[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.

[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. 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 1982] Snodgrass, R. "Monitoring Distributed Systems: A Relational Approach." PhD. Dissertation. Computer Science Department, Carnegie Mellon University, Dec. 1982.

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

[Snodgrass et al. 1989] Snodgrass, R., S. Gomez and E. McKenzie. "Aggregates in the Temporal Query Language TQuel." Technical Report TR–89–26. Department of Computer Science, University of Arizona. Nov. 1989.

Acknowledgement

This work was supported in part by NSF grant ISI-8902707.

References

[Barbará et al. 1990] Barbará, D., H. Garcia–Molina and D. Porter. "A Probabilistic Relational Data Model," in *Proceedings of the International Conference on Extending Database Technology: Advances in Database Technology — EDBT '90.* Venice, Italy: Mar. 1990, pp. 60–74.

[Barton 1989] Barton, B. "Dinosaurs, Dinosaurs." New York, NY: Thomas Crowell, 1989.

[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 & White 1990] Date, C. J. and C. J. White. "A Guide to DB2." Reading, MA: Addison-Wesley, 1990. Vol. 1, 3rd edition.

[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 1992] Dyreson, C. E. and R. T. Snodgrass. "Timestamp Semantics and Representation." TempIS TR 33. Computer Science Department, University of Arizona. Feb. 1992.

[Gadia et al. 1991] Gadia, S.K., S. Nair and Y.-C. Poon. "Incomplete Information in Relational Temporal Databases." Technical Report Technical Report. Department of Computer Science, Iowa State University. Dec. 1991.

[Hsu & Snodgrass 1991] Hsu, S.H. and R.T. Snodgrass. "Optimal Block Size for Repeating Attributes." TempIS Technical Report No. 28. Department of Computer Science, University of Arizona. Dec. 1991.

and ordering plausibility. We have augmented the create and modify statements to specify which relations incorporate historical indeterminacy, extended the range statement with an optional with clause to specify range credibility, extended the retrieve statement to specify ordering plausibilities, and added variants to the set statement to specify default plausibilities and credibilities. The approach is orthogonal to those proposed by others to handle value indeterminacy and generalized events, refines previously proposed techniques to handle multiple granularities of time, and is theoretically sound.

We showed how indeterminate events with a uniform probability distribution can be represented in only two words in most cases; for user-defined distributions the representation is only three words in most cases. We outlined efficient implementations of the five extended functions required by the altered semantics. The result is an expressive yet practical extension to TQuel to support historical 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.

One important assumption we make throughout is that tuples are row-independent, with no information shared between indeterminate tuples. All the other approaches we are aware of that utilize probabilities to model various flavors of incompleteness make this assumption as well. We also assume that the indeterminate events can be modeled by contiguous sets of possible events. We do not support noncontiguous sets which could model indeterminate events such as "it happened yesterday morning or this morning". We exploit both of these assumptions to achieve efficiency in representation and in query processing.

One possible problem is that plausibility and credibility values may have little intuitive meaning for novice or infrequent users. A useful extension of the current work would be to provide the user with more intuitive and flexible tools to express credibility and plausibility. We are considering using spans instead of values. For instance, the user could specify a range credibility of a "day" or a "year", causing sets of possible events 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. 1989] 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 historical indeterminacy.

Generalized temporal elements are defined somewhat differently in a more recent paper [Kouramajian & Elmasri 1992]. Generalized temporal elements combine transaction time and valid time in the same temporal element. Since TQuel also supports transaction time, historical indeterminacy and generalized temporal 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 noncontiguous possible events. Unlike historical 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]. Generalized valid times are composed of valid times by the operators of alternation (only one valid time applies) and/or union (both valid times could apply). Alternation is used to model alternative temporal scenarios. We provide no capability for "generalizing" valid times to handle such scenarios.

Another proposal intertwines support for value, temporal, and tuple incompleteness [Gadia et al. 1991]. 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 historical 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 indeterminacy). 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.

We note that there is little discussion in any of the aforementioned papers of implementation aspects. We feel that both efficient representations and efficient query processing algorithms are crucial.

Finally, the approach to historical 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 tuples 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 historical indeterminacy. This support provides the user with two controls on the retrieval process, range credibility

1990]. In PDM, attribute values are sets with weights attached to each element. The weights are the probabilities that the element is *the* attribute value. Queries use the probabilistic representation in conjunction with a single user-given "confidence" to compute a result. The novelty in our work can be seen in the methods used to retrieve the incomplete information and how that information is represented. The large number of possible events for an indeterminate event precludes storing each possible event separately, as would be done in PDM. The encoding we developed, using sets of possible events and event probability distributions, is space efficient. In addition, our use of both credibility and plausibility values permits greater flexibility and finer control in query evaluation. These techniques are a product of the unique nature of the type of incomplete information, historical indeterminacy, that we investigated.

Information that is historically indeterminate is also similar to disjunctive information, especially in the context of deductive databases [Liu & Sunderraman 1990]. Disjunctive information is a collection of facts, one (or more) of which is true. A set of possible events is of the exclusive-or variety of disjunctive information (only one disjunct is true) [Ola 1992]. Historical indeterminacy, like PDM, differs from the above investigations because the alternatives are "weighted" and the weights are integrated into the query semantics.

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 in three ways: we support arbitrary starting and terminating chronons, rather than requiring that these chronons be aligned on a set of predefined granularities; we support user-defined probability distributions; and we support user-specified range credibilities and ordering plausibilities.

Dutta uses a fuzzy set approach to handle temporal events [Dutta 1989]. His model allows a single event to have multiple occurrences. For example suppose that, in an employee database, we want to record when "Margaret's salary is high." The event "Margaret's salary is high" may occur at various times as Margaret's salary fluctuates to reflect promotions and demotions. The incompleteness enters in the definition of what is meant by "high"; it is not a crisp predicate. In Dutta's model all the possibilities for "high" are represented in a *generalized* event and the user selects some subset according to his or her interpretation of "high".

This contrasts sharply with the task of encoding the type of information we have characterized as historically indeterminate. We view events as having a single occurrence. The time that an indeterminate event occurred is a set of alternatives, one and only one of which is the actual time. Every member in a fuzzy set is always possible, to a greater or lesser extent depending on the degree of membership, but always possible (although some fuzzy databases stipulate by fiat that only one member is possible [Dubois & Prade 1988]).

Our approach and that of Dutta's model different kinds of temporal incompleteness. We feel that a probabilistic approach is better suited to modeling historical indeterminacy as formulated in this paper, and that fuzzy set approaches like Dutta's are better suited to modeling generalized events. The two approaches are orthogonal, and the user may pick the one(s) most appropriate to her application.

The resulting semantics for the optimized version of *Adjust* approximate the semantics presented in this paper in that events are always shrunk by the maximum amount possible, i.e. $(100 - \gamma)$, reducing the cost to 21 adds and 12 multiplies. The modified semantics may result in determinate portions that are too large; an example of this was shown in Figure 15.

Perhaps the costliest addition is the new *Before* relation for $\gamma < 100$. The algorithm for *Before* first tests whether one event is entirely *Before* the other. If so, *Before* is satisfied for any plausibility value. We anticipate that it is common that the sets of possible events of two indeterminate events do not overlap. But if they do overlap, the algorithm uses an efficient "pivoting" technique. A chronon that is in the set of possible events of indeterminate event α and in the set of possible events of indeterminate event β is a pivot. The chronons in α that are before the pivot are *Before* the chronons in β that are after the pivot. Thus, the starting probability of α at the pivot multiplied by the terminating probability of β at the pivot is part of the probability that $\alpha \leq_{prob} \beta$. At most N pivots must be chosen (where N is the size of the smallest set of possible events) to compute $\alpha \leq_{prob} \beta$. However, the algorithm for *Before* performs the pivoting on the superchronons. Hence, in the worst case, 256 pivots at a cost of 2056 multiplies and 5392 arithmetic/comparison operations must be performed, although on average only 2 pivots are necessary at a cost of 24 multiplies and 58 arithmetic/comparison operations.

The implementation of *Reduce'* is little changed from that of the original *Reduce*. The indeterminate events with the earliest and latest extent must be computed, but this adds only two timestamp comparisons to each step. Of greater consequence is that *Reduce'* must deal with more tuples that *Reduce* because of the nondeterminism in *First* and *Last*. But at most $T \times (2^{|First|+|Last|} - 1) - 1$ more tuples will be in the input to *Reduce'* (where $|First|$ is the number of *First* operations in the tuple calculus statement and T is the number of tuples in the input to *Reduce*). In the tuple calculus semantics shown in section 5.5 for the query in Figure 9, at most $3T - 1$ more tuples will be produced. When evaluated on the database shown in Figure 1, no additional tuples are produced.

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].

In previous work, the second author proposed modeling indeterminate events with sets of possible events, and assumed a uniform distribution [Snodgrass 1982]. *Before* was extended to possibly return the value *unknown*, necessitating an extension to a three-valued logic. Our current approach allows a probability distribution to be associated with each indeterminate event, and does not require a three-valued logic.

Our work can be seen as an extension of the Probabilistic Data Model (PDM) [Barbará et al.

