Valid-time Indeterminacy

Curtis E. Dyreson and Richard T. Snodgrass

Department of Computer Science
University of Arizona
Tucson, AZ 85721
{curtis,rts}@cs.arizona.edu

Abstract

In valid-time indeterminacy, *it is known that an event stored in a temporal database did in fact occur, but it is not known exactly* when *the event occurred. We present an extension of the tuple-timestamped temporal data model, called the the* possible chronons *data model, to support valid-time indeterminacy In the possible chronons data model, each event is represented with a set of possible chronons, delimiting when the event might have occurred, and a probability distribution over that set. We extend the TQuel query language with constructs that specify the user's credibility in the underlying valid-time data and the user's plausibility in the relationships among that data. We outline a formal tuple calculus semantics, and show that this semantics reduces to the determinate semantics on determinate data.*

1 Overview

A valid-time database records the history of an enterprise [Jensen et al. 1992]. It associates with each event a *timestamp* indicating when that event occurred. Often a user knows only approximately when an event happened. For instance, she may know that it happened "between 2 PM and 4 PM," "on Friday," "sometime last week," or "around the middle of the month." These are examples of *valid-time indeterminacy*. Information that is valid-time indeterminate can be characterized as "don't know when" information, or more precisely, "don't know *exactly* when" information. This kind of information has various sources, including the following.

- *granularity* — In perhaps most cases, the granularity of the database does not match the precision to which an event time is known. For example, an event time known to within one day and recorded on a system with timestamps in the granularity of a microsecond happened sometime *during* that day, but during which microsecond is unknown.
- *dating techniques* — Many dating techniques are inherently imprecise, such as radioactive and Carbon-14 dating.
- *future planning* — Projected completion dates are often inexactly specified, i.e., the project will complete three to six months from now.
- *unknown or imprecise event times* — In general, event times could be unknown or imprecise. For example, assume that we do not know when an individual was born. The individual's date of birth could be recorded in the database as either unknown (they were born between now and the beginning of time) or imprecise (they were born between now and 150 years ago).

Temporal database management systems should provide support for valid-time indeterminacy. In particular, users should be able to control, via query language constructs, the amount of indeterminacy present in derived information; and the query evaluator should accommodate valid-time indeterminacy in its processing. Query evaluation efficiency should remain high in the presence of valid-time indeterminacy, and it should not be affected at all in its absence.

This document describes the *possible chronons* data model. The model adds valid-time indeterminacy to TQuel [Snodgrass 1987]. TQuel is a strict superset of Quel, the query language for Ingres [Stonebraker et al. 1976]. TQuel has a complete, formal semantics which we extend to support valid-time indeterminacy. We could have extended SQL [Melton 1990]. While there are numerous proposed temporal extensions of SQL, none of these extensions have a complete, formal semantics. In addition, the temporal database research community has yet to adopt a common model for research purposes. Since Quel is equivalent to SQL in expressive power, our ideas can be applied to both languages.

The next section introduces an example that will be used throughout the paper. We then examine the representation of valid-time indeterminacy. After that, we explore what it means to retrieve information from a database with valid-time indeterminacy. Emphasis is placed on providing a simple and intuitive retrieval method. We outline syntactic and semantic extensions to TQuel to support retrieval of valid-time indeterminate information. The final sections trace related work, summarize our approach, and discuss future work.

2 Motivating Example

An example valid-time database is shown in Figure 1. This database models a single company with two warehouses and one airplane factory. The warehouses supply parts to the factory. Each warehouse keeps its own *Sent* relation, which is a history of parts shipments sent from the warehouse to the factory. The factory maintains the *In_Production* relation, which is a production history of airplanes built by the factory.

Valid-time indeterminacy naturally arises in both base relations and derived relations. It may surprise the reader to note that the *In_Production* base relation is a valid-time indeterminate relation. This is because the granularity of the *In_Production* relation is a month while that of the *Sent* and *Received* relations are just a single day (we are assuming an underlying timestamp granularity of one day). A month is an indeterminate value that represents a set of possible days. We know that production on an airplane started on some day in the indicated month, but we can't be sure which one. For this example, we assume that production is equally likely to have started or ended during any day in an indicated month; although, in general we allow nonuniform distributions.

The *Received* relation is not maintained by either the factory or a warehouse; rather it is a derived relation, the product of educated guesswork. Parts are shipped by truck from a

Sent_by_Trump(Lot#,Part)

Lot#	Part	Valid time (at)
23	wing strut	May 6
24	engine	June 4

Sent_by_Griffin(Lot#,Part)

Lot#	Part	Valid time (at)
30	wing strut	May 26
31	wing strut	June 9

In_Production(Model, Serial#)

		Valid time	
Model	Serial#	(from)	(to)
Centurion	AB33	March	June
Cutlass	Z19	June	July
Centurion	AB34	June	August
Caravan	FA2K	April	May

Received(Warehouse, Lot#, Part)

Warehouse	Lot#	Part	Valid time (at)	
Trump	23	wing strut	May 10 – May 29	e_1
Griffin	30	wing strut	May 30 – June 18	e_2
Trump	24	engine	June 8 – June 27	e_3
Griffin	31	wing strut	June 13 – July 2	e_4

Figure 1: An historical database

warehouse and arrive at the factory no earlier than 4 and no later than 24 days after they leave a warehouse. The *Received* relation is computed from each warehouse's *Sent* relation by adding a 4–24 day "fudge factor" to the valid-time attribute. The valid times in the *Received* relation are indeterminate; that is, we know roughly when the parts were received, but we do not know exactly which day they were received. We will assume that each possible day indicated by the recorded range of days is equally likely. For example, the batch of engines received from the Trump warehouse arrived on one of the days in the set {June 8, June 9, ..., June 27}, but we have no reason to favor one day rather than another.

In a database that supports valid-time indeterminacy, queries can make use of indeterminate information. Suppose that a few of the Centurion airplane owners have reported a faulty wing strut. Naturally, we would like to query the database to determine which warehouse(s) supplied the defective parts and, specifically, which lots are implicated (we give such a query in Section 5). In TQuel with valid-time indeterminacy, we could query to determine which received shipment of wing struts "overlaps" the production of a Centurion airplane. Overlap is the operation of temporal intersection.

There are two stages to determining an answer to a query. The first stage retrieves the data that is relevant to the query. The second stage constructs an answer that satisfies the constraints specified in the query. We provide separate controls for each stage.

