\lambda\iota} < d_{\lambda\nu} + d_{\nu\iota}$. Then

$$x^0_\rho - x^0_\lambda \leq d_{\lambda\rho} \leq d_{\lambda\iota} + d_{\iota\rho} < d_{\lambda\nu} + d_{\nu\iota} + d_{\iota\rho}$$

which contradicts the last equation of (9.6).

(d) c is introduced in step 2.2 of BCN-GConsistency with $g = \eta, l = \alpha, i = \iota, m = \beta, t = \nu, k = \xi$ and $s = \rho$. Thus c is

$$x_\beta - x_\alpha \neq d_{\iota\xi} + d_{\alpha\iota} \ \vee \ x_\rho - x_\nu \neq d_{\xi\rho} + d_{\nu\xi} \ \vee \ x_\beta - x_\rho \neq r^\eta_{\iota\xi} + d_{\iota\beta} - d_{\xi\rho}.$$

This case is symmetric to case 3(c).

■

Corollary 9.1.1 *Strong 5-consistency is necessary and sufficient for achieving global consistency of a BCN.*

Proof: Example 9.1.4 shows the necessity of achieving strong 5-consistency. The sufficiency follows from the previous theorem; the algorithm BCN-GConsistency essentially achieves strong 5-consistency. ■

9.2 An Improved Algorithm for Global Consistency

As we have already mentioned, BCN-GConsistency can generate disjunctions of inequations that are not strictly necessary for enforcing global consistency. This can happen in the following ways:

1. An inequation constraint can be equivalent to *true*. For example if the BCN corresponding to the constraint set $C = \{x_1 - x_2 \leq 1, \ x_4 - x_1 \leq 2, \ x_1 \neq x_4\}$ is given as input to BCN-GConsistency, step 2.1 introduces the inequation constraint $x_4 - x_4 \neq 2 \ \vee \ x_2 - x_4 \neq 1$ which is equivalent to *true*.

2. A new inequation constraint can be subsumed by a constraint of N. For example if the BCN corresponding to the constraint set $C = \{x_1 - x_2 \leq 1,\ x_3 - x_1 \leq 2,\ x_3 - x_2 < 3,\ x_1 \neq x_4\}$ is given as input to BCN-GConsistency, step 2.1 introduces the inequation constraint $x_3 - x_4 \neq 2 \vee x_2 - x_4 \neq -1$. This disjunction is equivalent to $x_3 - x_2 \neq 3 \vee x_2 - x_4 \neq -1$ thus it is subsumed by $x_3 - x_2 < 3$.

3. An inequation constraint of C can be subsumed by another inequation constraint of C. For example, for the set of constraints

$$\{x_1 - x_2 \leq 3,\ x_2 - x_3 \leq 2,\ x_1 - x_4 \leq 4,\ x_4 - x_3 \leq 1,\ x_2 - x_4 \neq 1\}$$

BCN-GConsistency adds to C the following disjunctions of inequations:

$$x_2 - x_4 \neq 1,\ x_1 - x_4 \neq 4 \vee x_3 - x_4 \neq -1,\ x_2 - x_1 \neq -3 \vee x_2 - x_3 \neq 2,\ x_1 - x_3 \neq 5.$$

Observe that $x_1 - x_3 \neq 5$ subsumes the second and the third constraint.

We will now discuss how to improve BCN-GConsistency to avoid the generation of redundant constraints.

Improving step 2.1. The body of the for-loop of lines 15-19 must be executed only if $m \neq l, m \neq k$ and $l \neq k$. Let us see why this is enough. Assume that $m = l = \mu,\ i = \iota$ and $k = \xi$. Then the disjunction generated by step 2.1 is

$$x_\mu - x_\xi \neq d_{\iota\mu} + r^g_{\iota\xi} \ \vee\ x_\mu - x_\xi \neq -d_{\mu\iota} + r^g_{\iota\xi}.$$

If $d_{\iota\mu} = -d_{\mu\iota}$ then the above constraint becomes $x_\mu - x_\xi \neq d_{\iota\mu} + r^g_{\iota\xi}$. Since $N_{\iota\mu} = [d_{\iota\mu}, d_{\iota\mu}]$ the above constraint is not necessary. An equivalent constraint is added to N in step 1 of BCN-GConsistency (when $pass = 2, k = \iota, i = \xi$ and $j = \mu$).

Let us now assume that $m = k = \mu,\ i = \iota$ and $l = \lambda$. Then the inequation constraint generated by step 2.1 is $0 \neq d_{\iota\mu} + r^g_{\iota\mu} \vee x_\lambda - x_\mu \neq -d_{\lambda\iota} + r^g_{\iota\mu}$. This constraint evaluates to *true* since $-d_{\iota\mu} < r^g_{\iota\mu}$. Similar comments apply when $l = k$.

Improving step 2.2. Initially, we can make the following improvements.

1. The for-loop of lines 21-25 must be executed only when $i < k$. In this way we avoid considering inequation $x_i - x_k \neq r^g_{ik}$ twice. The same comment does *not* apply to the for-loop of lines 15-19.

2. The body of the for-loop of lines 20-25 must be executed only if $m \neq l$ and $s \neq t$.

   Discussion: Let us assume that $m = l = \mu, s = \sigma, t = \tau, i = \iota$ and $k = \xi$. Then the generated constraint is

$$0 \neq d_{\iota\mu} + d_{\mu\iota} \ \vee\ x_\sigma - x_\tau \neq d_{\xi\sigma} + d_{\tau\xi} \ \vee\ x_\mu - x_\sigma \neq r^g_{\iota\xi} + d_{\iota\mu} - d_{\xi\sigma}. \qquad (9.7)$$

   If $d_{\iota\mu} + d_{\mu\iota} \neq 0$ then the above constraint evaluates to *true*. If $d_{\iota\mu} = -d_{\mu\iota}$ then $Constraints(N) \models x_\mu - x_\iota = d_{\iota\mu}$. Thus step 1 propagates $x_\iota - x_\xi \neq r^g_{\iota\xi}$ and adds $x_\mu - x_\xi \neq d_{\iota\mu} + r^g_{\iota\xi}$ to N (when $pass = 2, i = \xi, k = \iota$ and $j = \mu$). Then in step 2.1 the constraints

$$x_\sigma - x_\xi \leq d_{\xi\sigma},\ x_\xi - x_\tau \leq d_{\tau\xi},\ x_\xi - x_\mu \neq -d_{\iota\mu} - r^g_{\iota\xi}$$

   are used to generate $x_\sigma - x_\mu \neq d_{\xi\sigma} - d_{\iota\mu} - r^g_{\iota\xi} \vee x_\tau - x_\mu \neq -d_{\tau\xi} - d_{\iota\mu} - r^g_{\iota\xi}$. Now observe that the last constraint is just another way of writing (9.7). Similar comments apply when $s = t$.

With the above changes, step 2.2 introduces only constraints which participate in the enforcement of 4- or 5-consistency (depending on the values of m, l, s and t). Step 2.2 can be improved further by considering these two cases separately. The conditions under which step 2.2 enforces 4-consistency are discussed in detail below.

1. $l = t$ and $m = s$.

   Discussion: Let us assume that $l = t = \lambda, m = s = \mu, i = \iota$ and $k = \xi$. Then step 2.2 examines the constraint set

   $$\{x_\mu - x_\iota \le d_{\iota\mu},\ x_\iota - x_\lambda \le d_{\lambda\iota},\ x_\mu - x_\xi \le d_{\xi\mu},\ x_\xi - x_\lambda \le d_{\lambda\xi},\ x_\iota - x_\xi \ne r^g_{\iota\xi}\}$$

   and generates the constraint

   $$x_\mu - x_\lambda \ne d_{\iota\mu} + d_{\lambda\iota} \ \vee\ x_\mu - x_\lambda \ne d_{\xi\mu} + d_{\lambda\xi} \ \vee\ 0 \ne r^g_{\iota\xi} + d_{\iota\mu} - d_{\xi\mu}.$$

   If $r^g_{\iota\xi} + d_{\iota\mu} - d_{\xi\mu} = 0$ and $d_{\iota\mu} + d_{\lambda\iota} = d_{\xi\mu} + d_{\lambda\xi}$ then the above constraint becomes $x_\mu - x_\lambda \ne d_{\iota\mu} + d_{\lambda\iota}$ otherwise it evaluates to *true*.

2. $l = s$ and $m = t$.