6.2 Query Evaluation Algorithms

The five functions discussed in Section 5, *Adjust*, *Shrink_r*, *Shrink_l*, *Before*, and *Reduce'*, make use of normalized probability distributions. A normalized distribution is a discrete approximation of a user-given probability distribution. The distribution is sampled at $N+1$ points. Between the points, the curve of the user's probability distribution is linearly approximated. We currently use a sample size of $N = 256$. The sampled points are *superchronon* boundaries in the normalized distribution. Thus, the normalized distribution is a discrete distribution of 256 values. A normalized distribution is mapped to an indeterminate event by mapping each superchronon to a group of chronons in the event's set of possible events.

The starting and terminating probabilities for a normalized distribution could be computed "on the fly" from the normalized distribution and the offsets. However, this computation can be expensive. We heavily preprocess the normalized distribution to improve the efficiency of computing the starting and terminating probabilities. In particular, the starting and terminating superchronons for the credibilities 0 to 100 are precomputed and stored. The starting and terminating probabilities for each integral superchronon value are also precomputed and stored. The precomputation ensures that the computed approximations are within an absolute error of 2^{-8} at each point between the sampled points. The precomputation phase produces a 2.5KB table for each normalized distribution. If there are many normalized distributions, the tables can be stored on disk and paged into a distribution cache as needed during evaluation of a query. There are a few other precomputed tables storing valuable constants which occupy approximately 700 bytes, and must be pinned in main memory.

For a plausibility of 1 or 100 or a credibility of 0 or 100, the algorithms to support the new functions do not use the precomputed information. These special, but common, credibility and plausibility levels indicate 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 efficient.

For other credibility and plausibility values, the algorithms to support the new functions make intensive use of the precomputed information. The shrink functions use table lookup to find the precomputed starting and terminating superchronons for a given credibility (with the appropriate scaling by the left and right offsets). The computed superchronon must then be mapped to a particular chronon value. The *Shrink_l* and *Shrink_r* functions each cost 5 16-bit multiplies and 11 other arithmetic/comparison operations (one addition and one subtraction are on 64-bit quantities; the remaining nine operations are on 8-bit and 16-bit integers).

The *Adjust* function constructs an interval from two indeterminate events by shrinking the events as needed. The test to determine if shrinking is needed is a single comparison operation (*Before*$(100, \alpha, \beta)$ may be implemented by comparing the last chronon in α to the first chronon in β). If shrinking is needed, then each event is shrunk by $(100 - \gamma)$ to determine if the maximum amount of shrinking constructs a valid interval. If a valid interval can be constructed, then *Adjust* uses binary search to find the minimum amount, δ, by which it should shrink each event. The binary search identifies δ in $\lceil \log_2(100 - \gamma) \rceil$ steps. Each step costs one *Shrink_l* and one *Shrink_r*. The total cost is 147 adds and 84 multiplies. As an optimization, the search for δ could be eliminated.

example, *Shrink_l* must compute the new probability for each chronon in the shrunken set of possible events in the terminating indeterminate event, which can span a large number of chronons. We employ a representation that makes the shrink operators very fast, effectively by delaying the computation of the new probability distribution.

6.1 Representing Indeterminacy

The historical algebra is attribute-value timestamped; each timestamp a set of chronons termed a *temporal element*. This representation must be augmented to allow storage of a set of indeterminate events or intervals, each comprised of a set of possible events or intervals [Hsu & Snodgrass 1991]. A temporal element is thus essentially a set of sets of chronons. We can store temporal elements as linked lists of indeterminate events or intervals. An indeterminate event is represented by a set of possible events and an event probability distribution [Dyreson & Snodgrass 1992]. The set of possible events representation occupies either two or four 32-bit words. The four word representation uses 64-bit timestamps for both the starting and terminating chronons in the interval. The 64-bit timestamp can model times that range over the age of the universe to a granularity of a second or times that range over 28,000 years to the granularity of a microsecond. The two word representation is a compact version adequate for most sets of possible events commonly encountered.

The representation of a probability distribution has three parts, the name of a *normalized distribution* (including identifying parameters), a *left offset*, and a *right offset*; occupying somewhat less than 32 bits in toto. For relations with the distribution or duration stored in the schema as discussed in Section 4.2, this additional word may not be necessary. The probability distribution of events that have the same normalized distribution are distinguished by how much of the distribution has been chopped off by the shrink functions. The left offset is the percentage of the normalized distribution from the left that has been eliminated by *Shrink_l*. The right offset is the percentage of the normalized distribution from the right that has been eliminated by *Shrink_r*. If part of the distribution is eliminated, the rest of the distribution must be scaled by the eliminated portion to keep the sum of the probabilities over every chronon at 1. Since we expect that most indeterminate events will use the uniform probability distribution, we optimized representation of this distribution. The representation of the uniform distribution uses a single flag bit. It is unnecessary to record either offset with the uniform distribution because the starting and terminating probabilities are linear functions regardless of the offset values.

In summary, an indeterminate event data structure occupies, at worst, five words. We expect, however, that a compact representation of three words for nonuniform distributions and two words for uniform distributions and for intensional distributions will be sufficient for most applications. As a comparison, the current DB2 timestamp representation is 2.5 words ([Date & White 1990]), the commercial Ingres ([Stonebraker et al. 1976]) representation is three words, and the proposed SQL2 ([Melton 1990]) representation is six words, all with a significantly shorter extent and without any historical indeterminacy.

PROOF. The outline of what to prove is illustrated in Figure 18. The only differences between C and C' are the new and redefined functions *Adjust*, *Before*, *Shrink_l*, *Shrink_r*, and *Reduce'*. In the evaluation of C', the *Shrink* functions are applied first. For the default range credibility of 100, *Shrink_l* and *Shrink_r* reduce each bounding event's set of possible events to a singleton set. Thus, the default range credibility for the *Shrink* functions essentially removes all historical indeterminacy from a database with indeterminate interval relations. The reduced database contains only determinate events and intervals.

Determinate events have a well-defined ordering, hence the new *Before* relation is almost equivalent to the previous relation. For determinate events α and β, evaluation of $Before(\delta, \alpha, \beta)$ for any plausibility value δ will be satisfied only if the chronon for α is less than the chronon for β or if α and β are the same determinate event. If α and β are equivalent (they span the same chronon) but are not the same determinate event, then both are *Before* each other under the definition of *Before* in the extended semantics. This will cause the *First* and *Last* temporal constructors (which use *Before*) to generate two equivalent events. Each may contribute to a tuple that is a candidate for the target relation. But since the events differ in neither their set of possible events nor their probability distribution, two equivalent events can only contribute to tuples which are duplicates of each other. Two duplicate tuples are pared to one unique tuple by the *Reduce'* function since duplicate determinate tuples are value-equivalent and overlap in determinate valid time.

In the previous semantics, the *Adjust* function simply returned the interval if the starting chronon was *Before* the ending chronon. With determinate events, the extended semantics for *Adjust* reduce to the previous semantics, because the shrink functions in the definition have no effect on determinate bounding events and $Before(100, \alpha, \beta)$ is equivalent to the previous $Before(\alpha, \beta)$.

Finally, the new definition of *Reduce'* coalesces value-equivalent tuples that are adjacent or overlap in determinate time exactly as the old definition. ∎

6 Implementation

The foundation for implementing historical indeterminacy is the historical algebra [McKenzie & Snodgrass 1991]. Changes to the historical algebra to support historical indeterminacy are isolated to the representation of temporal elements and to the derivation operator (which performs temporal selection and projection, i.e., the when and valid clauses) and the rollback operator (which retrieves the tuples from the appropriate historical state, implementing the *interval* function described in Section 5.2). The derivation operator already uses *Adjust*, *Before*, and *Reduce*. However, the new definition of each of these operators must be implemented. Finally, the *Shrink_l* and *Shrink_r* must be added to the processing of the rollback operator to effect range credibility.

Each of these changes necessitates the construction of new algorithms and new data structures. Our goal in developing these algorithms is to meet the efficiency challenge. The rollback and derivation operators are executed in the "inner loop" of query processing. Significant slowdown of these operators would have a dramatic effect on the overall speed of query evaluation. Yet to support historical indeterminacy, implementing the new functions appears to be costly. For

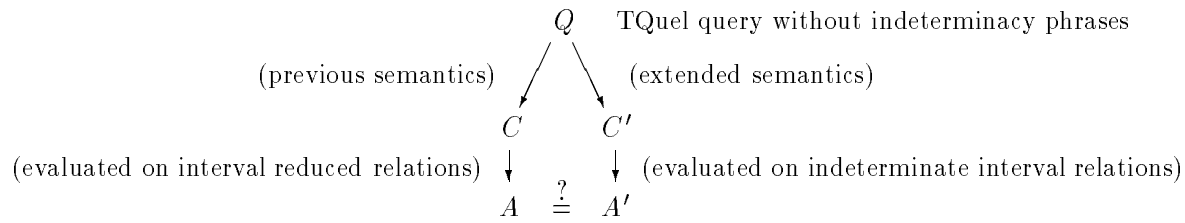


Figure 18: Does the evaluation of C produce the same answers as the evaluation of C'?

a δ of 76, which is greater than $100 - \gamma = 50$.

For the Early Jurassic tuple, the valid clause will result in an event. *Last*(50, T, EJ_s) generates just one event, T, while *First*(50, T, EJ_t) also generates the single event T. *Adjust*(50, T, T) detects that it is really trying to adjust an event (an event is always *Before*(100, ,) itself), so it succeeds and returns the interval ⟨T, T⟩. The *beginof* function strips the starting event from the interval because the result is an event relation. Hence, the query will result in one tuple, shown in Figure 9 on page 12. If the ordering plausibility in the when clause were to be set to 50 instead of 05, the query would generate the same result.