Range credibility changes the information available to query processing. For instance, given a uniform distribution assumption, it is unlikely that production on the Centurion serial number AB33 began early in March; but more likely that it started by late March. A typical user might be interested in only those production times which are likely, late March to early June for the Centurion, ignoring those that are unlikely. In the possible chronons data model, the user can express this preference by selecting an appropriate range credibility value. The chosen range credibility potentially modifies every interval in a valid-time relation, restricting the range of each interval. Effectively, non-credible starting and terminating times are eliminated to the chosen level of credibility during query processing, allowing the user to control the quality of the information used in the query.

Ordering plausibility controls the construction of an answer to the query using the pool of credible information. For instance, a Centurion owner could query which shipment of wing struts plausibly arrived during production of his or her plane. Intuitively such a query relaxes the constraints on the relationship between the production times and the day a shipment was received from "absolutely sure of overlap?" to "is it probable that they overlap?" or even to "is it even remotely possible that they overlap?". The user selects the kind of overlap that she or he requires by setting an appropriate ordering plausibility value. It is probable that lot number 31 from the Griffin warehouse was received during production the Centurion with serial number AB33, but it is impossible to be absolutely sure that it did.

We believe that there is a natural division between indeterminacy in the data and indeterminacy in the query. The support for valid-time indeterminacy that we add to TQuel allows the user to control both kinds of indeterminacy. Range credibility massages the information from which a plausible answer to the query is constructed.

3 Extending the Data Model with Indeterminacy

In this section, we discuss how valid-time indeterminate events and intervals are represented in the data model.

3.1 Time

In the temporal database community, two basic time models have been proposed: the *continuous model*, in which time is viewed as being isomorphic to the real numbers, with each real number corresponding to a "point" in time, and the *discrete model*, in which time is viewed as being isomorphic to the integers [Clifford & Tansel 1985]. In the discrete model, the continuous time-line is partitioned into line segments. Each segment is called a *chronon* [Ariav 1986, Clifford & Rao 1987]. A chronon is the smallest duration of time that can be represented. We choose to use the discrete model.

We do not assume a specific *granularity* or chronon size; a chronon may be of any duration (e.g., nanoseconds, years, Chinese imperial dynasties). We believe that specifying the granularity should be left to the implementation rather than fixed in the data model. Our data model supports only a single chronon size, although multiple granularities can be handled by representing the indeterminacy explicitly.

We assume that every event occurs at a point in time. Because we are using a discrete model, a chronon represents a line segment rather than a point. Hence, we can only record that an event occurred *during* a particular chronon. Two events that occur during the same chronon may still occur at different *times*.

3.2 Indeterminate Events

An event is *determinate* if it is known when (i.e., during which chronon) it occurred. A determinate event cannot overlap two chronons. If it is unknown when an event occurred, but known that it did occur, then the event is *indeterminate*. The indeterminacy refers to the *time* when the event occurred, not whether the event occurred or not. Indeterminate events do not

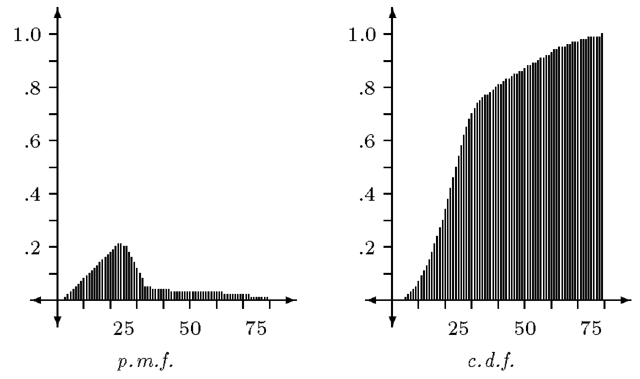

Figure 2: A "probably early" distribution

model the situation where it is unknown if an event occurred at all.

An indeterminate event is described by a *lower support chronon*, an *upper support chronon*, and a *probability mass function (p.m.f.)*. The support chronons delimit when the event occurred; it occurred no earlier than the lower support chronon and no later than the upper support chronon. Between the support chronons is a *period of indeterminacy*. The period of indeterminacy is a contiguous set of *possible chronons*. The event occurred during some chronon in this set, but during which is unknown. We use α_* to denote the lower support chronon for the indeterminate event α, and α^* to denote the upper support chronon.

In some situations not all the possible chronons are equally likely. For instance, it could be that the event most likely happened during the earliest chronon in the period of indeterminacy. The probability mass function gives the probability that the event occurred during each chronon. In the terminology of probability theory, this distribution is the *density function* for the event random variable. For an indeterminate event α, we define its p.m.f., P_α, by

$$P_\alpha(i) \;=\; \mathbf{Pr}[\alpha = i] \quad i \in \mathbb{Z}$$

where $\mathbf{Pr}[\alpha = i]$ is the probability that the event occurred during chronon i. All probability mass functions are considered to be independent; we make no provisions for joint, marginal, or dependent density functions. Note that, because we adopted the discrete model of time, a p.m.f. is a discrete, rather than a continuous, function. Figure 2 depicts a "probably early" mass function. The probability mass function for an indeterminate event is supplied by the user; the default distribution is uniform probability. For example, the mass function depicted in Figure 2 might illustrate the probability that a performer is "gonged" on *The Gong Show* (the performance is likely to end early). We will sometimes write the indeterminate event α as $([\alpha_*,\alpha^*],P_\alpha)$.

A useful function derived from the probability mass function is the *cumulative density function (c.d.f.)*, which is defined to be:

$$F_\alpha(i) = \mathbf{Pr}[\alpha \leq i] \;=\; \sum_{k \leq i} P_\alpha(k).$$

Intuitively, for each chronon the c.d.f. measures the probability that the event occurs sometime *before or during* a chronon. Figure 2 shows the c.d.f. for the "probably early" p.m.f.

	Warehouse	Lot#	Part	Valid time (at)
s_5	Trump	40	*unknown*	May 31
s_6	Griffin	70	*electrical*	May 31
s_7	Trump	41	{yoke, throttle}	May 31

Figure 3: Examples of value incompleteness

While the terminology used in the possible chronons data model suggests a difference between indeterminate and determinate events, it is instructive to note that indeterminate events can be used to model determinate events. A determinate event is modeled by an indeterminate event with a set of possible chronons that contains a single chronon. In this case, the probability that the event occurred during that single chronon is 1.