   Discussion: Let us assume that $l = s = \lambda, m = t = \mu, i = \iota$ and $k = \xi$. Then step 2.2 examines the constraint set

   $$\{x_\mu - x_\iota \le d_{\iota\mu},\ x_\iota - x_\lambda \le d_{\lambda\iota},\ x_\lambda - x_\xi \le d_{\xi\lambda},\ x_\xi - x_\mu \le d_{\mu\xi},\ x_\iota - x_\xi \ne r^g_{\iota\xi}\}$$

   and generates a constraint equivalent to

   $$x_\mu - x_\lambda \ne d_{\iota\mu} + d_{\lambda\iota} \ \vee\ x_\mu - x_\lambda \ne -d_{\mu\xi} - d_{\xi\lambda} \ \vee\ -d_{\iota\mu} - d_{\mu\xi} \ne r^g_{\iota\xi}.$$

   If $r^g_{\iota\xi} + d_{\iota\mu} + d_{\mu\xi} = 0$ and $d_{\iota\mu} + d_{\lambda\iota} = -d_{\mu\xi} - d_{\xi\lambda}$ then the above constraint becomes $x_\mu - x_\lambda \ne d_{\iota\mu} + d_{\lambda\iota}$ otherwise it evaluates to *true*.

3. $l = t$.

   Discussion: Let us assume that $l = t = \lambda, m = \mu, s = \sigma, i = \iota$ and $k = \xi$. Then step 2.2 examines the constraint set

   $$\{x_\mu - x_\iota \le d_{\iota\mu},\ x_\iota - x_\lambda \le d_{\lambda\iota},\ x_\sigma - x_\xi \le d_{\xi\sigma},\ x_\xi - x_\lambda \le d_{\lambda\xi},\ x_\iota - x_\xi \ne r^g_{\iota\xi}\}$$

   and generates a constraint equivalent to

   $$x_\mu - x_\lambda \ne d_{\iota\mu} + d_{\lambda\iota} \ \vee\ x_\sigma - x_\lambda \ne d_{\xi\sigma} + d_{\lambda\xi} \ \vee\ d_{\lambda\iota} - d_{\lambda\xi} \ne r^g_{\iota\xi}.$$

   If $d_{\lambda\iota} - d_{\lambda\xi} - r^g_{\iota\xi} = 0$ then the above constraint becomes

   $$x_\mu - x_\lambda \ne d_{\iota\mu} + d_{\lambda\iota} \ \vee\ x_\sigma - x_\lambda \ne d_{\xi\sigma} + d_{\lambda\xi}$$

   otherwise it evaluates to *true*.

4. $l = s$.

   Discussion: Let us assume that $l = s = \lambda, m = \mu, t = \tau, i = \iota$ and $k = \xi$. Then step 2.2 examines the constraint set

   $$\{x_\mu - x_\iota \le d_{\iota\mu},\ x_\iota - x_\lambda \le d_{\lambda\iota},\ x_\lambda - x_\xi \le d_{\xi\lambda},\ x_\xi - x_\tau \le d_{\tau\xi},\ x_\iota - x_\xi \ne r^g_{\iota\xi}\}$$

and generates the constraint

$$x_\mu - x_\lambda \neq d_{\iota\mu} + d_{\lambda\iota} \ \vee \ x_\lambda - x_\tau \neq d_{\xi\lambda} + d_{\tau\xi} \ \vee \ x_\mu - x_\lambda \neq d_{\iota\mu} - d_{\xi\lambda} + r^g_{\iota\xi}.$$

If $d_{\lambda\iota} + d_{\xi\lambda} - r^g_{\iota\xi} = 0$ then the above constraint becomes

$$x_\mu - x_\lambda \neq d_{\iota\mu} + d_{\lambda\iota} \ \vee \ x_\lambda - x_\tau \neq d_{\xi\lambda} + d_{\tau\xi}$$

otherwise it evaluates to *true*.

5. $m = s$.

   Discussion: Let us assume that $m = s = \mu, l = \lambda, t = \tau, i = \iota$ and $k = \xi$. Then step 2.2 examines the constraint set

   $$\{x_\mu - x_\iota \leq d_{\iota\mu},\ x_\iota - x_\lambda \leq d_{\lambda\iota},\ x_\mu - x_\xi \leq d_{\xi\mu},\ x_\xi - x_\tau \leq d_{\tau\xi},\ x_\iota - x_\xi \neq r^g_{\iota\xi}\}$$

   and generates the constraint

   $$x_\mu - x_\lambda \neq d_{\iota\mu} + d_{\lambda\iota} \ \vee \ x_\mu - x_\tau \neq d_{\xi\mu} + d_{\tau\xi} \ \vee \ -d_{\iota\mu} + d_{\xi\mu} \neq r^g_{\iota\xi}.$$

   If $-d_{\iota\mu} + d_{\xi\mu} = r^g_{\iota\xi}$ then the above constraint becomes

   $$x_\mu - x_\lambda \neq d_{\iota\mu} + d_{\lambda\iota} \ \vee \ x_\mu - x_\tau \neq d_{\xi\mu} + d_{\tau\xi}$$

   otherwise it evaluates to *true*.

6. $m = t$.

   Discussion: Let us assume that $m = t = \mu, l = \lambda, s = \sigma, i = \iota$ and $k = \xi$. Then step 2.2 examines the constraint set

   $$\{x_\mu - x_\iota \leq d_{\iota\mu},\ x_\iota - x_\lambda \leq d_{\lambda\iota},\ x_\sigma - x_\xi \leq d_{\xi\sigma},\ x_\xi - x_\mu \leq d_{\mu\xi},\ x_\iota - x_\xi \neq r^g_{\iota\xi}\}$$

   and generates the constraint

   $$x_\mu - x_\lambda \neq d_{\iota\mu} + d_{\lambda\iota} \ \vee \ x_\sigma - x_\mu \neq d_{\xi\sigma} + d_{\mu\xi} \ \vee \ x_\mu - x_\sigma \neq r^g_{\iota\xi} + d_{\iota\mu} - d_{\xi\sigma}.$$

   If $-d_{\mu\xi} = r^g_{\iota\xi} + d_{\iota\mu}$ then the above constraint becomes

   $$x_\mu - x_\lambda \neq d_{\iota\mu} + d_{\lambda\iota} \ \vee \ x_\sigma - x_\mu \neq d_{\xi\sigma} + d_{\mu\xi}$$

   otherwise it evaluates to *true*.

Other Improvements. Finally, the following general improvements will be made to BCN-GCONSISTENCY.

1. We will build a list L of all inequations in $Constraints(N)$ immediately after step 1 is executed. Then the rest of the algorithm can use L instead of searching through N for inequations.

2. Every disjunction of inequations generated in step 2 will be simplified. Disjunctions that evaluate to *true* will be discarded. Then we will check whether the generated disjunction is subsumed by an inequality of N or another disjunction of C. If this is the case then this disjunction will also be discarded.

Algorithm BCN-GCONSISTENCY1
Input: A consistent BCN N.
Output: A globally consistent set of constraints equivalent to N.

Method:
Step 1: Call PC-2 to enforce strong 3-consistency on N.

Step 2: Build a list L of all inequations in $Constraints(N)$

```
L := ∅
For k = 1 to n do
    For i = k + 1 to n do
        For g = 1 to h_ik do
            L := L ∪ {(i, k, r^g_ik)}
        EndFor
    EndFor
EndFor
```

Step 3: Enforce 4-consistency.

```
For (i, k, r^g_ik) in L do
    For m, l = 1 to n do
        If CHECK3 then
            N_lm := N_lm ⊕ ((−∞, d_li + d_im) ∪ (d_li + d_im, ∞))
        EndIf
    EndFor
EndFor
```

Step 4: Enforce 4-consistency.

```
C_4 := ∅
For (i, k, r^g_ik) in L do
    For m, l = 1 to n do
        If CHECK4A then C_4 := C_4 ∪ {x_m − x_k ≠ d_im + r^g_ik  ∨  x_l − x_k ≠ −d_li + r^g_ik}
    EndFor
    For s, t = 1 to n do
        If CHECK4B then C_4 := C_4 ∪ {x_s − x_i ≠ d_ks − r_ik  ∨  x_t − x_i ≠ −d_tk − r_ik}
    EndFor
EndFor
```

Figure 9.4: An improved algorithm for global consistency

Step 5: Enforce 4-consistency.
$C_5 := \emptyset$
<u>For</u> $l, m, p = 1$ to n <u>do</u>
    $S := \emptyset$
    <u>For</u> (i, k, r^g_{ik}) in L <u>do</u>
        <u>If</u> CHECK5A <u>then</u>
            $S := S \cup \{x_m - x_l \neq d_{im} + d_{li} \ \vee \ x_p - x_l \neq d_{kp} + d_{lk}\}$
        <u>ElseIf</u> CHECK5B <u>then</u>
            $S := S \cup \{x_m - x_l \neq d_{im} + d_{li} \ \vee \ x_l - x_p \neq d_{kl} + d_{pk}\}$
        <u>ElseIf</u> CHECK5C <u>then</u>
            $S := S \cup \{x_m - x_l \neq d_{im} + d_{li} \ \vee \ x_m - x_p \neq d_{km} + d_{pk}\}$
        <u>ElseIf</u> CHECK5D <u>then</u>
            $S := S \cup \{x_m - x_l \neq d_{im} + d_{li} \ \vee \ x_p - x_m \neq d_{kp} + d_{mk}\}$
        <u>EndIf</u>
    <u>EndFor</u>
    $C_5 := C_5 \cup S$
<u>EndFor</u>