5.6 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 an historical 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 historical indeterminacy) on the corresponding "reduced" database without historical indeterminacy. We will call this property *query reducibility*. By an *interval reduced* database, we mean an historical 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 her default choice).

More specifically, under the extended semantics, a retrieve statement without credibility or plausibility phrases, Q, will be translated to a tuple calculus statement, C, of the form described earlier in this paper. Under the previous semantics, Q will be translated to a tuple calculus statement, C', of the form discussed in Section 5. We claim that if every event relation participating in Q is determinate (but every interval relation need not be), then C is query reducible to C', that is, evaluation of C is equivalent to evaluation of C'.

Theorem *The extended semantics is query reducible to the previous, historically determinate, semantics.*

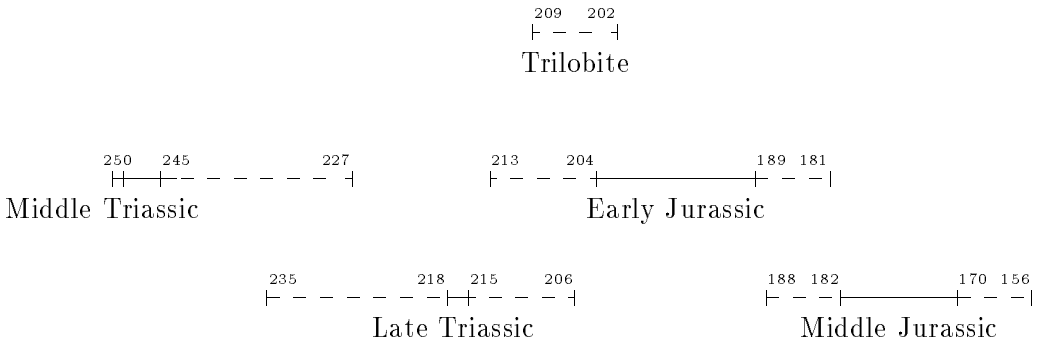


Figure 16: The overlap relationships among the "shrunken" epochs and the Trilobite

$\leq_{prob}$	T	LT_s	LT_t	EJ_s	EJ_t
T		0	13	26	100
LT_s	100		100	100	100
LT_t	91	0		72	100
EJ_s	77	0	36		100
EJ_t	0	0	0	0	

Figure 17: Table of $\alpha \leq_{prob} \beta$ for the relevant events in `Fossils` and `Classifications`

that is possible, then an additional conjunct is required: $\wedge \neg(u[r+1]$ *is* $u[r+2])$.

Let's trace this expression on the database given in Figure 1 on page 2. First, the extent of the intervals in `Classifications` is changed. A schematic of the overlap relationships among the Trilobite fossil and the shrunken indeterminate intervals of `Classifications` is shown in Figure 16. The where clause eliminates every tuple from `Fossils` except the Trilobite fossil.

Both the Middle Triassic and Middle Jurassic tuples from `Classifications` are immediately eliminated by `f` ***overlap***`(05) c` in the when clause because they do not overlap the Trilobite fossil at all.

That leaves both the Late Triassic and Early Jurassic tuples. For the sake of brevity we will use abbreviations for events throughout the rest of this example (i.e. the terminating event of the Late Triassic is abbreviated LT_t while the starting event is abbreviated LT_s). To help us unravel what happens during query processing, Figure 17 shows a table of the relevant $\leq_{prob}$ relationships between the event in the Trilobite tuple and the events in the Late Triassic and Early Jurassic tuples. The Late Triassic tuple satisfies the overlap in the when clause because *Before*(05, T, LT_t) is true. Similarly, the Early Jurassic tuple satisfies the overlap in the when clause because *Before*(05, EJ_s, T) is also true.

However the valid clause for the Late Triassic tuple will not result in an event because *Adjust* is unable to construct a plausible interval. *Last*(50, T, LT_s) generates only the event T (T is always *Before* LT_s) while *First*(50, T, LT_t) generates only the event LT_t. But *Adjust*(50, T, LT_t) requires

that overlap in determinate time are coalesced. It also stipulates that the extent of the resulting coalesced indeterminate interval is maximal. The coalesced interval extends from the earliest to the latest indeterminate event in the set of value-equivalent tuples that contribute to the coalescing. The second conjunct ensures that the starting event is constructed from an actual indeterminate event in the set of value-equivalent tuples that contribute to the coalescing. Finally, the third conjunct ensures that the terminating event is also manufactured from an actual indeterminate event in the set of value-equivalent tuples that contribute to the coalescing.

One feature of *Reduce'* is that it, rather arbitrarily, selects one particular history as the history of an historical object. For example, if two indeterminate events for value-equivalent tuples overlap, one event will be selected for inclusion in the history and the other will be discarded. Alternatively, we could have defined *Reduce'* to construct a new indeterminate event that would contain every possible event in both of the overlapping indeterminate events and a combined probability distribution. We did not adopt this approach because it is unclear whether or not probabilities from overlapping indeterminate events in value-equivalent tuples should be pooled in the construction of a new event. Through projection, initially unrelated tuples may become value-equivalent. There is no reason to pool the indeterminate event probabilities for such tuples because such probabilities are also unrelated.

5.5 Semantics of the Example Query

At this point, the semantics of the retrieve statement have been specified. As a review, the following is the tuple calculus semantics of the query in Figure 9, using a default range credibility of 25 and a default ordering plausibility of 50, as specified using a set statement, and chosen for illustrative purposes.

$$
\begin{aligned}
&\mathit{Reduce'}(\{u^{3+1} \mid (\exists f)\,(\exists c)\,(\mathit{Fossils}(f) \wedge \mathit{Classifications}(c) \\
&\quad \wedge u[1] = f[1] \wedge u[2] = c[1] \wedge u[3] = c[2] \\
&\quad \wedge u[4] = \mathit{beginof}(\mathit{Adjust}(50,\ \mathit{Last}(50,\ f[2],\ \mathit{Shrink_l}(25, c[3])), \\
&\qquad\qquad\qquad \mathit{First}(50,\ f[2],\ \mathit{Shrink_r}(25,\ c[4])))) \\
&\quad \wedge f[1] = \text{“Trilobite”} \\
&\quad \wedge \mathit{Before}(05,\ f[2],\ \mathit{Shrink_r}(25,\ c[4])) \wedge \mathit{Before}(05,\ \mathit{Shrink_l}(25,\ c[3]),\ f[2])
\end{aligned}
$$

We have chosen this example because it illustrates both a temporal constructor and a temporal predicate. The `overlap` appearing in the valid clause (a constructor) generates the *Adjust* on lines three and four. The `overlap` in the when clause (a predicate) generates line six. The default range credibility generates the *Shrink* function invocations.

There is one subtlety that should be mentioned. If the `valid during` variant is used, then the valid clause for some queries might evaluate to an event $\langle \alpha,\ \alpha \rangle$ rather than to an interval. If

$$
\begin{aligned}
overlap(\gamma,\ \langle\alpha,\ \beta\rangle,\ \langle\eta,\ \delta\rangle) &= Adjust(\gamma,\ Last(\gamma,\ \alpha,\ \eta),\ First(\gamma,\ \beta,\ \delta)) \\
extend(\gamma,\ \langle\alpha,\ \beta\rangle,\ \langle\eta,\ \delta\rangle) &= Adjust(\gamma,\ First(\gamma,\ \alpha,\ \eta),\ Last(\gamma,\ \beta,\ \delta)) \\
First(\gamma,\ \alpha,\ \beta) &= \begin{cases} \alpha & \text{if } Before(\gamma,\ \alpha,\ \beta) \\ \beta & \text{if } Before(\gamma,\ \beta,\ \alpha) \end{cases} \\
Last(\gamma,\ \alpha,\ \beta) &= \begin{cases} \beta & \text{if } Before(\gamma,\ \alpha,\ \beta) \\ \alpha & \text{if } Before(\gamma,\ \beta,\ \alpha) \end{cases} \\
beginof(\alpha,\ \beta) &= \langle\alpha,\ \alpha\rangle \\
endof(\alpha,\ \beta) &= \langle\beta,\ \beta\rangle
\end{aligned}
$$

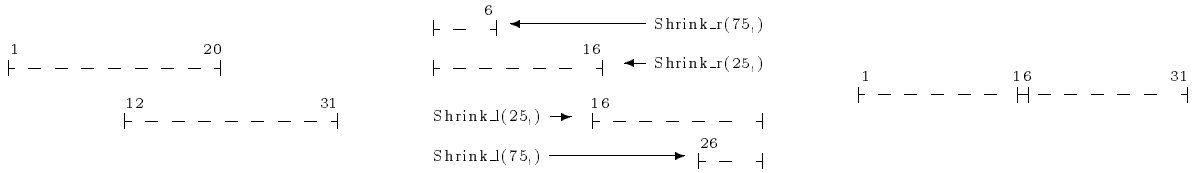
Note that some of the temporal constructors are nondeterministic. For example *First* takes two indeterminate events and determines which probably occurs first. Both events (or neither) could occur first. If neither occurs first, then those events do not participate in the computation of an output tuple. In the actual implementation, the "neither" event raises an exception that terminates the evaluation of the temporal constructor. The last two temporal constructors do not depend on γ, and so retain their original definitions.