3.3 Indeterminate Intervals

An interval bounded by indeterminate events (called the *starting* and *terminating events*) is termed an *indeterminate interval*. An indeterminate interval could start during any chronon in the set of possible chronons of the starting event. Likewise, the indeterminate interval could end during any chronon in the set of possible chronons of the terminating event. Since it is unknown precisely when the starting or terminating events happen, it follows that it is unknown precisely when an indeterminate interval begins or ends.

An indeterminate interval represents a set of *possible intervals*, one of which is the "real" interval, but which is unknown. A single possible interval is obtained by choosing one possible chronon from each bounding indeterminate event's set of possible chronons. Every combination of chronons in the starting and terminating events' set of possible chronons is in the set of possible intervals.

For every indeterminate interval, every member of the set of possible chronons for the starting event must be before every member in the set of possible chronons in the terminating event. This ensures that every possible interval in an indeterminate interval is a valid interval. That is, a possible interval cannot terminate before it starts, as might happen if the sets of possible chronons overlapped. There is one exception to this maxim; the sets of possible chronons in the bounding events can overlap on a single chronon. As a result, each possible interval must span at least one chronon, and some possible intervals might span only that single chronon.

Thus far we have only considered indeterminate intervals bounded by indeterminate events. What of indeterminate intervals that have determinate events as one or both bounding events? Since indeterminate events can be used to model determinate events, no special provisions are needed to handle determinate bounding events.

3.4 Other Kinds of Indeterminacy

In the possible chronons data model, valid-time indeterminacy is orthogonal to other sources of incompleteness (c.f., [Motro 1990]). In particular, it can peacefully coexist with *value incompleteness*, where the value of an attribute (as opposed to a timestamp) is not fully known. For example, in the *Received* relation, we may be shipped a part which we have yet to identify (s_5 in Figure 3), has been partially identified (s_6 restricts the kind of part to belong to the specified class of parts), or has been narrowed down to a set of possibilities (s_7). While there are approaches that combine temporal and value incompleteness (e.g., [Gadia et al. 1992]), we advocate separating the various kinds of indeterminacy, so that users can choose the combination that is most appropriate for their application.

We turn now from the data model to the query semantics.

4 Review of TQuel

We assume that the reader is familiar with TQuel and the tuple calculus; we provide a quick review of TQuel's retrieve statement. The interested reader will find many examples as well as a complete description of the syntax and semantics elsewhere [Snodgrass 1993, Snodgrass 1987]. An example retrieve query that determines which `wing strut` shipments arrived during production of a `Centurion` airplane is shown in Figure 4. The retrieve has several components: the *target list*, specifying how the attributes of the relation being derived are computed from the attributes of the underlying relations; a *valid clause*, specifying the valid time of tuples in the target relation; a *where clause*, specifying a relationship that must be satisfied among the explicit attributes (those visible to the user) of the participating tuples; a *when clause*, specifying a relationship among the valid-time attributes of the participating tuples; and an *as of* clause that performs a rollback on the temporal database (not shown in Figure 4).

In the valid clause, a temporal expression consisting solely of *temporal constructors* specifies the valid time of tuples in the target relation. A temporal constructor chooses an event or interval that satisfies some constructor specific constraints. For example, the *First* temporal constructor chooses the earliest event from a pair of events. The temporal expression associated with the when clause is composed of temporal constructors, boolean connectives, and *temporal predicates*. A temporal predicate determines whether a pair of events or intervals satisfies some predicate specific constraints. For example, the *precede* predicate determines whether one event (or interval) is earlier than another. If so, the predicate evaluates to "true;" if not, it evaluates to "false."

5 Extensions to TQuel

This section proposes a syntax and semantics for extending TQuel's retrieve statement to support valid-time indeterminacy. A primary design goal is to make this extension a minimal extension. It will be shown in Section 5.8 that the new syntax and semantics preserves the meaning of all extant TQuel retrieve statements.

5.1 Syntactic Extensions for Valid-time Indeterminacy

We make two syntactic extensions to TQuel's retrieve statement, one to specify the range credibility and the other to specify the ordering plausibility. Details are presented elsewhere [Dyreson & Snodgrass 1992A].

Range credibility appears (optionally) in the range statement of an interval relation. The credibility applies independently to the starting and terminating events in the interval. It can be any integer value between 0 and 100 (inclusive). The credibility phrase has an initial default value of 100; this default value can be changed using a set statement. Range credibility is not applicable to event relations because removing indeterminacy from an indeterminate event might require partitioning the event's period of indeterminacy.

Ordering plausibility is the plausibility in the temporal ordering of the events that participate in the retrieval. The ordering plausibility may be specified either for the entire when predicate and valid constructor or for a particular temporal predicate

```
range of r is Received
range of p is In_Production with confidence 0
retrieve (WH = r.Warehouse, L# = r.Lot#, S# = p.Serial#)
  valid at r
  where p.Model = "Centurion" and r.Part = "wing strut"
  when r overlap p probably
```

Defective_Shipment_Candidates(WH, L#, S#)

WH	L#	S#	Valid time (at)		
Trump	23	AB33	May 10	–	May 29
Griffin	30	AB33	May 30	–	June 18
Griffin	31	AB34	June 13	–	July 2

Figure 4: A sample query and its result

or constructor, in which case it appears in parentheses immediately after the operator. The ordering plausibility is specified with an integer between 1 and 100 (inclusive). The plausibility phrases are optional and have an initial default value of 100, which can be changed using a set statement.

The retrieve statement in Figure 4 shows a plausibility value of "probably" for the *overlap* temporal predicate in the when clause. This is syntactic sugar for a plausibility value of 60.

5.2 Semantic Extensions

The semantic extensions to support valid-time indeterminacy involve the redefinition of several existing functions and relations and the introduction of new functions. Specifically, we redefine the temporal ordering relation to support ordering plausibility, we introduce two "shrink" functions to effect range credibility, we add an "adjust" function to the valid clause to ensure that only valid indeterminate intervals are constructed, and we redefine the coalescing operator, *Reduce*. In subsequent sections we consider each of these modifications in some detail. It is important, however, to observe that each function or relation that we redefine or add incorporates the determinate semantics. Support for valid-time indeterminacy is an extension of the determinate semantics rather than a replacement. Hence, the semantics of existing queries is left unchanged (this point is reiterated in Section 5.8).