Step 6: Enforce 5-consistency.
$C_6 := \emptyset$
<u>For</u> $m, l, s, t = 1$ <u>to</u> n <u>do</u>
    $S := \emptyset$
    <u>For</u> (i, k, r^g_{ik}) in L <u>do</u>
        <u>If</u> CHECK6 <u>then</u>
            $S := S \cup \{x_m - x_l \neq d_{im} + d_{li} \ \vee \ x_s - x_t \neq d_{ks} + d_{tk} \ \vee \ x_m - x_s \neq r^g_{ik} + d_{im} - d_{ks}\}$
    <u>EndFor</u>
    $C_6 := C_6 \cup S$
<u>EndFor</u>

Return $Constraints(N) \cup C_4 \cup C_5 \cup C_6$

Figure 9.5: Algorithm BCN-GConsistency1 (continued)

3. We will use algorithm PC-2 of [DMP91] to enforce strong 3-consistency. In our case, PC-2 terminates. It might also be more efficient than the two-pass procedure of step 1 since it propagates constraints locally in a more "controlled" fashion.

The above discussion leads us to algorithm BCN-GConsistency1 of figures 9.4 and 9.5. The first step of BCN-GConsistency1 calls PC-2 to enforce strong 3-consistency. The second step builds the list L discussed above. The third step of BCN-GConsistency1 comes from step 2.2 of BCN-GConsistency and contributes to the enforcement of 4-consistency. It covers the cases where $(l = t \wedge m = s) \vee (l = s \wedge m = t)$ is true. CHECK3 is a boolean expression which evaluates to true when one of the following two conditions is true:

1. $N_{im}, N_{li}, N_{km}, N_{lk}$ are right-closed, $m \neq l, r^g_{ik} + d_{im} - d_{km} = 0$ and $d_{li} + d_{im} = d_{lk} + d_{km}$.

2. $N_{im}, N_{li}, N_{kl}, N_{mk}$ are right-closed, $m \neq l, r^g_{ik} + d_{im} + d_{mk} = 0$ and $d_{li} + d_{im} = -d_{mk} - d_{kl}$.

The fourth step of BCN-GConsistency1 comes from step 2.1 of BCN-GConsistency and contributes to the enforcement of 4-consistency. CHECK4A is a boolean expression which evaluates to true if all the following conditions are true:

1. N_{im}, N_{li} are right-closed and $m \neq l, m \neq k, l \neq k$.

2. The constraint to be added to C_4 is not subsumed by a constraint in N.

3. The constraint to be added to C_4 is not already in C_4.

CHECK4B is defined similarly to CHECK4A.

The fifth step of BCN-GConsistency1 comes from step 2.2 of BCN-GConsistency and contributes to the enforcement of 4-consistency as well. It covers the cases where $l = t \vee l = s \vee m = s \vee m = t$ is true. The first case of the if-statement covers the case where $l = t$. CHECK5A is a boolean expression which evaluates to true if all the following conditions are true.

1. $N_{im}, N_{li}, N_{kp}, N_{lk}$ are right-closed, $m \neq l, m \neq p, l \neq p$ and $d_{li} - d_{lk} - r^g_{ik} = 0$.

2. The constraint to be added to S is not subsumed by a constraint in N.

3. The constraint to be added to S is not already in S or C_4.[1]

The remaining cases of the if-statement correspond to the cases where $l = s, m = s$ and $m = t$. The boolean expressions CHECK5B, CHECK5C and CHECK5D are defined accordingly.

The final step of BCN-GConsistency1 comes from step 2.2 of BCN-GConsistency and contributes to the enforcement of 5-consistency. It covers the case where x_l, x_m, x_s and x_t are distinct variables. CHECK6 is a boolean expression which evaluates to true if all the following conditions are true.

1. $N_{im}, N_{li}, N_{ks}, N_{tk}$ are right-closed and m, l, s, t are different from each other.

2. The constraint to be added to S is not subsumed by a constraint in N.

3. The constraint to be added to S is not subsumed by a constraint in $C_4 \cup C_5$.

4. The constraint to be added to S is not already in S.

[1] For some fixed values of m, l and p, S is the only subset of C_5 where a constraint identical to the one to be added to C_5 can be found. This is obvious since $C_5 \setminus S$ contains constraints with at least one variable different from x_m, x_l and x_p.

The following theorem demostrates the correctness of BCN-GConsistency1. Its proof is an easy consequence of theorem 9.1.1 and the above discussion.

Theorem 9.2.1 *The algorithm* BCN-GConsistency1 *is correct i.e., it returns a globally consistent set of constraints equivalent to the input network.*

We will now establish an upper bound on the running time of BCN-GConsistency1. Let H denote the total number of inequations in $Constraints(N)$ initially. Let I denote the total number of inequations in $Constraints(N)$ after step 1 of BCN-GConsistency1 has been executed. In other words, I is the length of list L built in step 2 of BCN-GConsistency1. Observe that $m \leq H$ and $I = O(n^2 H)$. The latter equality has only theoretical value. In practice m, H and I will usually be much smaller than n. Since we want to be realistic in our calculations, the time bounds will be given in terms of both H and I.

Theorem 9.2.2 *The running time of* BCN-GConsistency1 *is* $O(n^4 I^2) = O(n^8 H^2)$.

Proof: We will first estimate the running time of step 1 of algorithm BCN-GConsistency. Let m' be the maximum number of "holes" in any interval N_{ij} during the execution of step 1. If I_1 and I_2 are almost-convex intervals containing more than one value then $I_1 \otimes I_2 = conv(I_1) \otimes conv(I_2)$. If $I_1 = [a, h_1) \cup (h_1, h_2) \cup \cdots \cup (h_{m'}, b]$ and $I_2 = [c, c]$ then $I_1 \otimes I_2 = [a + c, h_1 + c) \cup (h_1 + c, h_2 + c) \cup \cdots \cup (h_{m'} + c, b + c]$. Since $m' \leq nm$, we can conclude that the composition and intersection operations of step 1 of algorithm BCN-GConsistency can be executed in $O(nm)$ time. Therefore step 1 of algorithm BCN-GConsistency takes $O(n^4 m)$ time. The running time of PC-2 is bounded by the running time of step 1 of BCN-GConsistency. In other words, PC-2 takes $O(n^4 m)$ time.

In step 2 the list L can be built in $O(n^2 m') = O(n^3 m)$ time. As we have already said, the length of L is $I = O(n^2 H)$.

Let us now estimate the time needed to determine the truth value of the boolean expressions with prefix CHECK. The truth value of CHECK3 can be determined in constant time. The truth value of the first two conditions of CHECK4A can be determined in constant time. The truth value of the third condition can be checked by the following procedure: search through the inequations of L processed so far to determine whether a constraint identical to the one we have has already been generated. The latter check can be done in constant time since the values of m and l will be fixed. In a similar way we can see that the remaining expressions with prefix CHECK can be evaluated in $O(I)$ time.

We conclude that step 3 takes $O(n^2 I) = O(n^4 H)$ time, step 4 takes $O(n^2 I^2) = O(n^6 H^2)$ time, step 5 takes $O(n^3 I^2) = O(n^7 H^2)$ time and step 6 takes $O(n^4 I^2) = O(n^8 H^2)$ time. Finally BCN-GConsistency1 takes $O(n^4 I^2) = O(n^8 H^2)$ time. ■

Corollary 9.2.1 *If no redundancy checks are performed then the global consistency of an BCN can be enforced in* $O(n^4 I) = O(n^6 H)$ *time.*

Let us now ivestigate the relationship between enforcing global consistency and computing the minimal network.

Theorem 9.2.3 *The minimal network equivalent to a BCN* N *can be computed by executing steps 1, 2 and 3 of* BCN-GConsistency1.