5.4 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.

$$
\begin{aligned}
Reduce'(R) = \{t \mid\ & \forall r \in R \quad [(t \equiv r \wedge overlap(100,\ valid(r),\ valid(t))) \rightarrow \\
& \qquad (beginof(r) = ([\alpha_1,\ \alpha_m],\ P_\alpha) \\
& \qquad \wedge\ endof(r) = ([\beta_1,\ \beta_n],\ P_\beta) \\
& \qquad \wedge\ beginof(t) = ([\delta_1,\ \delta_k],\ P_\delta) \\
& \qquad \wedge\ endof(t) = ([\eta_1,\ \eta_j],\ P_\eta) \\
& \qquad \wedge\ \delta_k \leq \alpha_m \ \wedge\ \beta_1 \leq \eta_1 \\
& \qquad \wedge\ \delta_1 \leq \alpha_1 \ \wedge\ \beta_n \leq \eta_j)] \\
& \wedge \exists u \in R \quad [t \equiv u \wedge overlap(100,\ valid(u),\ valid(t)) \\
& \qquad \wedge \exists \delta(Shrink_r(\delta,\ beginof(u)) = beginof(t))] \\
& \wedge \exists v \in R \quad [t \equiv v \wedge overlap(100,\ valid(v),\ valid(t)) \wedge \\
& \qquad \wedge \exists \gamma(Shrink_l(\gamma,\ endof(v)) = endof(t))]\}
\end{aligned}
$$

Reduce' computes the minimal set of value-equivalent tuples, i.e., the set for which there are no such tuples. The first conjunct ensures that valid time intervals of value-equivalent tuples in R



(a) Initially (b) Shrinks for $\delta = 25$ and $(100 - \gamma) = 75$ (c) Constructed Interval

Figure 15: The steps for $Adjust(25, ([1,20], \mathit{Uniform}), ([12,31], \mathit{Uniform}))$ from (a) to (c)

If the sets of possible events of the starting and terminating events overlap, the function will shrink the sets of possible events so that they do not overlap. The shrinking process eliminates some possible intervals. Thus the constructed indeterminate interval represents only a subset of all the possible overlap intervals. The maximum amount that *Adjust* can shrink the sets of possible events by is dictated by the value of γ.

The definition of $Adjust(\gamma,\ \alpha,\ \beta)$ is as follows.

$$Adjust(\gamma,\ \alpha,\ \beta) = \begin{cases} \langle \alpha,\ \beta \rangle & \text{if } \mathit{Before}(100,\ \alpha,\ \beta) \\ \langle \mathit{Shrink_r}(\delta, \alpha), \mathit{Shrink_l}(\delta, \beta) \rangle & \text{if } \mathit{Before}(100, \mathit{Shrink_r}(\delta, \alpha), \mathit{Shrink_l}(\delta, \beta)) \\ & \quad \wedge \neg \mathit{Before}(100,\ \alpha,\ \beta) \end{cases}$$

where $\delta \leq (100 - \gamma)$ satisfies

$$\mathit{Before}(100, \mathit{Shrink_r}(\delta, \alpha), \mathit{Shrink_l}(\delta, \beta)) \wedge \neg(\exists \chi < \delta)(\mathit{Before}(100, \mathit{Shrink_r}(\chi, \alpha), \mathit{Shrink_l}(\chi, \beta))).$$

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 events 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 of the answer. Figure 15 shows two events, how they overlap, and the interval the *Adjust* function constructs from those events. As shown in the figure, shrinking by the maximum possible amount, $100 - \gamma$ or 75, could yield a bigger determinate interval than is needed.

With *Adjust* one can define the semantics of the temporal constructors.

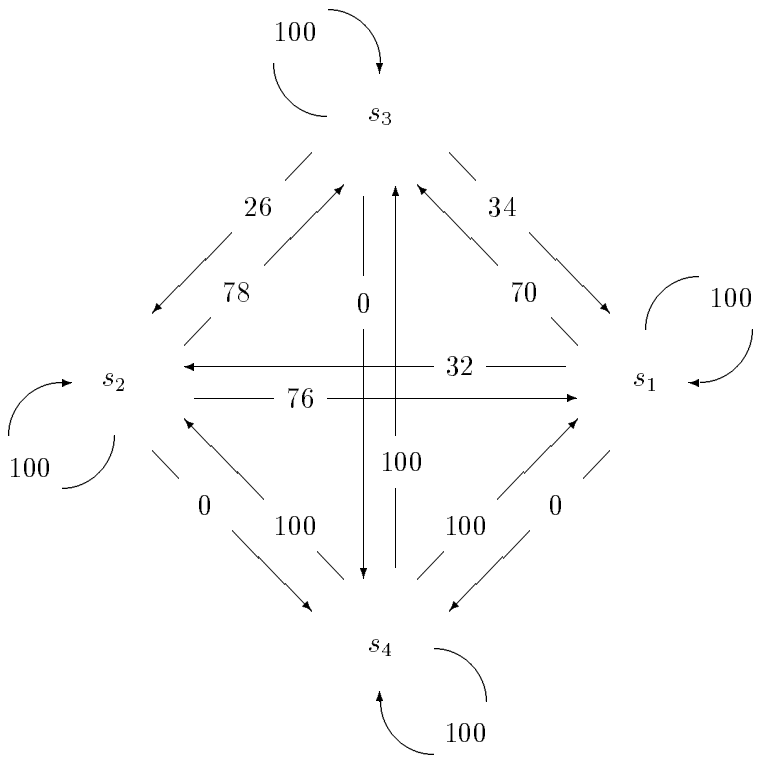


Figure 13: Orderings of the events in `Fossils` by *Before* for several values of γ

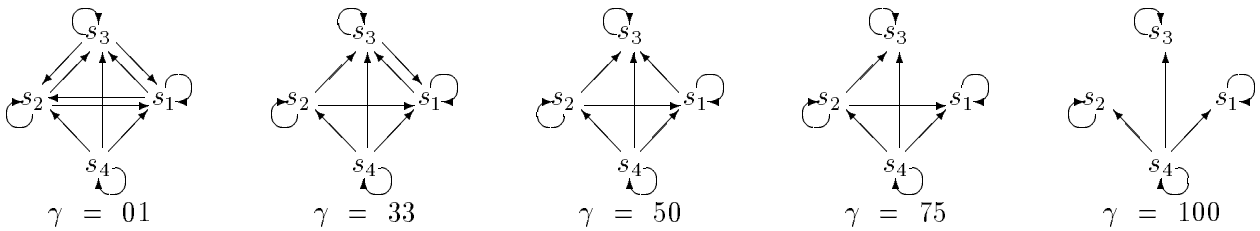


Figure 14: Orderings of the events in `Fossils` by *Before* for several values of γ

$\leq_{prob}$	s_1	s_2	s_3	s_4
s_1		32	70	0
s_2	76		78	0
s_3	34	26		0
s_4	100	100	100	

Figure 12: Table of $\alpha \leq_{prob} \beta$ for the events in `Fossils`

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 will 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 also 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 $(\alpha \leq_{prob} \beta \leq_{prob} \eta)$.

Figure 12 shows the value of $\leq_{prob}$ for each pair of events in the relation `Fossils`. For instance, $s_1 \leq_{prob} s_3 = 70$. The table is graphically depicted in Figure 13. Each directed edge in a graph indicates that the originating event is *Before* the terminating event. The label on the edge is the plausibility in the relationship. The ordering relation for the events in `Fossils` depends on the value of γ. The orderings given by differing values of γ are graphically depicted in Figure 14. 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 neither transitive nor asymmetric, although it is always reflexive (*Before* for $\gamma = 100$ *is* transitive, asymmetric, and reflexive for nonequivalent events).

The new ordering relation is used to redefine the temporal predicates.

$$
\begin{aligned}
\mathit{precede}(\gamma,\ \langle\alpha,\ \beta\rangle,\ \langle\eta,\ \delta\rangle) &= \mathit{Before}(\gamma,\ \beta,\ \eta) \\
\mathit{overlap}(\gamma,\ \langle\alpha,\ \beta\rangle,\ \langle\eta,\ \delta\rangle) &= \mathit{Before}(\gamma,\ \alpha,\ \delta) \wedge \mathit{Before}(\gamma,\ \eta,\ \beta) \\
\mathit{equal}(\gamma,\ \langle\alpha,\ \beta\rangle,\ \langle\eta,\ \delta\rangle) &= \mathit{Before}(\gamma,\ \alpha,\ \eta) \wedge \mathit{Before}(\gamma,\ \eta,\ \alpha) \\
&\quad \wedge \mathit{Before}(\gamma,\ \beta,\ \delta) \wedge \mathit{Before}(\gamma,\ \delta,\ \beta)
\end{aligned}
$$

To define the temporal constructors, we need a new function, *Adjust*. *Adjust* is used to construct "good" intervals. In a database of determinate events, the only constraint that an interval must satisfy is that the starting event precede the terminating event. That is, the determinate interval $\langle\alpha,\ \beta\rangle$ must satisfy the relationship $\mathit{Before}(\alpha,\ \beta)$. With indeterminate events, however, this constraint by itself is insufficient. An indeterminate interval is valid only if the sets of possible events do not overlap (or overlap on at most a single chronon). But the sets of possible events of two indeterminate events that are in the relationship $\mathit{Before}(\gamma,\ \alpha,\ \beta)$ might overlap 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.

indeterminacy will be eliminated. The function then constructs an interval consisting of the pair of the starting event and the ending event, each perhaps indeterminate.