5.3 Supporting Ordering Plausibility

To support ordering plausibility we redefine the ordering relation *Before*. The semantics of retrieve without indeterminacy is based on a well-defined ordering of the valid time events in the underlying relations [Snodgrass 1987]. Every temporal predicate and temporal constructor refers to this ordering to determine if the predicate is true or the constructor succeeds. A set of determinate events has a single temporal ordering. Given a temporal expression consisting of temporal predicates and temporal constructors, this ordering either satisfies the expression or fails to satisfy it.

A set of indeterminate events, however, typically has many possible temporal orderings. Some of these temporal orderings are plausible while others are implausible. The user specifies which orderings are plausible by setting an appropriate ordering plausibility value. We stipulate that a temporal expression is satisfied if there exists a plausible ordering that satisfies it.

In the determinate semantics, *Before* is the "<" relation

$<_{prob}$	e_1	e_2	e_3	e_4
e_1		100	100	100
e_2	0		83	94
e_3	0	13		70
e_4	0	3	26	

Figure 5: Table of $\alpha <_{prob} \beta$ for the events in *Received*

on event times. In the indeterminate semantics, the temporal ordering is given by the probabilistic ordering operator "$<_{prob}$" which is defined as follows. For any two indeterminate events, α and β

$$\alpha <_{prob} \beta = \lfloor 100 \times \mathbf{Pr}[\alpha < \beta] \rfloor$$

where

$$\mathbf{Pr}[\alpha < \beta] = \sum_{i<j} P_\alpha(i) \times P_\beta(j) \quad i, j \in \mathbb{Z}.$$

This formulation of the probabilistic ordering operator treats ordering probabilities that are between 0 and 1 (after scaling by 100) as 0. That is, it treats two events that have a small chance of occurring before each other as well-ordered in time. To distinguish the well-ordered case from this other case, we define the ordering probability to be 1 whenever its value as defined above, prior to taking the floor, is between 0 and 1. Hence, to evaluate every possible ordering, however improbable, an ordering plausibility of 1 suffices.

The probabilistic ordering operator assumes that there are no dependencies between the probabilities associated with indeterminate events. It cannot be used to accurately compute the probability of orderings such as $(\alpha <_{prob} \beta <_{prob} \eta)$. In the expression "($\alpha$ `precede` β) and (β `precede` η)" the two `precede`s are separately evaluated, returning boolean values that are subsequently anded. While this evaluation strategy is consistent with the determinate semantics, it is not equivalent to computing an ordering of $(\alpha <_{prob} \beta <_{prob} \eta)$.

Figure 5 shows the value of $<_{prob}$ for each pair of events in the relation *Received*, placed on a time-line in Figure 6. For instance, $e_2 <_{prob} e_3 = 88$.

To handle indeterminate events, we modify *Before* to include an additional initial parameter, the ordering plausibility γ. The value of γ can be any integer between 1 and 100 (inclusive). In general, higher (closer to 100) ordering plausibilities stipulate that more probable orderings should be considered plausible. The indeterminate *Before* is defined as follows.

$$Before(\gamma,\ \alpha,\ \beta) \iff \neg(\alpha\ is\ \beta) \wedge ((\alpha <_{prob} \beta) \geq \gamma)$$

An event is never *Before* itself, regardless of the value of γ. Two events are said to be *equivalent* if they have the same support chronons and the same probability mass functions. Two equivalent, but not identical, events may or may not be *Before* one another, depending on γ. To distinguish identical from equivalent events, each event appearing as an argument to *Before* is tagged with the tuple from which it originates. The tags are compared by the *Before* function. If the tags do not match, the binary infix operator $<_{prob}$ determines the discrete probability of one event occurring "before" another.

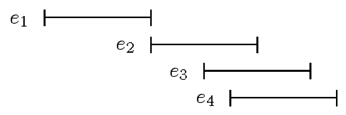

Figure 6: A pictorial representation of the *Received* event times

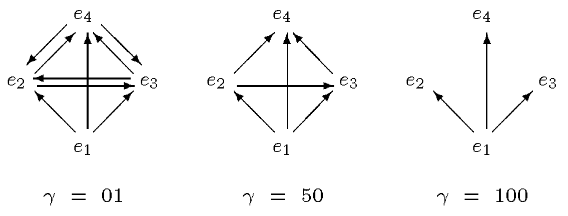

Figure 7: Ordering the events in *Received* depends on γ

The ordering relation among the events in the relation *Received* depends on the ordering plausibility, γ. The orderings given by differing values of γ are graphically depicted in Figure 7. Each directed edge in a graph indicates that the originating event is *Before* the terminating event. Some pairs of events are "indistinguishable," that is each occurs *Before* the other. If no edge connects two events, the events are "incomparable," neither occurs *Before* the other. Note that *Before*, for $\gamma \neq 100$, is not a typical ordering relation in that it is not transitive nor asymmetric, although it is always irreflexive (*Before* for $\gamma = 100$ is transitive, asymmetric, and irreflexive for nonequivalent events).

A useful generalization of *Before* is *Set_Before*. *Set_Before* is similar to *Before*, but operates on sets of events.

$$Set\_Before(\gamma,\ \alpha,\ \beta) \iff \forall x \in \alpha\ \forall y \in \beta\ Before(\gamma,\ x,\ y)$$

Set_Before stipulates that the set of events α is before the set of events β if every event in α is before every event in β, to the specified ordering plausibility.

The new ordering relations are used to redefine the temporal constructors and predicates. Below, we consider the *First* constructor in some detail since *First* is used in other constructors. Recall that *First* chooses the earliest event among a pair of events. But, with indeterminate events, choosing the earliest event among a pair of events is not always straightforward. In particular, for a given ordering plausibility, it could be that that neither event in a pair or events is earlier, or it could be that both are earlier. In the indeterminate semantics, $First(\gamma,\ \alpha,\ \beta)$ evaluates to

$$\begin{array}{ll} \alpha & \text{if } Set\_Before(\gamma,\ \alpha,\ \beta) \\ \beta & \text{if } Set\_Before(\gamma,\ \beta,\ \alpha) \\ \eta - \delta & \text{otherwise, where} \\ & \quad \eta = \alpha \bigcup \beta \text{ and} \\ & \quad \delta = \{x |\ x \in \eta \wedge \neg\exists y \in \eta\ (Before(\gamma,\ y,\ x))\}. \end{array}$$

To simplify discussion of *First*, consider the case where α and β each contain a single indeterminate event. Determining which event occurs first has several possible outcomes:

- only α is first,
- only β is first,
- both α and β are first (each is before the other; the events are indistinguishable), or
- neither α nor β is first (neither is before the other; the events are incomparable).