Proof: Let M be the minimal network of N, C the set returned by BCN-GConsistency1(N) and C' the set of binary constraints in C. From the definitions of global consistency and minimal network, we have that $Constraints(M) \subseteq C'$. Finally, observe that steps 4-6 of BCN-GConsistency1 add to C only disjunctions of inequations. Therefore they should not be taken into account when computing M. ■

Corollary 9.2.2 *The minimal network equivalent to a BCN can be computed in $O(n^4H)$ time.*

Example 9.2.1 If M is the minimal network equivalent to the BCN N in figure 8.1 then

$$Constraints(M) = Constraints(N) \cup \{x_3 - x_2 < 2,\ x_3 - x_1 > -4\}.$$

9.3 Global Consistency of Point Algebra Networks

We will now turn our attention to an important subset of BCN: the point algebra (PA) networks introduced in [VKvB89]. Let us recall the definition of PA networks from section 2.2. A *point algebra (PA) network* is a labeled directed graph where nodes represent variables and edges represent PA constraints. The labels of the edges are chosen from the set of relations $\{<, \leq, >, \geq, =, \neq, ?\}$. The symbol ? is used to label an edge $i \rightarrow j$ whenever there is no constraint between variables x_i and x_j.

Van Beek and Cohen have studied PA networks in detail [vBC90, vB92]. The reader is invited to compare the following results of [vBC90, vB92] to theorems 8.2.2 and 9.2.3.

Theorem 9.3.1 *Deciding whether a given PA network is consistent can be done in $O(n^2)$ time where n is the number of nodes.*

Theorem 9.3.2 *The minimal network equivalent to a PA network can be computed in $O(max(In^2, n^3))$ time where n is the number of nodes and I is the number of edges labeled with $\neq$ after path consistency is enforced.*

In [vBC90] the minimal network is computed by algorithm AAC. The steps of AAC are in almost exact correspondence with steps 1, 2 and 3 of algorithm BCN-GConsistency1. However, in the proof of correctness of AAC (theorem 4 of [vBC90]), Van Beek and Cohen suggest that the algorithm for computing the minimal network of a given PA network also achieves global consistency. This is not true as the following example demonstrates.

Example 9.3.1 For the PA network with constraints

$$x_1 \leq x_2,\ x_2 \leq x_3,\ x_4 \leq x_5,\ x_5 \leq x_6, x_2 \neq x_5$$

AAC will also introduce constraints $x_1 \leq x_3,\ x_4 \leq x_6$. The resulting PA network is strong 3-consistent but *not* globally consistent. This can be demonstrated via an argument similar to the one for example 9.1.4. If we enforce strong 5-consistency with the addition of constraints $x_1 \neq x_5 \lor x_3 \neq x_5,\ x_4 \neq x_2 \lor x_6 \neq x_2$ and $x_1 \neq x_3 \lor x_1 \neq x_4 \lor x_1 \neq x_6$, then the resulting set is globally consistent.

The above disjunctions of inequations are necessary for achieving global consistency. But algorithm AAC of [vBC90] does not introduce these disjunctions so it cannot achieve global consistency.

Global consistency of PA networks can be enforced by BCN-GConsistency1 if PA networks are represented by their equivalent BCN. Alternatively, global consistency can be enforced by the following modification of BCN-GConsistency1. Step 1 of BCN-GConsistency1 will be substituted by the algorithm Path-Consistency of [vB92] (page 314). Step 2 can remain unchanged. Step 3 can be substituted by the algorithm Find-Subgraphs of [vB92] (page 314). Steps 4, 5 and 6 can be appropriately modified. The redundancy checks of BCN-GConsistency1 should be performed for PA networks as well. We can also achieve more efficiency by making use of adjacency lists as [vB92] shows. The following theorem concludes our discussion.

Theorem 9.3.3 *For PA networks, strong 5-consistency is necessary and sufficient for achieving global consistency. Strong 5-consistency can be enforced in $O(n^4 I^2)$ time.*

Let us now comment on some observations of Dechter [Dec92] on the problem of enforcing global consistency in PA networks. [Dec92] discusses global consistency in general constraint networks with finite variable domains. The most important result of [Dec92] is the following. If N is a constraint network with constraints of arity r or less and domains of size k or less which is strongly $(k(r-1)+1)$-consistent, then N is globally consistent.

The above result can be applied to PA networks if PA networks are redefined as "traditional" constraint networks where variables represent relations between two points and constraints are defined by the transitivity table of [VKvB89]. This representation yields a constraint network with $k = 3$ and $r = 3$. Dechter's result now gives us the following. If strong 7-consistency in PA networks can be enforced with ternary constraints then strong 7-consistency implies global consistency. Dechter uses the aforementioned incorrect assertion of [vBC90] to conclude that strong 7-consistency in the traditional formulation of PA networks can be enforced with ternary constraints. Thus she also concludes that in the traditional formulation strong 7-consistency implies global consistency [Dec92, page 100]. In the light of theorem 9.3.3, Dechter's conclusion remains unjustified.

9.4 Conclusion

In this chapter we studied the problem of enforcing global consistency of BCN. We identified the level of local consistency which is necessary and sufficient for achieving global consistency, and presented detailed algorithms for this task. We also analyzed in sufficient depth the problem of enforcing global consistency in PA networks and pointed out an error in the published literature on the subject.

Chapter 10

Concluding Remarks

Our journey has now come to an end. Let us summarize the main achievements of this dissertation. We pointed out that reasoning about time is important in many applications e.g., planning, scheduling, medical expert systems, geographical information systems, office information systems and natural language processing systems. These applications can benefit from a database model which allows the representation of *definite, indefinite, finite* and *infinite* temporal information in a single unifying framework. There is currently *no* database model with this functionality. We argued that the combination of *relational databases* and *temporal constraints* offers a powerful framework which addresses the requirements of the above applications.

Initially our developments took place in a very abstract setting. Following [JL87, KKR90] we developed three database schemes: $\mathbf{M}_{\mathcal{L}}$-relational databases, $\mathcal{L}$-constraint databases and indefinite $\mathcal{L}$-constraint databases. The language $\mathcal{L}$ defines the constraint vocabulary and $\mathbf{M}_{\mathcal{L}}$ is the structure over which $\mathcal{L}$-constraints will be interpreted. The scheme of $\mathcal{L}$-constraint databases has been originally proposed in [KKR90]. We have enriched the theory of this scheme by presenting an algebraic query language which lends itself to more "practical" query evaluation algorithms than the ones described in [KKR90]. The temporal database model and query language of [KSW90] is an instance of our scheme. We also extended the $\mathcal{L}$-constraint databases model to include indefinite information in the style of [IL84, Gra89]. We defined declarative and procedural query languages for the new model and studied the semantics of query evaluation.

Then we studied several theories of time allowing the representation of qualitative and quantitative temporal constraints on points and intervals. We distinguished between discrete and dense time structures and studied the problems of quantifier elimination and decision of the corresponding theories. The most important theories we considered are subtheories of Presburger arithmetic and real addition with order. Therefore we expect our results to be of interest to the theoretical computer science community as well.

Following [KKR90], we used our results on quantifier elimination and decision for analyzing the complexity of query evaluation in temporal constraint databases and indefinite temporal constraint databases. Our most important result is the following. The complexity of query evaluation (data or combined) does *not* change when we move from relational calculus queries over relational databases to relational calculus with temporal constraints over temporal constraint databases. The same result is true when we consider indefinite information. This is a very encouraging result which lends support to the feasibility of temporal constraint databases.

Finally, we studied a new tractable class of temporal constraints involving inequations and disjunctions of inequations.

10.1 Future Work

There are a number of problems which need to be addressed in future work. An implementation of the model of temporal constraint databases is badly needed. Our complexity results suggest that an efficient implementation is indeed possible when *only definite* information is present (considering discrete or dense time). In this effort we can be guided by implementations of similar temporal reasoning systems (e.g., TMM [Dea89]), implementations of constraint logic programming languages and preliminary results on constraint query languages [JMSY92, SR92, Sri92, JBM93]. Handling indefinite information will be more challenging. From the existing systems only TMM has addressed this case in some depth by considering polynomial time algorithms that are sound but incomplete. The model which will be implemented must also be enhanced with deductive rules. Among other things this will allow us to express causality, persistence [DM87, KS86] and recursive queries.

On the theoretical front now there are a number of interesting open questions related to the languages diPCL and dePCL. It is interesting to investigate whether diPCL and dePCL can become more expressive while retaining the PSPACE upper bound for quantifier elimination. In this context the following questions are important:

- What is the combined complexity of evaluating queries in relational calculus with diPCL-constraints over the linear repeating point databases of [KSW90]? The data complexity of this problem is in LOGSPACE as follows from [FR75].
- What is the complexity of quantifier elimination in dePC if the atomic formulas are extended to arbitrary inequalities with *at most two* variables per inequality?

