Handling Constraints in Temporal Databases

Pierre Wolper

Université de Liège

Representing Temporal Data

Every morning I take a shower from 7:00 to 7:30

How can this be represented in a database?

1. Explicitly

Activity

Pierre	shower	Day 0 - 7:00	Day 0 - 7:30
Pierre	shower	Day 1 - 7:00	Day 1 - 7:30
Pierre	shower	Day 2 - 7:00	Day 2 - 7:30
Pierre	shower	Day 3 - 7:00	Day 3 - 7:30
Pierre	shower	Day 4 - 7:00	Day 4 - 7:30
Pierre	shower	Day 5 - 7:00	Day 5 - 7:30
Pierre	shower	Day 6 - 7:00	Day 6 - 7:30

•
•
•

Boring, even infinitely boring!

2. By recursive rules

Activity(Pierre, shower, 7:00, 7:30)

Activity(Pierre, shower, t_1+1day, t_2+1day)$\leftarrow$
Activity(Pierre, shower, t_1, t_2)

Better, but how does one efficiently answer queries?

3. By special constraints:
Linear Repeating Points (LRPs)

Activity

Pierre	shower	7:00+1day.n	7:30+1day.n

Constraint Temporal Databases

Collections of

generalized relations themselves collections of

generalized tuples which include

- temporal attributes,
- non-temporal attributes,
- constraints on temporal attributes.

Representing Temporal Attributes: Linear Repeating Points (lrps)

An lrp is an infinite set of integers represented by an expression of the form

$$x(n) = c + kn,$$

$c, k \in \mathcal{Z}$ and
n taking all values from $-\infty$ to $+\infty$ in $\mathcal{Z}$.

Example :

$$3 + 50n$$

represents

$$\{\ldots, -97, -47, 3, 53, 103, 153, \ldots\}$$

Constraints in Generalized Tuples

- *General constraints* are conjunctions of arbitrary linear equalities or inequalities between temporal attributes.

  Example : $3x \leq 5y + 4$

- *Restricted constraints* are conjunctions of equalities and inequalities in which the coefficients of temporal attributes are 1.

  Example : $x \leq y - 3$

Examples

1. **tuple :** $[1, 1 + 100n] \land X_2 \geq 0$

   set : $\{[1, 101], [1, 201], [1, 301], \ldots\}$

2. **tuple :** $[3 + 20n_1, 5 + 20n_2] \land X_1 = X_2 - 2$

   set : $\{\ldots, [-17, -15], [3, 5], [23, 25], \ldots\}$.

Expressiveness

- A k-ary predicate on the integers is

  lrp definable

  if its extension can be defined by a generalized relation with k temporal arguments using *general* constraints.

- A k-ary predicate on the integers is

  weak lrp definable

  if its extension can be defined by a generalized relation with k temporal arguments using *restricted* constraints.

- A k ary predicate on the integers is

  Presburger definable

  if it is definable by a Presburger arithmetic formula with k free variables.

Example :

$$P(x) \equiv x \geq 1 \ \wedge \ \forall y \ \ x = y + y \supset 0 \leq y \leq 4$$

- A *unary* predicate on the integers is

  weak lrp definable
  iff
  it is *Presburger definable*.

- A *binary* predicate on the integers is

  lrp definable
  iff
  it is *Presburger definable*.

Operations on the database

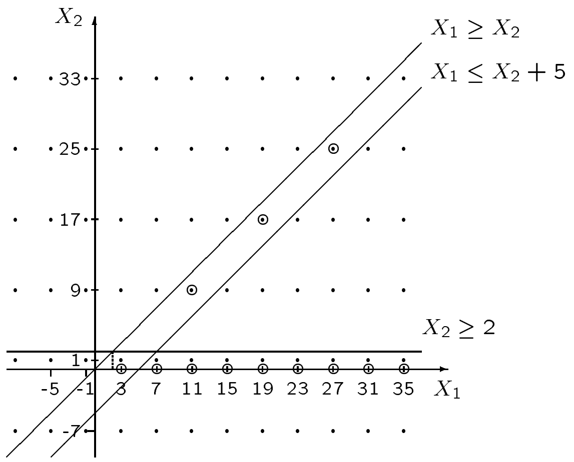
All relational algebra operations :

- union,
- intersection,
- join,
- selection,
- projection and
- negation

can be computed on generalized relations.