The first two outcomes are straightforward. The third outcome, that for indistinguishable events, is handled by the fact that *First* is nondeterministic; each event is generated separately and may result in a separate output tuple. For the final possible outcome, since neither event is before the other, *First* constructs the set containing both events. Other temporal constructors and temporal predicates will treat the set as a set of events with no *Before* relationships between the members. In general, all members in a set of events are pairwise incomparable. Below we show several temporal expressions composed of the *First* constructor and the result of each expression using the events from the relation *Received*.

$First(50,\ \{e_2\},\ \{e_3\}) = \{e_2\}$	(α is first)
$First(100,\ \{e_2\},\ \{e_1\}) = \{e_1\}$	(β is first)
$First(1,\ \{e_2\},\ \{e_3\}) = \{e_2\}$ and $\{e_3\}$	(both are first)
$First(100,\ \{e_2\},\ \{e_3\}) = \{e_2,\ e_3\}$	(incomparable events)

The *First* constructor can deduce the first event among a group of events, even when some of those events are incomparable (e_2, e_3, and e_4 are incomparable for a plausibility of 100 as shown in Figure 7), for example:

$$First(100,\ \{e_2\},\ First(100,\ \{e_3\},\{e_1\})) = \{e_1\}.$$

The *First* constructor also works when some of the events are indistinguishable (e_2, e_3, and e_4 are indistinguishable for a plausibility of 1), for example:

$$First(1,\ First(1,\ \{e_2\},\ \{e_1\}),\ First(1,\ \{e_3\},\ \{e_4\})) = \{e_1\}.$$

The redefinition of the *Last* temporal constructor is similar to that of *First* and is omitted to save space. The definitions of the other temporal constructors change little; a parameter for the plausibility is added to each, e.g.,

$$overlap(\gamma, \langle\alpha,\beta\rangle, \langle\eta,\delta\rangle) \quad = \quad \langle Last(\gamma,\alpha,\eta), First(\gamma,\beta,\delta)\rangle.$$

Contrast this with the determinate semantics for the overlap constructor:

$$overlap(\langle\alpha,\beta\rangle, \langle\eta,\delta\rangle) \quad = \quad \langle Last(\alpha,\eta), First(\beta,\delta)\rangle.$$

We are now in a position to redefine the temporal predicates. These definitions differ only slightly from the determinate semantics. A plausibility parameter is added to each predicate and *Set_Before* replaces *Before* since the temporal constructors now build sets of events rather than events. For example, in the determinate semantics, $precede(\langle\alpha,\beta\rangle,\langle\eta,\delta\rangle)$ is defined as $Before(Last(\alpha,\beta), First(\eta,\delta))$, while in the indeterminate semantics $precede(\gamma,\langle\alpha,\beta\rangle,\langle\eta,\delta\rangle)$ is defined as $Set\_Before(\gamma, Last(\gamma,\alpha,\beta), First(\gamma,\eta,\delta))$.

5.4 Supporting Range Credibility

Range credibility changes the data that is available for query evaluation. In general, range credibility is used to eliminate some possible intervals from an indeterminate interval. It does so by eliminating some possible chronons from both the starting and terminating events' set of possible chronons. To support range credibility we introduce two "shrink" functions: *Shrink_s* (shrink the *s*tarting event) and *Shrink_t* (shrink the *t*erminating event). The shrink functions compute a "shortened" version of an indeterminate event by shrinking its period of indeterminacy and modifying its probability mass function.

Shrink_s computes a "later" period of indeterminacy by removing some of the "earlier" chronons from the set of possible chronons. How many chronons to remove is governed by the first

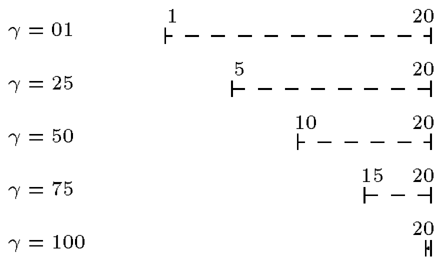

Figure 8: $Shrink\_s(\gamma, ([1,20), Uniform))$ for several values of γ

argument, γ, the range credibility. The value of γ is between 0 and 100 (inclusive). Every possible chronon that has a cumulative probability less than the level of credibility is removed. Higher values (closer to 100) of γ will remove more chronons from the set. $Shrink\_s(100,\alpha)$ will remove every chronon except the latest possible chronon in α. $Shrink\_s(0,\alpha)$ will leave α unchanged. Figure 8 shows the result of *Shrink_s* for several credibility values on the indeterminate event $\alpha = ([1,\ 20], Uniform)$. *Shrink_s* is defined as follows.

$$Shrink\_s(\gamma,\ ([\alpha_*,\ \alpha^*],\ P_\alpha)) = ([x,\ \alpha^*],\ P'_\alpha)$$

where x is constrained by

$$\begin{aligned}(\alpha_* \le x \le \alpha^* \wedge F_\alpha(x) \ge \gamma)&\\ \wedge \neg(\exists i)(x < i \le \alpha^* \wedge F_\alpha(i) = F_\alpha(x)&)\\ \wedge \neg(\exists j)(\alpha_* < j < x \wedge F_\alpha(x) > F_\alpha(j) \ge \gamma&)\end{aligned}$$

and P'_α is the new mass function, $P'_\alpha(i) = \frac{P_\alpha(i)}{1-F_\alpha(x)}$. Intuitively, the conditions on the function stipulate that the desired chronon is in a group of chronons with the same cumulative probability (the cumulative probability is the chance that the event occurs before or during the chronon in question). This group is the latest group such that the cumulative probability of all the chronons earlier than the group falls below γ while the cumulative probability of each chronon within the group matches or exceeds γ. The desired chronon is the latest chronon within this group. It is the latest rather than an arbitrary chronon so that repeated shrinks will make progress.

The function must also compute a new probability mass function since the old mass function might have assigned nonzero probability to chronons that are no longer in the period of indeterminacy. To construct the new mass function, the probability of each of the remaining chronons is scaled by the cumulative probability of the chopped chronons. The new mass function is a *conditional* density function. That is, the proabilities are conditioned by the fact that the period of indeterminacy is shrunk.