What happens when we add recursion and negation to the existential fragment of relational calculus with diPCL (or dePCL) constraints? [Rev90] has showed that Datalog with integer gap-order constraints (a subclass of diPCL-constraints) can be evaluated bottom-up in closed form with PTIME data complexity. [CM90] showed that the combined complexity of the same problem is EXPTIME (and it remains the same if the domain is densely ordered). [Rev90] has also proved that every Turing-computable function can be expressed by a program in stratified $Datalog^{\neg}$ with integer order constraints. We would like to investigate the case of stratified $Datalog^{\neg}$ with dePCL-constraints. For some more open problems for Datalog with diPCL-constraints see [Rev90].

Another interesting theoretical problem is to identify classes of indefinite diPCL- or dePCL-constraint databases and modal relational calculus queries for which query evaluation is tractable. The work of [vdM92] is an important first step towards this direction.

Our concern for temporal constraints has been shared by researchers in real-time temporal logics [Alu91] timed automata [Dil89] and logic programming with real-time modal operators [Brz91, Brz93a, Brz93b]. It will be fruitful to investigate the connections among the above formalisms. Preliminary work towards this direction has been reported in [BCW93, Brz91, Brz93a, Brz93b].

Another important area of future research is the development of practical algorithms for solving the reasoning problems discussed in chapters 3, 8 and 9. For the subclass of qualitative dePCL-constraints the most satisfactory solutions have been reported in [GS93]. Finally an important open problem for dePCL-constraints with disjunctions of inequations is *global variable elimination* i.e., eliminating all given variables in one operation and not one by one. Preliminary work on this problem has been reported in [Imb93].

Bibliography

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993. Preliminary version appears in the Proc. of 5th LICS, 1990.

[AH85] J.F. Allen and P.J. Hayes. A Common-Sense Theory of Time. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 528–531, Los Angeles, CA, 1985.

[AH89a] J.F. Allen and P.J. Hayes. Moments and Points In an Interval-Based Temporal Logic. *Computational Intelligence*, 5:225–238, November 1989.

[AH89b] R. Alur and T.A. Henzinger. A really temporal logic. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 164–169, 1989. Full version to appear in JACM.

[AH90] R. Alur and T.A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, 1990.

[AKG91] S. Abiteboul, P. Kanellakis, and G. Grahne. On the Representation and Querying of Sets of Possible Worlds. *Theoretical Computer Science*, 78(1):159–187, 1991.

[AKP91] J.F. Allen, H. Kautz, and R. Pelavin, editors. *Reasoning About Plans.* Morgan-Kaufmann, 1991.

[All81] J.F. Allen. An Interval Based Representation of Temporal Knowledge. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 221–226, Vancouver, B.C., 1981.

[All83] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[All84] J.F. Allen. Towards a General Model of Action and Time. *Artificial Intelligence*, 23(2):123–154, July 1984.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-Time Systems.* PhD thesis, Dept. of Computer Science, Stanford University, 1991.

[AM89] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

[AS86] I. Ahn and R. Snodgrass. Performance Evaluation of a Temporal Database Management System. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 96–107, 1986.

[Bau89a] M. Baudinet. *Logic programming semantics: Techniques and applications.* PhD thesis, Dept. of Computer Science, Stanford University, 1989.

[Bau89b] M. Baudinet. Temporal logic programming is complete and expressive. In *Proceedings of the Symposium on Principles of Programming Languages*, pages 267–280, 1989.

[BCW93] Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Temporal Deductive Databases. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 13. Benjamin/Cummings Pub. Co., 1993.

[BFAT91] R. Barrera, A. Frank, and K. Al-Taha. Temporal Relations in GIS: A Workshop at the University of Maine. *Sigmod Record*, 20(3):85–91, 1991.

[BG89] Baker Andrew B. and Matthew L. Ginsberg. Temporal Projection and Explanation. In *Proceedings of IJCAI-89*, pages 906–911, 1989.

[Bis81] J. Biskup. A formal approach to null values in database relations. In Gallaire, H. and Minker, J. and Nicolas, J.M., editor, *Advances in Database Theory*, volume 1, pages 299–341. Plenum Press, 1981.

[Bis83] J. Biskup. A foundation of Codd's relational may be operations. *ACM Transaction on Database Systems*, 8(4):608–636, December 1983.

[BNW91] Marianne Baudinet, Marc Niezette, and Pierre Wolper. On the Representation of Infinite Temporal Data and Queries. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 280–290, 1991.

[Bod93] Mark Boddy. Temporal Reasoning for Planning and Scheduling. *SIGART Bulletin*, 4(3):17–20, 1993.

[BP85] F. Barbic and B. Pernici. Time Modeling in Office Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 51–62, 1985.

[Brz91] C. Brzoska. Temporal Logic Programming and its Relation to Constraint Logic Programming. In *Proceedings of the 1991 Logic Programming Symposium*, 1991.

[Brz93a] C. Brzoska. Temporal Logic Programming with Bounded Universal Goals. In *Proceedings of the 10th International Conference on Logic Programming*, pages 239–256, 1993.

[Brz93b] C. Brzoska. Temporal Logic Programming with Metric and Past Operators Based on Constraint Logic Programming. Technical Report 2/93, University of Karlsruhe, Faculty of Informatics, 1993.

[BTA89] Fahiem Bacchus, Josh Tenenberg, and Koomen Johannes A. A Non-Reified Temporal Logic. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proceedings of 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 2–10, Toronto, Ontario, 1989.

[CC87] James Clifford and Albert Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of the International Conference on Data Engineering*, pages 528–537, 1987.

[CCT93] J. Clifford, A. Croker, and A. Tuzhilin. On the Completeness of Query Languages for Grouped and Ungrouped Historical Data Models. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 20. Benjamin/Cummings Pub. Co., 1993.

[CH82] A. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.

[Cha77] Tat-Hung Chan. An Algorithm For Checking PL/CV Arithmetic Inferences. Technical Report 77-326, Dept. of Computer Science, Cornell University, 1977.

[Cha88] S. Chaudhuri. Temporal Relationships in Databases. In *Proceedings of the 14th VLDB Conference*, pages 160–170, 1988.

[Cho90a] J. Chomicki. *Functional Deductive Databases: Query Processing in the Presence of Limited Function Symbols.* PhD thesis, Dept. of Computer Science, Rutgers University, 1990. Also available as LCSR-TR-142.

[Cho90b] J. Chomicki. Polynomial Time Query Processing in Temporal Deductive Databases. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 379–391, 1990.

[Cho93] Jan Chomicki. Temporal Databases. Unpublished notes from a tutorial presented at the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 1993.

[CI88] J. Chomicki and T. Imielinski. Temporal Deductive Databases and Infinite Objects. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 61–73, 1988.

[CI89] J. Chomicki and T. Imielinski. Relational Specifications of Infinite Query Answers. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 174–183, 1989.

[CK73] C.C. Chang and H.J. Keisler. *Model Theory.* North Holland, 1973.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[CM85] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison Wesley, 1985.

[CM90] J. Cox and K. McAloon. Decision Procedures for Constraint Based Extensions of Datalog. Technical Report 90-09, Dept. of Computer and Information Sciences, Brooklyn College of C.U.N.Y., 1990.

[CM93] J. Cox and K. McAloon. Decision Procedures for Constraint Based Extensions of Datalog. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research.* MIT Press, 1993. Originally appeared as Technical Report No. 90-09, Dept. of Computer and Information Sciences, Brooklyn College of C.U.N.Y.

[Cod75] E.F. Codd. Understanding Relations (Installment #7). *FDT Bulletin of ACM SIGMOD*, 7(3-4):23–28, 1975.

[Cod79] E.F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transaction on Database Systems*, 4(4):379–434, December 1979.

[Col90] A. Colmerauer. An Introduction to Prolog III. *Communications of ACM*, 33(7):69–90, July 1990.

[Coo90] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1):89–95, 1990.

[Cou93] C. Courcoubetis, editor. *Computer Aided Verification, Proceedings of the 5th International Conference*, volume 697 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[CR88] J. Clifford and A. Rao. A Simple, General Structure For Temporal Domains. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in Information Systems*, pages 17–28. North-Holland, 1988.

[CT85] James Clifford and Abdullah Uz Tansel. On An Algebra For Historical Relational Databases: Two Views. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 247–265, 1985.

[CW83] James Clifford and David Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, 8(2):214–254, 1983.

[DB88] T. Dean and M. Boddy. Reasoning About Partially Ordered Events. *Artificial Intelligence*, 36:375–399, 1988.

[DBG+85] U. Dayal, A. Buchmann, D. Goldhirsch, S. Heiler, F. A. Manola, J.A. Orenstein, and A.S. Rosenthal. PROBE – A Research Project in Knowledge-Oriented Database Systems: Preliminary Analysis. Technical Report CCA-85-03, Computer Corporation of America, 1985.