As mentioned in Section 4.2 the semantics of the temporal constructor consisting solely of a tuple variable associated with an event relation is unchanged. To simplify the semantics of the temporal constructors and predicates to be presented shortly, we represent the event α (determinate or indeterminate) as the pair $\langle \alpha,\ \alpha \rangle$.

$$event(t) = \langle t_{at},\ t_{at} \rangle$$

This function extracts the *at* timestamp from the tuple and constructs an interval consisting of two copies of the resulting event.

5.3 Supporting Ordering Plausibility

To support ordering plausibility we introduce a new ordering relation. The semantics of retrieve without indeterminacy are based in part on a well-defined ordering of the valid time events in the underlying relations [Snodgrass 1987]. The ordering is given by the *Before* relation (*Before* is the "$\leq$" relation on the event times). In particular, the semantics of the temporal predicates and constructors are all defined in terms of *Before*. 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). The membership property for the relation *Before* is defined as follows.

$$Before(\gamma,\ \alpha,\ \beta) = \begin{cases} TRUE & (\alpha\ is\ \beta) \vee ((\alpha \leq_{prob} \beta) \geq \gamma) \\ \\ FALSE & \text{otherwise} \end{cases}$$

An event is defined to be *Before* itself, for all values of γ. Two events are said to be *equivalent* if their sets of possible events span the same chronons and they have the same probability distribution. Two equivalent, but not identical, events may or may not be *Before* one another, depending on γ. Since each event appearing as an argument to *Before* originates in an underlying tuple, we can tag these timestamps with their origin, and compare the tags in the *Before* function.

If the tags do not match, the binary infix operator $\leq_{prob}$ determines the discrete probability of one event occurring "before" another. For any two independent indeterminate events, $\alpha = ([\alpha_1,\ \alpha_m],\ P_\alpha)$ and $\beta = ([\beta_1,\ \beta_n],\ P_\beta)$,

$$\alpha \leq_{prob} \beta = \lfloor 100 \times (\sum_{E_i=\alpha_1}^{\alpha_m} \sum_{E_j=\beta_1}^{\beta_n} (\texttt{if}\ E_i \leq E_j\ \texttt{then}\ P_\alpha(E_i) \times P_\beta(E_j)\ \texttt{else}\ 0)) \rfloor$$

where the possible events are ordered by the "$\leq$" operator on the integers. This formulation of the probabilistic less than or equal to operator treats ordering probabilities that are between 0 and

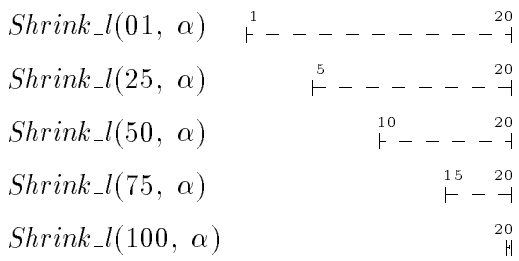


Figure 11: $Shrink_l(\gamma,\ ([1,20],\ Uniform))$ for several values o f γ

that repeated shrinks will make progress. The new density function scales the probability of each of the remaining chronons by the cumulative probability of the chopped chronons, so that the probabilities sum to one. The new function is a conditional density function. That is, the proabilities are conditioned by the fact that the set of possible events was shrunk. The definition is complicated by the fact that both the representation and the probability distribution function are discrete.

Shrink_l is similar to *Shrink_r*, but it removes the "early" members from an event's set of possible events. Figure 11 shows the result of *Shrink_l* for several values of γ applied to the same indeterminate event as the previous figure. $Shrink_l(\gamma,\ \alpha)$ is defined as follows.

$$Shrink_l(\gamma,\ ([\alpha_1,\ \alpha_n],\ P_\alpha)) =\ ([\alpha_l,\ \alpha_n],\ P'_\alpha)$$

where

$$\begin{aligned}(\exists\alpha_l)[&(\alpha_1 \le \alpha_l \le \alpha_n \wedge SP_\alpha(\alpha_l) \ge \gamma)\\ &\wedge \neg(\exists\alpha_i)(\alpha_i > \alpha_l \wedge SP_\alpha(\alpha_i) = SP_\alpha(\alpha_l))\\ &\wedge \neg(\exists\alpha_j)(\alpha_l > \alpha_j \wedge SP_\alpha(\alpha_l) \ne SP_\alpha(\alpha_j) \ge \gamma)]\end{aligned}$$

and P'_α is density function over $[\alpha_k,\ \alpha_n] : P'_\alpha(\alpha_i) = P_\alpha(\alpha_i) \times 100/(100 - SP_\alpha(\alpha_{l-1}))$.

The intuitive meaning of this function is similar to that of *Shrink_r*.

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

$$interval(\gamma,\ t) = \langle Shrink_l(\gamma,\ t_{from}),\ Shrink_r(\gamma,\ t_{to})\rangle$$

This function extracts the *from* timestamp from the tuple, shrinks it by γ to create a "later" set of possible events, extracts the *to* timestamp from the tuple, and shrinks it by γ to create an "earlier" set of possible events, thereby effecting the range credibility. If $\gamma = 100$, then all historical

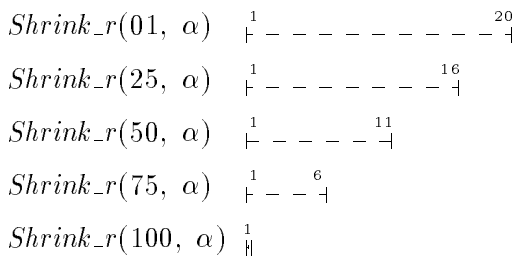


Figure 10: *Shrink_r*(γ, α) for event $\alpha = ([1,\ 20],\ \mathit{Uniform})$ and for several values of γ

5.2 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 events from both the starting and terminating events' set of possible events. To support range credibility we introduce two "shrink" functions: *Shrink_r* and *Shrink_l*. In general, the shrink functions compute a "shortened" version of an indeterminate event by shrinking its set of possible events and modifying its probability distribution.

Shrink_r computes an "earlier" set of possible events by removing some of the "late" members from the set of possible events. How many members to remove is governed by the first argument, γ, the range credibility. Every possible event whose terminating probability fails to match or exceed the level of credibility is removed. Higher values (closer to 100) of γ will remove more members from the set. *Shrink_r*$(100, \alpha)$ will remove every member except the earliest possible event in α. *Shrink_r*$(0, \alpha)$ will leave α unchanged. Figure 10 shows the result of *Shrink_r* for several values of γ on an indeterminate event $\alpha = ([1,\ 20], \mathit{Uniform})$. The function is defined as follows.

$$\mathit{Shrink_r}(\gamma,\ ([\alpha_1,\ \alpha_n],\ P_\alpha)) =\ ([\alpha_1,\ \alpha_r],\ P'_\alpha)$$

where

$$\begin{aligned}(\exists\alpha_r)[&(\alpha_1 \le \alpha_r \le \alpha_n \wedge TP_\alpha(\alpha_r) \ge \gamma)\\ &\wedge \neg(\exists\alpha_i)(\alpha_i < \alpha_r \wedge TP_\alpha(\alpha_i) = TP_\alpha(\alpha_r))\\ &\wedge \neg(\exists\alpha_j)(\alpha_r < \alpha_j \wedge TP_\alpha(\alpha_r) \ne TP_\alpha(\alpha_j) \ge \gamma)]\end{aligned}$$

and P'_α is the new density function over $[\alpha_1,\ \alpha_r]$: $P'_\alpha(\alpha_i) = P_\alpha(\alpha_i) \times 100/(100 - TP_\alpha(\alpha_{r+1}))$.

Intuitively, the conditions on the function stipulate that the desired chronon is in a group of chronons with the same terminating probabilities. This group is the latest group such that the terminating probability of all the chronons later than the group falls below γ while the terminating probability of the each chronon within the group matches or exceeds γ. The desired chronon is the earliest chronon within this group. It is the earliest rather than an arbitrary chronon so

5 Semantic Extensions

We now turn to the semantics of the extensions; Section 6 will discuss their implementation.