Shrink_t is similar to *Shrink_s*, but it removes the "late" chronons from an event's period of indeterminacy. The definition of this function is similar to that of *Shrink_s*.

With these two functions, it is possible to define the temporal constructor consisting entirely of a tuple variable associated with an interval relation.

$$interval(\gamma,\ t) = \langle\{Shrink\_s(\gamma,\ t_{from})\},\ \{Shrink\_t(\gamma,\ t_{to})\}\rangle$$

This function extracts the *from* timestamp from the tuple, shrinks it by γ to create a "later" set of possible chronons, extracts the *to* timestamp from the tuple, and shrinks it by γ to create an "earlier" set of possible chronons, thereby effecting the range credibility. If $\gamma = 100$, then all valid-time indeterminacy will be eliminated. The function then constructs an interval consisting of the pair of the starting event and the terminating event, each perhaps indeterminate.

5.5 Adjusting

The valid clause for a valid-time relation associates a valid time event or interval with each answer tuple. If the target relation is an event relation, one answer tuple is generated for each event in the set of events constructed by the valid clause. If the target relation is an interval relation, then the starting and terminating events are chosen from the starting and terminating set of events constructed by the valid clause. However, the selected events do not always form a valid interval since the events could have overlapping periods of indeterminacy (on more than a single chronon).

The *Adjust* function ensures that this condition is not violated by constructing a valid indeterminate interval from a pair of indeterminate events if it is plausible to do so. If the sets of possible chronons of the starting and terminating events overlap, the function will shrink the sets of possible chronons so that they do not overlap. The constructed indeterminate interval represents only a subset of all the possible intervals since the shrinking process eliminates some possible intervals. The maximum amount by which *Adjust* can shrink the sets of possible chronons is dictated by the ordering plausibility, *gamma*. We define *Adjust* as follows.

$$Adjust(\gamma,\ \alpha,\ \beta) = \langle Shrink\_t(\delta,\alpha), Shrink\_s(\delta,\beta)\rangle$$

where δ is the smallest value less than or equal to $(100 - \gamma)$ such that $Before(100, Shrink\_t(\delta,\alpha), Shrink\_s(\delta,\beta))$. When the starting event is entirely before the terminating event, this function simply returns the interval as is. If this is not the case, it attempts to isolate a plausibility, δ, that is the minimum amount each set of possible chronons needs to be shrunk by in order to construct a valid interval. The maximum amount each event is allowed to be shrunk is given by $(100 - \gamma)$ (if $\gamma = 100$, the *Before* should be true without any shrinking). If no such δ exists, then no interval is constructed because the construction would exceed the user chosen plausibility.

5.6 Coalescing

Tuples in TQuel relations are assumed to be coalesced, in that tuples with identical values for the explicit attributes (termed *value-equivalent tuples* [McKenzie & Snodgrass 1991]) neither overlap nor are adjacent in determinate valid time. However, the tuples could overlap in indeterminate valid time. The tuples produced by the retrieve statement are coalesced by the *Reduce* function. A new function *Reduce′* computes the minimal set of value-equivalent indeterminate tuples, i.e., the set for which there are no such tuples. Details are presented elsewhere [Dyreson & Snodgrass 1992A].

5.7 Semantics of the Example Query

At this point, the semantics of the retrieve statement have been specified. As an example, we trace the computation of the query given in Figure 4 on the database given in Figure 1. The query will result in three tuples, also shown in Figure 4. First, the extent of the intervals in *In_Production* is unchanged by the shrink functions because the query uses a range credibility of 0. The where clause eliminates every tuple from *In_Production* except the `Centurions`. Likewise, the where clause also eliminates every tuple from *Received* except the `wing strut` tuples.

The shipment of lot number 23 was definitely received during production of the `Centurion` serial number `AB33`; it satisfies the overlap with every plausibility. The other shipments might have been received. Lot number 30 satisfies the overlap for plausibilities lower than 65 because ([May 30, June 18], uniform) is *Before* ([June 1, June 30], uniform) for ordering plausibilities below 65. The other shipment, however, arrived too late in June to be considered plausible. It is plausible that lot number 31 arrived before the end of production only for ordering plausibilities of 28 or less.

For production of the `Centurion` serial number `AB34`, all the shipments arrived too early, except for lot number 31 from the `Griffin` warehouse.

5.8 Query Reducibility

An important feature of the extended syntax and semantics is that evaluation of a retrieve statement using the default plausibility and credibility (both are 100) on a valid-time database with indeterminate or determinate interval relations and determinate event relations is equivalent to evaluation of the retrieve statement with the previous semantics (which has no support for valid-time indeterminacy) on the corresponding "interval reduced" database without valid-time indeterminacy. We will call this property *query reducibility.* By an *interval reduced* database, we mean a valid-time database in which the interval indeterminacy has been removed by replacing each indeterminate interval with its determinate portion (every indeterminate interval has a determinate portion of at least one chronon). Query reducibility shows that the meaning of all extant TQuel queries and relations is preserved under the new semantics. It also shows that even if there is some indeterminacy in the database (i.e., if there are indeterminate interval relations), the user can choose to ignore it (this is the default choice).

Theorem *The extended semantics is query reducible to the previous, valid-time determinate, semantics.*

The proof is given elsewhere [Dyreson & Snodgrass 1992A].

6 Implementation

Our goal in implementing support for valid-time indeterminacy is to do so efficiently. The new or redefined functions discussed in the previous section, *Adjust*, *Shrink_s*, *Shrink_t*, *Set_Before*, *Before*, and *Reduce′*, are all executed in the "inner loop" of query processing. Significant slowdown of these operations would have a dramatic effect on the overall speed of query evaluation. Although implementing the new functions may appear costly, we have developed an efficient implementation based on heavy preprocessing of the probability information and approximating the actual computation when necessary [Dyreson & Snodgrass 1992A]. We have also developed a compact indeterminate event timestamp format (8 bytes for common indeterminate events) [Dyreson & Snodgrass 1992B]. In parallel with the theorem of query reducibility given in Section 5.8, conventional TQuel queries on determinate relations will incur no additional execution overhead under the new semantics. For queries on indeterminate relations, for a plausibility of 1 or 100 or a credibility of 0 or 100, the algorithms to support the new functions incur little overhead. These special, but common, credibility and plausibility levels indicate that the user has chosen either not to use any probabilistic information or to interpret probabilistic information as determinate. For these situations, the algorithms to support the new functions are straightforward and quite efficient.