[DdC88] R. Demolombe and L.F. del Cerro. An Algebraic Evaluation Method for Deduction in Incomplete Databases. *Journal of Logic Programming*, 5:183–205, 1988.

[Dea88] T. Dean. Reasoning About the Effects of Actions in Automated Planning Systems. *Annals of Operations Research*, 12:147–167, 1988.

[Dea89] T. Dean. Using Temporal Hierarchies to Efficiently Maintain Large Temporal Databases. *Journal of ACM*, 36(4):687–718, 1989.

[Dec92] Rina Dechter. From local to global consistency. *Artificial Intelligence*, 55:87–107, 1992.

[DFM88] T. Dean, R. James Firby, and D. Miller. Hierarchical Planning Involving Deadlines, Travel Time and Resources. *Computational Intelligence*, 4:381–398, 1988.

[Dil89] D. Dill. Timing Assumptions and Verification of Finite State Concurrent Systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite-State Systems*, volume 407 of *Lecture Notes in Computer Science*. Springer Verlag, 1989.

[DM87] T. Dean and D.V. McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32:1–55, 1987.

[DMP89] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proceedings of 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 83–93, Toronto, Ontario, 1989.

[DMP91] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991. Special Volume on Knowledge Representation.

[Doy79] Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12:231–272, 1979.

[DP88] Rina Dechter and Judea Pearl. Network-Based Heuristics for Constraint Satisfaction Problems. *Artificial Intelligence*, 34(1):1–38, 1988.

[DS93] C. Dyreson and R. Snodgrass. Valid-time Indeterminacy. In *Proceedings of the 9th International Conference on Data Engineering*, pages 335–343, 1993.

[Dut88] S. Dutta. Temporal Reasoning in Medical Expert Systems. In *Proceedings of IEEE Symposium on the Engineering of Computer-Based Medical Systems*, pages 118–122, 1988.

[DWP81] D.R. Dowty, R.E. Wall, and S. Peters. *Introduction to Montague Semantics*. D. Reidel Publishing Company, 1981.

[End72] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[EWK93] R. Elmasri, G. Wuu, and V. Kouramajian. A Temporal Model and Query Language for EER Databases. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 10. Benjamin/Cummings Pub. Co., 1993.

[FG77] J. Ferrante and J.R. Geiser. An Efficient Decision Procedure for the Theory of Rational Order. *Theoretical Computer Science*, 4(2):227–233, 1977.

[Fox87] M. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. London: Pitman, 1987.

[FR74] M.J. Fischer and M.O. Rabin. Super-exponential Complexity of Presburger Arithmetic. In *Proc. of AMS Symposium on Complexity of Real Computational Processes*, volume III, 1974.

[FR75] J. Ferrante and C. Rackoff. A Decision Procedure for the First Order Theory of Real Addition with Order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[FR79] J. Ferrante and C. Rackoff. *The Computational Complexity of Logical Theories*. Lecture Notes in Mathematics. Springer Verlag, 1979.

[Fre78] E. Freuder. Synthesizing Constraint Expressions. *Communications of ACM*, 21(11):958–966, November 1978.

[Fre82] E. Freuder. A Sufficient Condition For Backtrack-Free Search. *Journal of ACM*, 29(1):24–32, 1982.

[GA89] M. Ghallab and M.A. Alaoui. Managing Efficiently Temporal Relations through Indexed Spanning Trees. In *Proceedings of IJCAI-89*, pages 1297–1303, 1989.

[Gab89a] D. Gabbay. Modal and Temporal Logic Programming. In A. Galton, editor, *Temporal Logics and Their Applications*, Lecture Notes in Computer Science, pages 197–237. Academic Press, 1989.

[Gab89b] D. Gabbay. The Declarative Past and Imperative Future. In *Colloquium on Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 402–450. Springer Verlag, 1989.

[Gad86] Sashi K. Gadia. Toward a Multihomogeneous Model For a Temporal Database. In *Proceedings of the International Conference on Data Engineering*, pages 390–397, 1986.

[Gad88] Sashi K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, 1988.

[Gal90] Antony Galton. A Critical Examination of Allen's Theory of Action and Time. *Artificial Intelligence*, 42:159–188, 1990.

[GB89] Sashi K. Gadia and Gautam Bhargava. Achieving Zero Information Loss In a Classical Database Environment. In *Proceedings of the 15th VLDB Conference*, pages 217–224, 1989.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Co., 1979.

[GLR91] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the Limitations of the Situation Calculus. Working Notes, AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning, 1991.

[GNP92] S.K. Gadia, S. S. Nair, and Y.-C. Poon. Incomplete information in relation temporal databases. In *Proceedings of VLDB-92*, pages 395–406, 1992.

[Gra89] Gosta Grahne. The Problem of Incomplete Information in Relational Databases. Technical Report Report A-1989-1, Department of Computer Science, University of Helsinki, Finland, 1989. Also published as Lecture Notes in Computer Science 554, Springer Verlag, 1991.

[GS88a] Matthew L. Ginsberg and David E. Smith. Reasoning About Action I: A Possible Worlds Approach. *Artificial Intelligence*, 35:165–195, 1988.

[GS88b] Matthew L. Ginsberg and David E. Smith. Reasoning About Action II: The Qualification Problem. *Artificial Intelligence*, 35:311–342, 1988.

[GS93] A. Gerevini and L. Schubert. Efficient Temporal Reasoning Through Timegraphs. In *Proceedings of IJCAI-93*, pages 648–654, 1993.

[GSS93] A. Gerevini, L. Schubert, and S. Schaeffer. Temporal Reasoning in Timegraph I-II. *SIGART Bulletin*, 4(3):21–25, 1993.

[GV85] Sashi K. Gadia and Jay H. Vaishnav. A Query Language For a Homogeneous Temporal Database. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 51–56, 1985.

[GY88] Sashi K. Gadia and Chuen-Sing Yeung. A Generalized Model For a Relational Temporal Database. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 251–259, 1988.

[HA87] P.J. Hayes and J.F. Allen. Short Time Periods. In *Proceedings of IJCAI-87*, pages 981–983, Los Angeles, CA, 1987.

[Hau87] Brian Haugh. Non-Standard Semantics for the Method of Temporal Arguments. In *Proceedings of IJCAI-87*, pages 449–455, 1987.

[HC68] G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic*. Methuen, 1968.

[HM86] Steve Hanks and Drew McDermott. Default Reasoning, Nonmonotonic Logics, and the Frame Problem. In *Proceedings of AAAI-86*, pages 328–333, 1986.

[HS86] Joseph Y. Halpern and Yoav Shoham. A Propositional Modal Logic of Time Intervals. In *Proceedings of Symposium on Logic in Computer Science*, pages 279–292, 1986.

[HSW75] G.D. Held, M.R. Stonebraker, and E. Wong. INGRES-A relational data base system. In *Proceedings of the 1975 National Computer Conference*, pages 409–416, Anaheim, CA, May 1975.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory Languages and Computation*. Addison-Wesley, 1979.

[IL84] T. Imielinski and W. Lipski. Incomplete Information in Relational Databases. *Journal of ACM*, 31(4):761–791, 1984.

[Imb93] J.-L. Imbert. Variable Elimination for Generalized Linear Constraints. In *Proceedings of the 10th International Conference on Logic Programming*, 1993.

[Imi89] T. Imielinski. Incomplete Information in Logical Databases. *Data Engineering*, 12(2):29–39, 1989.

[INV91] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete Objects - A Data Model for Design and Planning Applications. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 288–297, 1991.

[IV89] T. Imielinski and K. Vadaparty. Complexity of Querying Database with OR-Objects. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1989.

[JBM93] Jaffar J., A. Brodsky, and M. Maher. Towards Practical Constraint Databases. In *Proceedings of 19th International Conference on Very Large Databases (VLDB-93)*, pages 567–580, 1993.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[JMSY92] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transaction on Programming Languages and Systems*, 14(3):339–395, July 1992.

[Joh90] D.S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. North-Holland, 1990.

[JR93] Pinto J. and R. Reiter. Adding a Time Line to the Situation Calculus. In *The Second Symposium on Logical Formalizations of Commonsense Reasoning*, pages 172–177, 1993.

[Kan90] Paris Kanellakis. Elements of Relational Database Theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 17. North-Holland, 1990.

[Kau86] Henry A. Kautz. The Logic of Persistence. In *Proceedings of AAAI-86*, pages 401–405, 1986.

[KKR90] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint Query Languages. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 299–313, 1990. Long version to appear in Journal of Computer and System Sciences.

[KL91] H. Kautz and P. Ladkin. Integrating Metric and Qualitative Temporal Reasoning. In *Proceedings of AAAI-91*, pages 241–246, 1991.