5.1 Determinate TQuel semantics

The semantics for TQuel associates a tuple calculus statement with each TQuel retrieve statement, ensuring that each construct has clear and unambiguous meaning [Snodgrass 1987]. Tuple relational calculus statements are of the form $\{t^i \mid \Psi(t)\}$, where the variable t denotes a tuple of arity i and $\Psi(t)$ is a first-order predicate calculus expression containing only one free tuple variable t. The tuple calculus statement for the skeletal TQuel retrieve statement

```
range of t_1 is R_1
...
range of t_k is R_k
retrieve (t_{i_1}.D_{i_j}, ..., t_{i_r}.D_{j_r})
      valid during υ
      where ψ
      when τ
```

is

$$
\begin{aligned}
&\mathit{Reduce}(\{u^{n+2} \mid (\exists t_1) \cdots (\exists t_k)\,(R_1(t_1) \wedge \cdots \wedge R_k(t_k) \\
&\quad \wedge u[1] = t_{i_1}[j_1] \wedge \cdots \wedge u[r] = t_{i_r}[j_r] \\
&\quad \wedge u[r+1] = \mathit{beginof}(\Phi_\upsilon) \wedge u[r+2] = \mathit{endof}(\Phi_\upsilon) \\
&\quad \wedge \Psi' \\
&\quad \wedge \Gamma_\tau) \\
&\quad \})
\end{aligned}
$$

Line 1 comes from the range statements. Line 2 is constructed from the target list. Φ_υ, appearing in line 3, is a function over the valid time attributes of a subset of the tuple variables. The function constructs a valid time interval using the temporal constructors given in the valid clause. Line 4 is constructed from the where clause. Γ_τ, appearing in line 5, is a predicate over the valid time attributes of a subset of the tuple variables; it uses the temporal predicates and constructors given in the when clause. The *Reduce* function ensures that tuples with identical values for all explicit attributes that overlap in valid time or are contiguous are *coalesced* into a single resulting tuple. For a retrieve statement that specifies an event relation (`valid at`), the superscript on u in line 1 is $n + 1$ and the *endof* in line 3 is omitted.

```
set default range credibility = 50
set default ordering plausibility = 50
retrieve (Name = f.Creature, Period = c.Period, Epoch = c.Epoch) with plausibility 83
        valid at f overlap c with plausibility 65
        where f.Name = "Trilobite"
        when f overlap(05) c and c(100) precede(100) |175 million years ago|
               with plausibility 75
```

Figure 8: Expressing ordering plausibility

```
retrieve (Name = f.Creature, Period = c.Period, Epoch = c.Epoch)
        valid at f overlap c
        where f.Name = "Trilobite"
        when f overlap(05) c
```

			Valid time	
Name	Period	Epoch	(at)	Distribution
Trilobite	Jurassic	Early	209–202	Uniform

Figure 9: A sample query and its result

clauses are provided for those infrequent queries in which the default range or ordering plausibility is not appropriate. If a plausibility phrase appears after the target list, it overrides the default plausibility for that query only. The when clause ordering plausibility phrase applies to the entire when clause unless superseded by a constructor or predicate plausibility. The valid clause ordering plausibility phrase applies to the entire valid clause unless superseded by a constructor or predicate plausibility. The temporal constructor and temporal predicate ordering plausibilities phrases apply only to that predicate or constructor.

The retrieve statement in Figure 9 shows a plausibility value (05) for the temporal predicate overlap. When this query is applied to the database shown in Figure 1, the relation shown in Figure 9 is computed. Intuitively, the query will determine, within certain plausibility and credibility levels, the geologic epochs during which the Trilobite was formed. The where clause selects the tuples from `Fossils` that are Trilobites (there is only one). The when clause selects those tuples from `Classifications` that overlap the Trilobite fossil with a plausibility of 05 (in this example the Late Triassic tuple and Early Jurassic tuple both qualify). Finally, the valid clause determines when, within plausibility, the Trilobite fossil overlapped the geologic epoch. It turns out that only the Early Jurassic epoch results in a valid event, as computed by the valid clause. We will discuss in detail in Section 5.5 how this query is evaluated to obtain this result.

specification of distributions is particularly helpful in optimizing both the representation and query processing, and is also of intuitive value to the user.

The default range credibility and ordering plausibility are each 100. The defaults can be changed using the set statement.

```
set default range credibility = 50
set default ordering plausibility = 33
```

This is very useful when a group of queries is to be made at a particular credibility level, or when the credibility is to be specified for a novice.

Range credibility appears (optionally) in the range statement. Below are two TQuel range statements with range credibilities.

```
range of f is Fossils with credibility 50
range of c is Classifications with credibility 25
```

The range credibility is the credibility in each valid time in the specified 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 phrases are optional, in which case the default credibility is used.

The range credibility initially ascribed to a tuple variable in a range statement for an interval relation with indeterminacy may be overridden for a particular use of the tuple variable by following the tuple variable name with the new range credibility in parentheses. An example will be given shortly.

Range credibility is not applicable to event relations because removing indeterminacy from an indeterminate event might require partitioning the event.

We extend the syntax of the retrieve statement to allow the user to express a level of plausibility in the answer constructed by retrieve. The ordering plausibility may be specified either for the entire when predicate and valid constructor or for a particular temporal predicate 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 credibility phrases are optional. The two (not extremely useful) statements in Figure 8 use all possible variants, as well as an override of `c`'s range credibility to 100, originally specified as 50. The constant event `|175 million years ago|` is also indeterminate; the starting and terminating chronons, as well as the probability distribution function, are inferred from the specified constant.

If no ordering plausibility phrase appears, the default plausibility, specified by the most recent set statement, is used. We anticipate that this will be the case most of the time; the multiple with

may be composed of logical connectives, temporal constructors, and temporal predicate operators. *Temporal constructors* are unary or binary operators that take one event, two events, or two intervals as arguments and return an event or an interval. The unary constructors are ***begin of*** and ***end of***. They return the starting and ending event of an interval, respectively. The binary constructors are ***overlap*** and ***extend***. ***overlap*** returns the period when two intervals intersect in time while ***extend*** is analogous to the temporal union of two intervals. *Temporal predicate operators* are binary infix operators that take events or intervals as arguments and return a Boolean value. The three temporal predicate operators are ***precede***, ***overlap*** and ***equal***. Note that ***overlap*** is overloaded; it is both a predicate and a constructor.

In the example query, pairs of Trilobite tuples and the `Classifications` with which they overlap are candidates for the target relation. The valid clause constructs the valid time for the tuples in the target relation. In this case, the valid time of each target tuple is the overlap between the Trilobite fossils and the geologic classifications. The other attributes in the target relation are specified by the target list: the `Name`, the `Period` and the `Epoch`. The valid clause specifies either a tuple variable or a temporal constructor; ***valid at*** specifies that an event relation is being derived, and ***valid during*** specifies a derived interval relation. An as of clause is not required when the query is evaluated on historical relations.

4.2 Extensions for Historical Indeterminacy

We make four extensions to TQuel, one to indicate that a relation is indeterminate, one to specify the range credibility, one to specify the ordering plausibility, and one to specify defaults.

Our first extension to TQuel involves the schema specification statements. In the create statement we add the modifier ***indeterminate*** before the keywords ***event*** and ***interval*** to indicate that the (valid time) timestamps may be indeterminate; the distribution is initially assumed to be uniform. In the modify statement we provide clauses that allow either the starting or ending time (or both) to be specified as either determinate or indeterminate, and in the latter case to specify a distribution that applies to all tuples in the relation. The following statements might be used for the historical database shown in Figure 1.

```
create indeterminate event Fossils (Creature: string)
create indeterminate interval Classifications (Period: string; Epoch: string)
modify Fossils to Uniform distribution
```

With the modify statement, it is also possible to specify the duration of the indeterminate event intensionally. In that case, the terminating chronon need not be stored; it can be computed from the starting chronon and this duration.

The semantics of these extensions are straightforward. The intensional information concerning the determinacy of the timestamps is recorded in the system catalogue. The intensional

	Creature	Valid time (at)	Distribution
s_5	*Unidentified*	185–195	Uniform
s_6	*Carnivorous*	185–195	Uniform
s_7	{Spinosaurus, Kentrosaurus}	185–195	Uniform

Figure 7: Examples of value indeterminacy

that is most appropriate for their application.

4 Retrieving Possible Tuples in TQuel

This section describes the syntax for retrieving information from databases with historical indeterminacy; the next section provides a formal semantics for these constructs. The proposed syntax and semantics are an extension of the syntax and semantics of the retrieve statement in the TQuel temporal query language [Snodgrass 1987]. A primary design goal in extending TQuel to support historical indeterminacy was to make a minimal extension. It will be shown in Section 5.6 that the new syntax and semantics preserves the meaning of all extant TQuel retrieve statements.

4.1 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 1987]. An example retrieve statement that determines the geologic epoch of Trilobite fossil is shown in Figure 9. The retrieve statement 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.

In the example, the default rollback was used, referring to the current state of the database. Since rollback concerns transaction time while historical indeterminacy only concerns valid time, the *as of* clause will not be discussed further; its syntax and semantics are unchanged in TQuel with historical indeterminacy.

The where clause in this example specifies that the only kind of `Fossils` to be considered are Trilobites. The where clause is also unaffected by historical indeterminacy and will not be discussed further. The when clause matches Trilobites with geologic classifications by testing for overlap (overlap can be thought of as temporal intersection). The predicate in the when clause

3.4 Indeterminate Tuples

The data model we propose is an extension of the TQuel data model [Snodgrass 1987]. The TQuel data model supports *temporal* relations. A temporal relation may be thought of as a sequence of historical states, each of which is a complete historical relation. The *rollback* operation on a temporal relation selects a particular historical state, on which an historical query may be performed. For the purposes of this paper, we will assume that a temporal relation is embedded in a snapshot relation by adding two implicit attributes. The value of the first attribute specifies a valid time while the value of the second attribute specifies a transaction time. Thus, tuples in relations in the database are timestamped with both transaction and valid times.