7 Related Work

Despite the wealth of research in incomplete information databases, there are few efforts that address temporal incompleteness. Much of the previous research in incomplete information databases has concentrated on issues related to null values [Codd 1990, Date 1986, Vassiliou 1979, Zaniolo 1984]. Another primary research thrust has studied the applicability of fuzzy set theory to relational databases [Dubois & Prade 1988, Zemankova & Kandel 1985].

Our work can be seen as an extension of the Probabilistic Data Model (PDM) [Barbará et al. 1992]. In PDM, attribute values are sets with weights attached to each element. The weight is the probability that the corresponding element is *the* value of the attribute. Queries use the probabilistic representation in conjunction with a single user-given "confidence" to compute a result within the framework of the possible world semantics.

Information that is valid-time indeterminate is also similar to disjunctive information, especially in the context of deductive databases [Liu & Sunderraman 1990]. A set of possible chronons is of the exclusive-or variety of disjunctive information (only one disjunct is true) [Ola 1992].

Recently, the issue of *multiple time granularities*, e.g., knowing some events to the accuracy of seconds, other events to within a day, and yet other events to only within a year, has been examined [Ladkin 1987, Wiederhold et al. 1991]. These approaches generally convert mixed granularities to the coarsest granularity, taking into account the semantics of the time-varying domains. Our work refines this approach because we convert the coarser granularities to indeterminacy and preserve the semantics of the finest granularity.

Dutta uses a fuzzy set approach to handle *generalized temporal events* [Dutta 1989]. A generalized temporal event is a single event that has multiple occurrences. For example the event "Margaret's salary is high" may occur at various times as Margaret's salary fluctuates to reflect promotions and demotions.

Generalized bitemporal elements are defined somewhat differently in a more recent paper [Kouramajian & Elmasri 1992]. Bitemporal elements combine transaction time and valid time in the same temporal element [Jensen et al. 1992]. Since TQuel also supports transaction time, valid-time indeterminacy and generalized bitemporal elements differ mainly in their handling of valid time. In their model, both the upper bound and the lower bound on a valid time interval could be a set of non-contiguous possible chronons. Unlike valid-time indeterminacy, the upper and lower bound sets could intersect and no probabilities are used. Since there are no probabilities, the user in general is limited to querying for answers which are either "definite" or those which are "possible" (or combinations thereof). Historically, these alternatives have a well-defined meaning in incomplete information databases [Lipski 1979].

Another proposal intertwines support for value and temporal incompleteness [Gadia et al. 1992]. By combining the different kinds of incomplete information, a wide spectrum of attribute values are simultaneously modeled, including values that are completely known, values that are unknown but are known to have occurred, values that are known if they occurred, and values that are unknown even if they occurred. We feel that conflating different kinds of incompleteness in a single temporal relational database model prevents the user from picking and choosing the kind of incomplete information support that she desires.

In our approach, value, tuple, and temporal incompleteness are orthogonal. By combining valid-time indeterminacy with other kinds of incomplete information we can support each of the kinds of incomplete information found in Gadia et al., plus others (e.g., fuzzy value incompleteness). Another difference between our approach and theirs is that they make no use of probabilistic information. The user cannot express his or her credibility in the underlying data nor plausibility in the temporal relationships in the data.

Finally, the approach to valid-time indeterminacy espoused by Kahn and Gorry [Kahn & Gorry 1977] is reminiscent of those employed by the artificial intelligence community [Maiocchi & Pernici 1991]. In their model, events and intervals are specified relative to each other; only a subset are actually tied to the valid time line. An event may only be known to have occurred, say, between two other events. Their model is more general than the possible chronons data model, but also exhibits significant query processing overhead.

8 Summary and Future Work

This paper has extended the syntax and formal semantics of TQuel to support valid-time indeterminacy. This support provides the user with two controls on the retrieval process, range credibility and ordering plausibility. We have extended the range statement with an optional with clause to specify range credibility and extended the retrieve statement to specify ordering plausibilities. Range credibility changes the information available to query processing. It eliminates some possible, but unlikely intervals from an indeterminate interval until the desired quality of information is reached. Ordering plausibility controls the construction of an answer to a query using the pool of credible information. A temporal expression is satisfied if there exists a plausible ordering (to the level specified by the user) that satisfies it. The approach has an intuitive semantics, is orthogonal to those proposed by others to handle value incompleteness and generalized events, refines previously proposed techniques to handle multiple granularities of time, and has a practical implementation.

The result is an expressive extension to TQuel to support valid-time indeterminacy. The extension is also "transparent" to the user who does not use the added query language support for indeterminacy. The extended semantics and implementation both reduce to the previous semantics and implementation under the default credibility and plausibility.

A useful extension of the current work would be to use spans instead of values to express credibility and plausibility. For instance, the user could specify a range credibility of a "day" or a "year," causing sets of possible chronons in the specified relations to be shrunk to the most probable day or year. Similarly, the ordering plausibility could make use of durations. The user could constrain retrieval to tuples that "overlap March, 1984" to "within a year" (this has been termed a "band join" [DeWitt et al. 1991] or a "fuzzy temporal equi-join" [Leung & Muntz 1991]). Both possibilities can be seen as extensions of the present paper.

This paper only considers the retrieve statement. The update statements (append, delete, and replace) can also be extended in an analogous manner. Extending temporal aggregates [Snodgrass et al. 1993] is more challenging; the goal, shared with this paper, is to simultaneously maximize the expressive power of the language and the efficiency of query evaluation. Finally, when a consensus temporal extension to SQL is available, we will apply our approach to that language to add valid-time indeterminacy.

Acknowledgements

This work was supported in part by NSF grant ISI-8902707 and IBM contract #1124. Comments on earlier drafts of this paper by Christian Jensen, Michael Soo, and Nick Kline helped improve the presentation.

References

[Ariav 1986] Ariav, G. "A Temporally Oriented Data Model." *ACM Transactions on Database Systems*, 11, No. 4, Dec. 1986, pp. 499–527.

[Barbará et al. 1992] Barbará, D., H. Garcia–Molina and D. Porter. "The Management of Probabilistic Data." *IEEE Transactions on Knowledge and Data Engineering*, 4, No. 5, Oct. 1992, pp. 487–502.