[Klo81] M.R. Klopprogge. TERM: An Approach to Include the Time Dimension in the Entity-Relationship Model. In P.P. Chen, editor, *Entity-Relationship Approach to Information Modelling and Analysis*, pages 477–512. ER Institute, 1981.

[Koo89] Johannes Koomen. The TIMELOGIC Temporal Reasoning System. Technical Report TR-231 (revised), Dept. of Computer Science, University of Rochester, March 1989.

[Kou92] Manolis Koubarakis. Dense Time and Temporal Constraints with $\neq$. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR'92)*, pages 24–35. Morgan Kaufmann, San Mateo, CA, October 1992.

[Kou93a] Manolis Koubarakis. Representation and Querying in Temporal Databases: the Power of Temporal Constraints. In *Proceedings of the 9th International Conference on Data Engineering*, pages 327–334, April 1993.

[Kou93b] Manolis Koubarakis. Representation and Querying in Temporal Databases: the Power of Temporal Constraints (Long Version). Technical Report KDBSLAB-TR-93-01, Knowledge and Data Base Systems Laboratory, Computer Science Division, Dept. of Electrical and Computer Engineering, National Technical University of Athens, August 1993.

[Kow86] R.A Kowalski. Database updates in the event calculus. Technical Report DoC 86/12, Department of Computing, Imperial College, 1986.

[KS86] R.A Kowalski and M.J. Sergot. A Logic-based Calculus of Events. *New Generation Computing*, 1(4):67–95, 1986.

[KSW90] F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling Infinite Temporal Data. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 392–403, 1990.

[Lad86a] Peter Ladkin. Primitives and Units for Time Specification. In *Proceedings of AAAI-86*, pages 354–359, 1986.

[Lad86b] Peter Ladkin. Time Representation: A Taxonomy of Interval Relations. In *Proceedings of AAAI-86*, pages 360–366, 1986.

[Lad87a] Peter Ladkin. Constraint Satisfaction in Time Interval Structures I: Convex Intervals. Technical Report KES.U.87.11 (Revised), Kestrel Institute Technical Report, 1987.

[Lad87b] Peter Ladkin. Models of Axioms for Time Intervals. In *Proceedings of AAAI-87*, pages 234–239, 1987.

[Lad87c] Peter Ladkin. Models of Axioms for Time Intervals (Revised Version). Technical Report KES.U.87.4, Kestrel Institute Technical Report, 1987.

[Lad87d] Peter Ladkin. The Completeness of a Natural System for Reasoning with Time Intervals. In *Proceedings of IJCAI-87*, pages 462–467, 1987.

[Lad88] Peter Ladkin. Satisfying First-Order Constraints About Time Intervals. In *Proceedings of AAAI-88*, pages 512–517, 1988.

[Lev84] H.J. Levesque. Foundations of a Functional Approach to Knowledge Representation. *Artificial Intelligence*, 23:155–212, 1984.

[LHM89] Jean-Louis Lassez, Tien Huynh, and Ken McAloon. Simplification and Elimination of Redundant Linear Arithmetic Costraints. In *Proceedings of NACLP-89*, pages 37–51. MIT Press, 1989.

[Lif87] Vladimir Lifschitz. Formal Theories of Action. In F. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 35–57. Morgan Kaufmann, 1987.

[Lip79] Witold Jr. Lipski. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transcactions on Database Systems*, 4(3):262–296, September 1979.

[Lip81] Witold Jr. Lipski. On Databases with Incomplete Information. *Journal of the ACM*, 28(1):41–70, January 1981.

[LJ88a] N.A. Lorentzos and R.G. Johnson. An Extension of the Relational Model to Support Generic Intervals. In *Proc. of EDBT-88*, pages 528–542, 1988.

[LJ88b] Nikos A. Lorentzos and Roger G. Johnson. Extending Relational Algebra To Manipulate Temporal Data. *Information Systems*, 13(3):289–296, 1988.

[LM88a] Peter Ladkin and Roger Maddux. On Binary Constraint Networks. Technical Report KES.U.88.8, Kestrel Institute Technical Report, 1988.

[LM88b] Peter Ladkin and Roger Maddux. Representation and Reasoning with Convex Time Intervals. Technical Report KES.U.88.2, Kestrel Institute Technical Report, 1988.

[LM89] Jean-Louis Lassez and Ken McAloon. A Canonical Form for Generalized Linear Costraints. Technical Report RC15004 (#67009), IBM Research Division, T.J. Watson Research Center, 1989.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, 1992.

[LS92] F. Lin and Y. Shoham. Concurrent Actions in the Situation Calculus. Working Notes, 4th International Workshop on Non-Monotonic Reasoning, 1992.

[LS93a] D. Lomet and B. Salzberg. Differential Query Processing in Transaction-Time Databases. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 19. Benjamin/Cummings Pub. Co., 1993.

[LS93b] D. Lomet and B. Salzberg. Transaction Time Databases. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 16. Benjamin/Cummings Pub. Co., 1993.

[Mac77] A.K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[Mah93] M. Maher. A Logic Programming View of CLP. In *Proceedings of the 10th International Conference on Logic Programming*, pages 737–753, 1993.

[Mai83] David Maier. *The theory of relational databases*. Computer Science Press, 1983.

[MBJK90] John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: A Language for Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, 8(4):325–362, October 1990.

[McD82] D.V. McDermott. A Temporal Logic for Reasoning About Processes and Plans. *Cognitive Science*, 6:101–155, 1982.

[Mei91a] I. Meiri. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *Proceedings of AAAI-91*, pages 260–267, 1991.

[Mei91b] I. Meiri. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. Technical Report R-160, Cognitive Systems Laboratory, University of California, Los Angeles, 1991.

[MF85] A.K. Mackworth and E.C. Freuder. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 25(1):65–74, 1985.

[MH69] John McCarthy and Patrick J. Hayes. Some Philosophical Problems From the Standpoint of Artificial Intelligence. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence*, pages 463–502. Edinburg University Press, 1969.

[Mon74] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences*, 7:95–132, 1974.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1991.

[MPB92] R. Maiocchi, B. Pernici, and F. Barbic. Automatic Deduction of Temporal Information. *ACM Transactions on Database Systems*, 17(4):647–688, 1992.

[MS87] E. McKenzie and R. Snodgrass. Extending the Relational Algebra to Support Transaction Time. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 467–478, 1987.

[MS88a] S. Miller and L. Schubert. Time Revisited. In *Proceedings of CSCSI-88*, pages 39–45, Edmonton, Alberta, 1988.

[MS88b] S. Miller and L. Schubert. Using Specialists to Accelerate General Reasoning. In *Proceedings of AAAI-88*, pages 161–165, 1988.

[MS88c] Leora Morgenstern and Lynn Andrea Stein. Why Things Go Wrong: A Formal Theory of Causal Reasoning. In *Proceedings of AAAI-88*, pages 518–523, 1988.

[MS90] S. Miller and L. Schubert. Time Revisited. *Computational Intelligence*, 4:108–118, 1990.

[NS92] M. Niezette and J.-M. Stevenne. An Efficient Symbolic Representation of Periodic Time. In *Proceedings of the 1st International Conference on Information and Knowledge Management*, 1992.

[OW92] M.A. Orgun and W.W. Wadge. Towards a unified theory of intensional logic programming. *Journal of Logic Programming*, 13(4), 1992.

[PR93a] J. Pinto and R. Reiter. Temporal Reasoning in Logic Programming: A Case for the Situation Calculus. In *Proceedings of the 10th International Conference on Logic Programming*, 1993.

[PR93b] J. Pinto and R. Reiter. Temporal Reasoning in Situation Calculus. Forthcoming, 1993.

[Pra77] V.R. Pratt. Two Easy Theories Whose Combination Is Hard. Technical Report, Massachusetts Institute of Technology, Cambridge, Mass., September 1977.

[Pri57] Arthur Prior. *Time and Modality.* Clarendon Press, 1957.

[Pri67] Arthur Prior. *Past, Present and Future.* Clarendon Press, 1967.

[Rab77] M.O. Rabin. Decidable theories. In *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 595–629. North-Holland, 1977.

[RB89a] B. Richards and I. Bethke. Temporal databases: An IQ approach. In *Temporal Representation and Inference*, Cognitive Science Series, pages 231–284. Academic Press, 1989.

[RB89b] B. Richards and I. Bethke. The temporal logic IQ. In *Temporal Representation and Inference*, Cognitive Science Series, pages 211–230. Academic Press, 1989.

[Rei84] Raymond Reiter. Towards a logical reconstruction of relational database theory. In Michael Brodie, John Mylopoulos, and Joachim Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, pages 191–233. Springer Verlag, 1984.

[Rei86] R. Reiter. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values. *Journal of the Association for Computing Machinery*, 33(2):349–370, 1986.