The granularity of a transaction time timestamp is the smallest inter-transaction time, the chronon during which a transaction takes place is always known. Hence transaction times are always determinate; historical indeterminacy occurs only in valid time. We will largely ignore transaction time in this paper. The example database (Figure 1) shows two historical relations with implicit valid time attributes only.

A valid time timestamp can either be an event or an interval. An *indeterminate tuple* is timestamped with either an indeterminate event or an indeterminate interval. Since the model assumes tuple rather than attribute-value timestamping, each indeterminate tuple represents a set of *possible tuples*. A possible tuple is a tuple in which the indeterminate event (interval) is replaced with a possible event (interval). Like a possible event and a possible interval, a possible tuple has no historical indeterminacy.

One consequence of tuple timestamping is that tuples in relations are "row-independent", that is, no information is shared between indeterminate tuples. An anticipated use of historical indeterminacy is to represent approximate changes in the historical state of a relation. For example, in the `Classifications` relation shown in Figure 1, the event representing the transition from the Early Jurassic to the Middle Jurassic is present in two tuples, one as the terminating event and one as the starting event. The set of possible events 189–182 million years ago is conceptually shared between the two tuples. However, the possible tuples model makes no overt provisions for sharing indeterminate information between tuples because such provisions significantly increase the computational complexity of query processing. Instead, each tuple is independent of the other tuples in the relation, and no inter-tuple temporal requirements, such as the probabilities of the sets of possible events associated with a key must sum to one, are imposed or ensured.

3.5 Other Kinds of Indeterminacy

In the possible tuples model, historical indeterminacy is orthogonal to other sources of incompleteness [Motro 1990]. In particular, it can peacefully coexist with *value indeterminacy*, where the value of an attribute (as opposed to a timestamp) is not fully known [Dubois & Prade 1988]. For example, in the `Fossils` relation, we may have a fossilized tooth that has not yet been identified (s_5 in Figure 7), has been partially identified (s_6 restricts the fossilized tooth to belong to the specified class of animals), or has been narrowed down to a set of possibilities (s_7) [Barton 1989]. We advocate separating the various kinds of indeterminacy, so that users can choose the combination

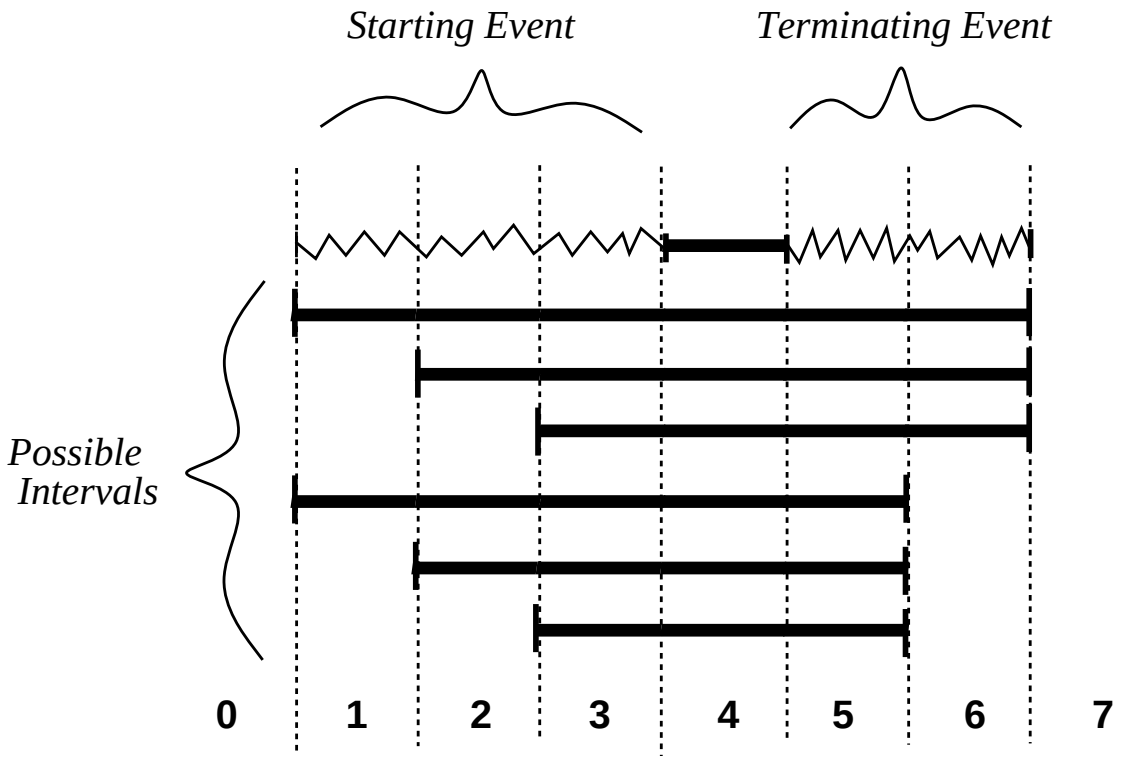


Figure 6: The possible intervals implicit in an indeterminate interval

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 replacing each bounding indeterminate event with a possible event (see Figure 6). Every combination of possible events in the starting and terminating events for an indeterminate interval is in the set of possible intervals.

For every indeterminate interval, every member of the set of possible events for the starting event must be before every member in the set of possible events 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 events overlapped. There is one exception to this maxim; the sets of possible events 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.

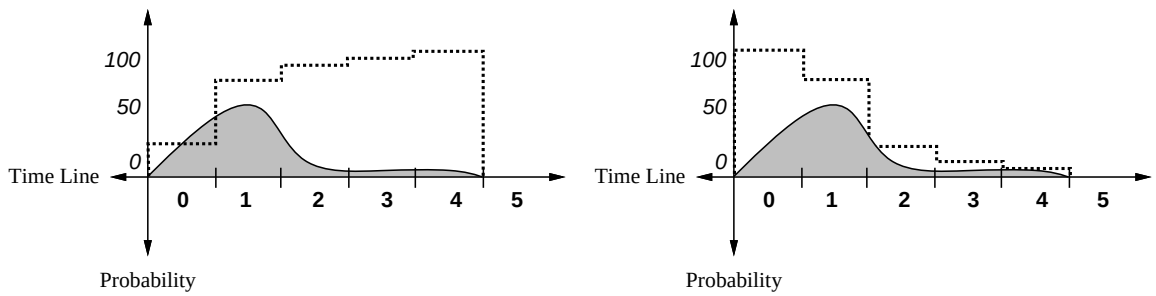


Figure 5: The *starting* and *terminating* probabilities

given in Figure 3. The starting probability is defined to be:

$$SP_\alpha(\alpha_k) = \lfloor 100 \times \sum_{i=-\infty}^{k} P_\alpha(\alpha_i) \rfloor$$

Note that the starting probability is expressed as an integral percentage. For each chronon in a set of possible events the cumulative probability that the event occurs on or after it is called the *terminating probability* for that chronon. Figure 5 shows the terminating probabilities for the distribution given in Figure 3. The terminating probability is also expressed as an integral percentage:

$$TP_\alpha(\alpha_k) = \lfloor 100 \times \sum_{i=k}^{\infty} P_\alpha(\alpha_i) \rfloor$$

In the terminology of probability theory, the function that computes the starting probability is the *cumulative density function* for the event random variable, scaled by 100, while the terminating probability is the (scaled) cumulative density function computed for the "reversed" set of possible events and density function. Note that for all indeterminate events $\alpha = [\alpha_1,\ \alpha_n]$, $SP_\alpha(\alpha_n) = 100$ and $TP_\alpha(\alpha_1) = 100$.

While the terminology used in the possible tuples 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 events that contains a single chronon. In this case, the starting and terminating probabilities for that chronon are 100.

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 events of the starting event. Likewise, the indeterminate interval could end during any chronon in the set of possible events 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.