[Clifford & Tansel 1985] Clifford, J. and A.U. Tansel. "On an Algebra for Historical Relational Databases: Two Views," in *Proceedings of ACM SIGMOD International Conference on Management of Data*. Ed. S. Navathe. Association for Computing Machinery. Austin, TX: May 1985, pp. 247–265.

[Clifford & Rao 1987] Clifford, J. and A. Rao. "A Simple, General Structure for Temporal Domains," in *Proceedings of the Conference on Temporal Aspects in Information Systems*. AFCET. France: May 1987, pp. 23–30.

[Codd 1990] Codd, E. F. "Missing Information," in The Relational Model for Database Management: Version 2. Addison-Wesley Publishing Company, Inc., 1990. Chap. 8–9.

[Date 1986] Date, C.J. "Null Values in Database Management," in Relational Database: Selected Writings. Reading, MA: Addison-Wesley, 1986. Chap. 15. pp. 313–334.

[DeWitt et al. 1991] DeWitt, D., J. Naughton and D. Schneider. "An Evaluation of Non-Equijoin Algorithms," in *Proceedings of the Conference on Very Large Databases*. 1991, pp. 443–452.

[Dubois & Prade 1988] Dubois, D., and H. Prade. "Handling Incomplete or Uncertain Data and Vague Queries in Database Applications," in Possibility Theory: An Approach to Computerized Processing of Uncertainty. New York and London: Plenum Press, 1988. Chap. 6. pp. 217–257.

[Dutta 1989] Dutta, S. "Generalized Events in Temporal Databases," in *Proceedings of the Fifth International Conference on Data Engineering*. Los Angeles, CA: Feb. 1989, pp. 118–126.

[Dyreson & Snodgrass 1992A] Dyreson, C. E. and R. T. Snodgrass. "Historical Indeterminacy." Technical Report TR 91-30a. Computer Science Department, University of Arizona. Revised Feb. 1992.

[Dyreson & Snodgrass 1992B] Dyreson, C. E. and R. T. Snodgrass. "Time-stamp Semantics and Representation." Technical Report TR 92-16a. Computer Science Department, University of Arizona. July 1992.

[Gadia et al. 1992] Gadia, S.K., S. Nair and Y.-C. Poon. "Incomplete Information in Relational Temporal Databases," in *Proceedings of the Conference on Very Large Databases*. Vancouver, Canada: Aug. 1992.

[Jensen et al. 1992] Jensen, C.S., J. Clifford, S.K. Gadia, A. Segev and R.T. Snodgrass. "A Glossary of Temporal Database Concepts." *SIGMOD Record*, 21, No. 3, Sep. 1992.

[Kahn & Gorry 1977] Kahn, K. and G. A. Gorry. "Mechanizing Temporal Knowledge." *Artificial Intelligence*, (1977), pp. 87–108.

[Kouramajian & Elmasri 1992] Kouramajian, V. and R. Elmasri. "A Generalized Temporal Model." Tech. Report. University of Texas at Arlington. Feb. 1992.

[Ladkin 1987] Ladkin, P. "The Logic of Time Representation." PhD. Dissertation. University of California, Berkeley, Nov. 1987.

[Leung & Muntz 1991] Leung, T.Y. and R. Muntz. "Temporal Query Processing and Optimization in Multiprocessor Database Machines." Technical Report CSD-910077. Computer Science Department, UCLA. Nov. 1991.

[Lipski 1979] Lipski, W., Jr. "On Semantic Issues Connected with Incomplete Information Databases." *ACM Transactions on Database Systems*, 4, No. 3, Sep. 1979, pp. 262–296.

[Liu & Sunderraman 1990] Liu, K.C. and R. Sunderraman. "Indefinite and Maybe Information in Relational Databases." *ACM Transactions on Database Systems*, 15, No. 1, Mar. 1990, pp. 1–39.

[Maiocchi & Pernici 1991] Maiocchi, R. and B. Pernici. "Temporal Data Management Systems: A Comparative View." *IEEE Transactions on Knowledge and Data Engineering*, 3, No. 4, Dec. 1991, pp. 504–524.

[McKenzie & Snodgrass 1991] McKenzie, E. and R. Snodgrass. "Supporting Valid Time in an Historical Relational Algebra: Proofs and Extensions." Technical Report TR–91–15. Department of Computer Science, University of Arizona. Aug. 1991.

[Melton 1990] Melton, J. (ed.) "Solicitation of Comments: Database Language SQL2." American National Standards Institute, Washington, DC, 1990.

[Motro 1990] Motro, A. "Imprecision and incompleteness in relational databases: survey." *Information and Software Technology*, 32, No. 9, Nov. 1990, pp. 579–588.

[Ola 1992] Ola, A. "Relational Databases with Exclusive Disjunctions," in *Proceedings of the Eighth International Conference on Data Engineering*. Tempe, AZ: Feb. 1992, pp. 328–336.

[Snodgrass 1993] Snodgrass, R. "An Overview of TQuel," in Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, 1993. Chap. 6. pp. 45.

[Snodgrass et al. 1993] Snodgrass, R., S. Gomez and E. McKenzie. "Aggregates in the Temporal Query Language TQuel." *IEEE Transactions on Knowledge and Data Engineering*, (1993), to appear.

[Snodgrass 1987] Snodgrass, R. T. "The Temporal Query Language TQuel." *ACM Transactions on Database Systems*, 12, No. 2, June 1987, pp. 247–298.

[Stonebraker et al. 1976] Stonebraker, M., E. Wong, P. Kreps and G. Held. "The Design and Implementation of Ingres." *ACM Transactions on Database Systems*, 1, No. 3, Sep. 1976, pp. 189–222.

[Vassiliou 1979] Vassiliou, Y. "Null values in database management–a denotational semantics approach," in *Proceedings of ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery. New York: ACM Press, May 1979, pp. 162–169.

[Wiederhold et al. 1991] Wiederhold, G., S. Jajodia and W. Litwin. "Dealing with Granularity of Time in Temporal Databases," in *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*. Trondheim, Norway: May 1991.

[Zaniolo 1984] Zaniolo, C. "Database Relations with Null Values." *Journal of Computer and System Sciences*, 28 (1984), pp. 142–166.

[Zemankova & Kandel 1985] Zemankova, M. and A. Kandel. "Implementing Imprecision in Information Systems." *Information Sciences*, 37 (1985), pp. 107–141.