Problems with Integer Projection.

$$[4n_1 + 3, 8n_2 + 1] \wedge \begin{cases} X_1 \geq X_2 \wedge \\ X_1 \leq X_2 + 5 \wedge \\ X_2 \geq 2 \end{cases}$$



Solution: Normalization

Tuples with restricted constraints can be replaced by equivalent tuples in *normal form*; i.e. where

- all lrps have the same period and,
- all constraints are rounded inequalities.

With these restrictions classical variable elimination can be performed.

Complexity of the operations

Two types of complexity measures:

- **Fixed-schema complexity:** The number of tuples in the relation(s) varies; the schema of the database, i.e. the number of columns, is assumed to be fixed.
- **General complexity:** Both the number of tuples and the size of the schema vary.

- N: number of tuples
- m: number of attributes

	normalized relation	
Complexity type	Fixed-schema	General
Union	$O(N)$	$O(Nm^2)$
Cross-product	$O(N^2)$	$O(m^2N^2)$
Intersection	$O(N^2)$	$O(m^2N^2)$
Join	$O(N^2)$	$O(m^2N^2)$
Projection	$O(N)$	$O(m^2N)$
Emptiness of a relation	$O(N)$	$O(m^3N)$
Negation	$O(N^c)$	$O(N^{(m^2)})$
Emptiness of the complement relation	$O(N^c)$	$O(N^{(m^2)})$ NP-Complete

A More Abstract View

- A (purely temporal) generalized tuple is a representation of a subset of $\mathbf{Z}^n$
- We have chosen a *normalized from* representation for these subsets of $\mathbf{Z}^n$: lrps and (restricted) constraints.
- The requirement on the normal form is that it be
  - sufficiently expressive,
  - closed under the relational algebra operators (with reasonable efficiency), and
  - Efficiently checkable for consistency and other properties.
- Could there exist other convenient normal forms for describing subsets of $\mathbf{Z}^n$?

A First Idea: Use Presburger Arithmetic Itself

Why not allow full Presburger Arithmetic to describe subsets of $\mathbf{Z}^n$?

- It is expressive,
- All relational algebra operations are immediately "computed".
- Consistency and other properties are very hard to decide (double exponential space).

Using Presburger arithmetic amounts to delaying computation as long as possible. The consequence is that extracting information from the resulting data becomes intractable.

An Interesting Normal Form for Presburger Arithmetic: Automata

- Encode integers as binary strings.
- n-component vectors are encoded as strings over the alphabet 2^n.
- A subset of $\mathbf{Z}^n$ is thus a language over the alphabet 2^n.
- Finite automata are used to represent such languages.

Encoding Integers by Strings

Principles :

- Binary representation,
- Unbounded numbers,
- Most significant bit first.
- 2's complement for negative numbers (at least p bits for a number x such that $-2^{p-1} \leq x < 2^{p-1}$).

Examples :

$$\begin{array}{rcl} 4 & : & 0100, 00100, 000100, \ldots \\ -4 & : & 100, 1100, 11100, \ldots \end{array}$$

Vectors are represented by using same length encodings of the components and reading them bit by bit.

Expressiveness of the Automaton Representation

- To simplify operations, we use automata that accept all valid encodings of a given subset of $\mathbf{Z}^n$.
- The subsets of $\mathbf{Z}^n$ representable by automata are those definable in a slight extension of Presburger arithmetic: one adds a function giving the largest power of 2 dividing its argument.
- If one requires representability by automata in all bases ≥ 2, then the representable subsets are exactly those definable in Presburger arithmetic.

Building Automata for Linear Equations

Consider an equation $\vec{a}.\vec{x} = b$ with $\vec{a} \in \mathbf{Z}^n$ and $b \in \mathbf{Z}$.

The problem is to build an automaton $A = (S, 2^n, \delta, s_0, F)$ accepting the encodings of all $\vec{x} \in \mathbf{Z}^n$ satisfying the equation.

- Each state s of the automaton (except the initial state s_0) is uniquely labeled by an integer $\beta(s)$. The final state is the one labeled by b. The initial state is a special second state labeled by 0.
- The idea of the construction is that the label of a state represents the value of $\vec{a}.\vec{x}$ for the bits that have been read so far.