[Rei87] Ray Reiter. Nonmonotonic Reasoning. *Annual Reviews of Computer Science*, 2:147–186, 1987.

[Rei88] Ray Reiter. On Integrity Constraints. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 97–111, Asilomar, CA, 1988.

[Rei89] Han Reichgelt. A Comparison of First Order and Modal Logics of Time. In Peter Jackson, Han Reichgelt, and Frank van Harmelen, editors, *Logic-Based Knowledge Representation*, pages 143–176. MIT Press, 1989.

[Rei91] Ray Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Rev90] Peter Z. Revesz. A Closed Form for Datalog Queries with Integer Order. In *Proceedings of the 3rd International Conference on Database Theory*, pages 187–201, 1990. Long version to appear in Theoretical Computer Science.

[RJC91] B. Richards, Y. Jiang, and H. Choi. On Interval-Based Temporal Planning: An IQ strategy. In *Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems*, volume 542 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[RL78] C.R. Reddy and D.W. Loveland. Presburger Arithmetic with Bounded Quantifier Alternation. In *Proc. of ACM Symposium on the Theory of Computing*, pages 320–325, 1978.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic*. Springer Verlag, 1971.

[SA85] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 236–246, 1985.

[SAD93] J. Stillman, R. Arthur, and A. Deitsch. Tachyon: A constraint-based temporal reasoning model and its implementation. *SIGART Bulletin*, 4(3), 1993.

[SBC92] R. Schrag, M. Boddy, and J. Carciofini. Managing Disjunction for Practical Temporal Reasoning. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR'92)*, pages 36–46. Morgan Kaufmann, San Mateo, CA, October 1992.

[Sch86] A. Schrijver, editor. *Theory of Integer and Linear Programming*. Wiley, 1986.

[Sch90] L. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, 1990.

[SFG85] A. Sathi, M. Fox, and M. Greenberg. Representation of Activity Knowledge for Project Management. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(5):531–552, 1985.

[SG88] Yoav Shoham and Nita Goyal. Temporal Reasoning in Artificial Intelligence. In Strobe, Howard and AAAI, editor, *Exploring Artificial Intelligence*, pages 419–439. Morgan Kaufmann, 1988.

[Sha89] M. Shanahan. Prediction is Deduction but Explanation is Abduction. In *Proceedings of IJCAI-89*, 1989.

[Sho81] Robert Shostak. Deciding Linear Inequalities by Computing Loop Residues. *Journal of the ACM*, 28(4):769–779, 1981.

[Sho87a] Yoav Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, 1987.

[Sho87b] Yoav Shoham. Temporal Logics in AI: Semantical and Ontological Considerations. *Artificial Intelligence*, 33:89–104, 1987.

[Sho88] Yoav Shoham. Chronological Ignorance: Experiments in Nonomonotonic Temporal Reasoning. *Artificial Intelligence*, 36:279–331, 1988.

[SM88] Yoav Shoham and Drew McDermott. Problems in Formal Temporal Reasoning. *Artificial Intelligence*, 36:49–61, 1988.

[Sno87] R. Snodgrass. The Temporal Query Language TQuel. *ACM Transcactions on Database Systems*, 12(2):247–298, June 1987.

[Sno92] R. Snodgrass. Temporal Databases. In *Proceedings of the International Conference on GIS – From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 22–64, Pisa, Italy, September 1992.

[Sno93] R. Snodgrass. An Overview of TQUEL. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 6. Benjamin/Cummings Pub. Co., 1993.

[Son85] E. Sontag. Real Addition and the Polynomial Time Hierarchy. *Information Processing Letters*, 20:115–120, 1985.

[SR92] Divesh Srivastava and Raghu Ramakrishnan. Pushing Constraint Selections. In *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 301–315, 1992.

[SRH90] Michael Stonebraker, Lawrence A. Rowe, and Michael Hirohama. The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, 1990.

[Sri88] S.M. Sripada. A logical framework for temporal deductive databases. In *Proceedings of the 14th VLDB Conference*, pages 171–182, 1988.

[Sri89] S. Sripada. A Metalogic Programming Approach to Reasoning About Time in Knowledge Bases. In *Proceedings of IJCAI-93*, pages 860–865, 1989.

[Sri91] S.M. Sripada. *Temporal Reasoning in Deductive Databases*. PhD thesis, Dept. of Computing, Imperial College, London, 1991.

[Sri92] Divesh Srivastava. Subsumption and Indexing in Constraint Query Languages with Linear Arithmetic Constraints. In *Proceedings of the 2nd International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 1992.

[Sto77] L.J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Stu91] P.J. Stuckey. Constructive Negation for Constraint Logic Programming. In *Proceedings of Symposium on Logic in Computer Science*, pages 328–339, 1991.

[SWKH76] M. Stonebraker, E. Wong, P. Kreps, and G. Held. The Design and Implementation of INGRES. *ACM Transcactions on Database Systems*, 1(3):189–222, 1976.

[TCG+93] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and Snodgrass R., editors. *Temporal Databases: Theory, Design and Implementation.* Database Systems and Applications Series. Benjamin/Cummings Pub. Co., 1993.

[Ull88] J. Ullman. *Principles of Data Base and Knowledge Base Systems*, volume 1. Computer Science Press, 1988.

[Vad92] K. Vadaparty. *Querying Databases with Non-Deterministic Information.* PhD thesis, Dept. of Computer Science, Rutgers University, 1992.

[Var82] Moshe Vardi. The Complexity of Relational Query Languages. In *Proceedings of ACM SIGACT/SIGMOD Symposium on Principles of Database Systems*, pages 137–146, 1982.

[Var86] Moshe Vardi. Querying Logical Databases. *Journal of Computer and System Sciences*, 33:142–160, 1986.

[Vas79] Y. Vassiliou. Null values in database management, a denotational semantics approach. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 162–169, 1979.

[Vas80] Y. Vassiliou. Functional dependencies and incomplete information. In *Proceedings of 7th International Conference on Very Large Databases (VLDB-80)*, pages 260–269, 1980.

[vB83] J. van Benthem. *The Logic of Time.* D. Reidel Publishing Company, 1983.

[vB89a] Peter van Beek. Approximation Algorithms for Temporal Reasoning. In *Proceedings of IJCAI-89*, pages 1291–1296, 1989.

[vB89b] J. van Benthem. Time, Logic and Computation. In W. Brauer, G. Rozenberg, and A. Salomaa, editors, *Lecture Notes in Computer Science*, volume 354, pages 1–44. Springer-Verlag, 1989.

[vB90a] Peter van Beek. Exact and Approximate Reasoning About Qualitative Temporal Relations. Technical Report TR 90-29, Department of Computing Science, University of Alberta, August 1990.

[vB90b] Peter van Beek. Reasoning About Qualitative Temporal Information. In *Proceedings of AAAI-90*, pages 728–734, 1990.

[vB92] Peter van Beek. Reasoning About Qualitative Temporal Information. *Artificial Intelligence*, 58:297–326, 1992.

[vBC90] Peter van Beek and Robin Cohen. Exact and Approximate Reasoning about Temporal Relations. *Computational Intelligence*, 6:132–144, 1990.

[vdM92] Ron van der Meyden. The Complexity of Querying Indefinite Data About Linearly Ordered Domains (Preliminary Version). In *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 331–345, 1992.

[vGT91] A. van Gelder and R.W. Topor. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(2):235–278, June 1991.

[vH89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[VK86] Marc Vilain and Henry Kautz. Constraint Propagation Algorithms for Temporal Reasoning. In *Proceedings of AAAI-86*, pages 377–382, 1986.

[VKvB89] Marc Vilain, Henry Kautz, and Peter van Beek. Constraint Propagation Algorithms for Temporal Reasoning: a Revised Report. In D.S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufmann, 1989.

[Web90] Jay C. Weber. On the Representation of Concurrent Actions in the Situation Calculus. In *Proceedings of CSCSI-90*, pages 28–32, Ottawa, Ontario, 1990.

[Wuu92] Wuu, G. and Dayal, U. A Uniform Model for Temporal Object-Oriented Databases. In *International Conference on Data Engineering*, pages 584–593, 1992.

[Wuu93] Wuu, G. and Dayal, U. A Uniform Model for Temporal and Versioned Object-oriented Databases. In Tansel, A. et. al., editor, *Temporal Databases: Theory, Design and Implementation*, chapter 10. Benjamin/Cummings Pub. Co., 1993.

[YA93] E. Yampratoom and J.F. Allen. Performance of Temporal Reasoning Systems. *SIGART Bulletin*, 4(3):26–29, 1993.

[Zan84] C. Zaniolo. Database relations with null values. *Journal of Computer and System Sciences*, 28(1):142–166, 1984.