- Therefore, for states s and s' other than s_0 to be linked by a transition labeled $\vec{d}$, the number $\beta(s)$ associated with the state s' has to be given by

  $$\beta(s') = \vec{a}.\vec{x}' = 2.\vec{a}.\vec{x} + \vec{a}.\vec{d} = 2.\beta(s) + \vec{a}.\vec{d}.$$

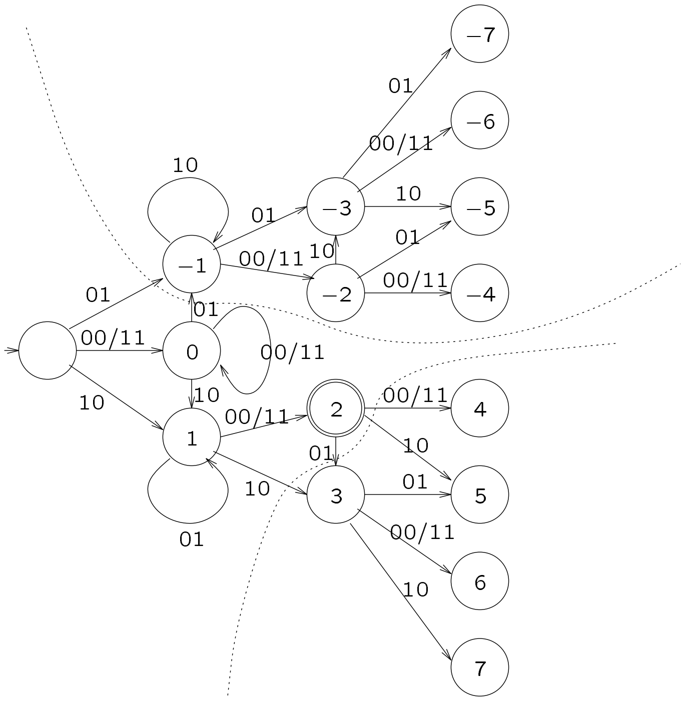
  where $\vec{(}x)$ and $\vec{(}x')$ respectively represent the vectors read when respectively reaching s and s'.

  Note that the state s' is unique.
- For the initial state, the associated value is 0, but one has to take into consideration that the first bit is a sign bit: in the function given the next state, a 1 bit is interpreted as -1.

Termination of the Construction and Inequations

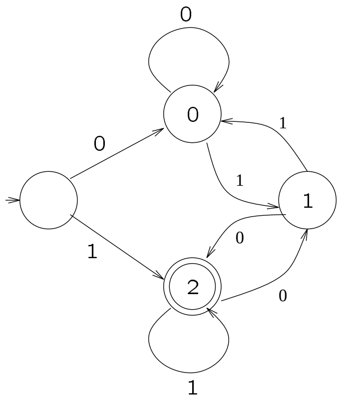
- If $\vec{a} = (a_1, \ldots, a_n)$, then the accepting state cannot be reached from any state s labeled by an integer $\beta(s) \geq |b/2| + \Sigma_i |a_i|$. Thus, only a finite number of states are needed in the automaton.
- In practice, it is more effective to do the construction starting with the final state and proceeding backwards.
- For an inequation $\vec{a}.\vec{x} < b$, one proceeds similarly except that all states with labels $< b$ are accepting.
- The automaton obtained is deterministic and hence can easily be minimized.

Example: $x - y = 2$



LRPs are also easily representable by automata.

Example: $x = 3k + 2$



Computing with Automata

- All relational algebra operations then become simple automata operations.
- Complexity wise, one has the following choice:
  - Always use deterministic automata. All operations are of reasonable complexity (at most $O(N^2)$), except projection (requires a determinization).
  - Allow nondeterministic automata. All operations are of reasonable complexity, except complementation (requires a determinization).
- Checking consistency reduces to finding a path through the automaton.

Potential drawback: Interpreting the result of a query in a natural way might be difficult.

Conclusions

- Constraints are a powerful way of representing temporal information
- A suitable normal form for temporal constraints is the key to an effective implementation of temporal constraint databases.
- Automata operating on the binary representation of numbers might be a useful solution.