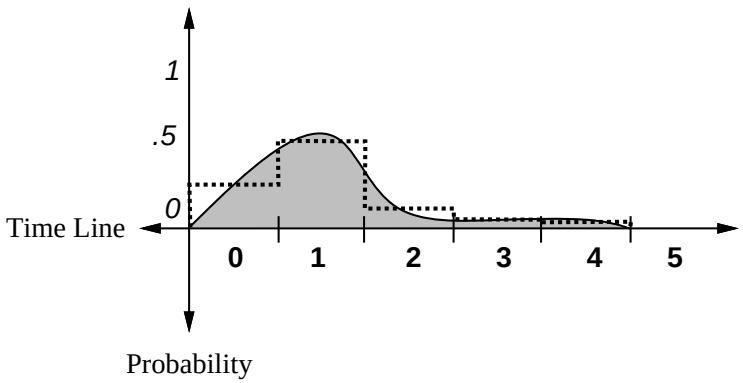


Figure 3: A probability distribution for an indeterminate event

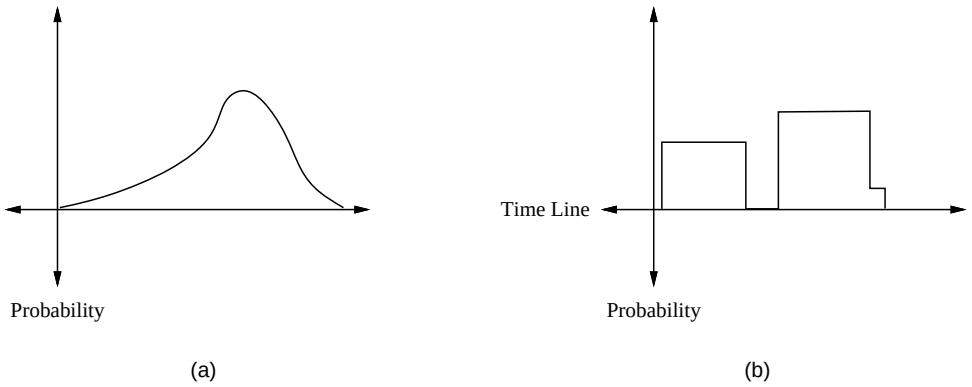


Figure 4: Example application-specific probability distribution functions

The event probability distribution could be a continuous or a discrete function. Recall, however, that chronons are the smallest unit of time. Therefore a discrete distribution that has a probability for each chronon is sufficient. Figure 3 depicts a "probably early" distribution, defined over chronons α_1 through α_n; $P_\alpha(\alpha_i)$ is the value of this function for chronon α_i. The dashed line in the figure shows the discrete probabilities. The event probability distribution for an indeterminate event is supplied by the user; the default distribution is uniform probability. As shown in Section 4.2, we allow users to provide probability distributions relevant to their application. For example, Figure 3 might illustrate the probability that a performer is "gonged" on *The Gong Show* (the performance is likely to end early). The time of occurrence of a reported heart attack might have the "probably late" distribution shown in Figure 4.a (most heart attacks are reported soon after they occur). A promotion event in a personnel relation might have the distribution shown in Figure 4.b (promotions occur during the business day, except at lunch; they occur at a higher probability in the afternoon, and at a much-reduced probability after the mail is picked up at 4:15pm).

For each chronon in a set of possible events the cumulative probability that the event occurs on or before it is called the *starting probability* ($SP_\alpha(\alpha_i)$) for that chronon. Figure 5 shows the starting probabilities for each chronon in the indeterminate event with the event probability distribution

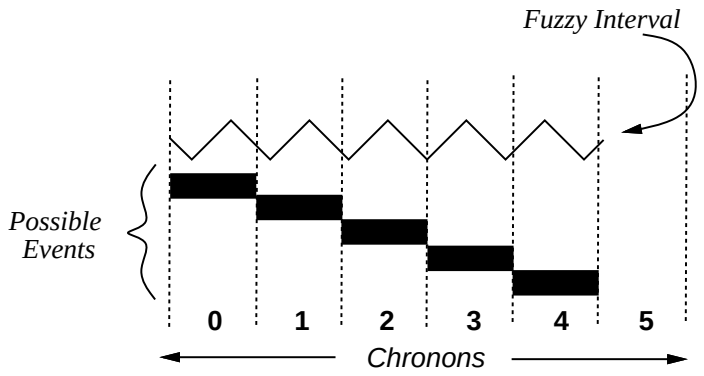


Figure 2: The possible events represented by an indeterminate event

3.2 Indeterminate Events

The duration of events in the real world is between "instantaneous" and "indefinite." But in the possible tuples model, events that have durations of less than a chronon cannot be accurately represented, since a chronon is the smallest representable unit of time. Nor can real world events that last more than a chronon (although use of an interval may suffice in such cases). The model treats real world events as though they have a duration of a single chronon.

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*. Two pieces of information completely describe an indeterminate event: a *set of possible events* and an *event probability distribution*.

A set of possible events is a set of contiguous chronons. The set is well-ordered. A single chronon from this set denotes when the indeterminate event actually occurred. Each chronon in a set of possible events represents a *possible event*. A possible event is a determinate event that might have occurred. Figure 2 illustrates a set of possible events. Here, there are five possible events shown: the one occurring during chronon 0, the one occurring during chronon 1, the one during chronon 2, the one during chronon 3, and the final one occurring during chronon 4. One of the possible events is determinate, but which is unknown. In this paper, we will represent a set of possible events that extends from chronon c_1 to chronon c_n using the notation $[c_1,\ c_n]$. The indeterminate event shown in Figure 2 is associated with the set of possible events $[0,\ 4]$.

Although an indeterminate event represents a set of possible events, not all the possibilities are equally likely. For instance, it could be that the event most likely happened earlier than the median of the set of possible events. In this case, the possible events that lie in the early half of the set of possible events are more probable. An *event probability distribution* gives the probability that the event occurred during each member in the set of possible events. In the terminology of probability theory, this distribution is the *density function* for the event random variable. We represent an event probability distribution for an event α with the notation P_α. Also, we denote the indeterminate event α by the ordered pair $([\alpha_1,\ \alpha_n],\ P_\alpha)$, where $[\alpha_1,\ \alpha_n]$ is the set of possible events for α and P_α is the event probability distribution for α; $P_\alpha(\alpha_i)$ is zero if $\alpha_i < \alpha_1$ or $\alpha_n < \alpha_i$.

likely that it started between 238–236 million years ago (the normal distribution favors ages near the mean possible age). A typical user might only be interested in those Late Triassic intervals that started between 238–236 million years ago, ignoring those that began earlier. In the possible tuples data model, the user can express this preference by selecting an appropriate range credibility value. The chosen range credibility changes each valid time interval in an historical relation, restricting the range of the 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 paleontologist might query during which epochs it is plausible that the Trilobite fossil was formed. Intuitively such a query relaxes the constraints on the relationship between the epoch intervals and the age of the fossil sample from "absolutely sure of overlap?" to "is it plausible that they overlap?". It is plausible that the Trilobite could have been formed during the Late Triassic or the Early Jurassic, but it is impossible to be absolutely sure of either. It turns out that the Early Jurassic is more plausible.

In part, ordering plausibility allows the user to express a level of plausibility in a temporal relationship (e.g., how plausible is it that the formation of the Trilobite preceded the start of the Early Jurassic?). But since queries can contain temporal constructors, ordering plausibility also controls construction of the answer to the query.

We believe that there is a natural division between indeterminacy in the data and indeterminacy in the query. The possible tuples model allows the user to control both kinds of indeterminacy. Range credibility and ordering plausibility are not orthogonal controls. 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 historically indeterminate events and intervals are represented. We make some concessions to practicality by employing discrete representations of time and probability. In Section 6, we make a few additional approximations to achieve efficiency.

3.1 Time

The possible tuples model quantizes a continuous time-line into discrete *chronons*. The chronons are contiguous, well-ordered, and in a one-to-one correspondence with the integers. A chronon is the smallest representable unit of time. We do not assume a specific granularity or chronon size; a chronon may be of any duration (e.g., nanoseconds, years, Chinese imperial dynasties). This feature allows the implementation to choose the necessary granularity. The granularity should not be dictated by the data model. We also assume that the implementation supports only a single chronon size; multiple granularities will be handled by representing the indeterminacy explicitly.

Fossils(Creature)

		Valid time	
	Creature	(at)	Distribution
s_1	Pleliasauras	211–204	Uniform
s_2	Trilobite	209–202	Uniform
s_3	Clamshell	224–200	Uniform
s_4	Stegasauras	180–170	Uniform

Classifications(Period, Epoch)

		Valid time			
Period	Epoch	(from)	Distribution	(to)	Distribution
Triassic	Middle	250–250	Normal	245–218	Normal
Triassic	Late	245–218	Normal	215–204	Uniform
Jurassic	Early	215–204	Uniform	189–182	Uniform
Jurassic	Middle	189–182	Uniform	170–156	Normal

Figure 1: An historical database

age in the interval of recorded ages is equally likely. The equal likelyhood assumption is actually encoded in the tuple, it is the "Uniform" value in the *Distribution* attribute of each tuple in `Fossils`. Since all of the distributions are the same, this information could be encoded in the schema, as will be discussed in Section 4.2. In general, though, each valid time event might have a different distribution. Because radioactive dating is one method of determining geologic epochs, it is also imprecisely known when each epoch began or ended. Some of the distributions associated with the valid time events are "Normal" (bell-shaped curve) distributions. We assume that the normal distributions in this example are parameterized by values that specify their mean and standard deviation; we do not include these parameters here. We also do not claim that the specific distributions used in this example are realistic; ours were chosen purely for expository purposes.

A typical user might be interested in retrieving the geologic epoch corresponding to the age of the Trilobite fossil (we give such a query in Section 5.5). In TQuel, she could query to determine which geologic epochs "overlap" the valid time of the Trilobite fossil. Overlap is the operation of temporal intersection. Note that some fossils could have been formed in more than one epoch. For example, the Trilobite could have been formed during the Late Triassic or Early Jurassic.

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, it is very unlikely that the Late Triassic started between 245–243 million years ago; it is much more

1 Overview

An historical database records the history of an enterprise. 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 *historical indeterminacy*. Information that is historically indeterminate can be characterized as "don't know when" information, or more precisely, "don't know *exactly* when" information. We believe that this type of knowledge is prevalent.

Temporal databases should provide support for historical indeterminacy. In particular, timestamps should include a representation for historical indeterminacy; users should be able to control, via query language constructs, the amount of indeterminacy present in derived information, and the query evaluator should accommodate historical indeterminacy in its processing. Query evaluation efficiency should remain high in the presence of historical indeterminacy, and it should not be affected at all in its absence. Efficient query evaluation is especially challenging, since timestamp manipulation is in the "inner loop" of query evaluation; it is performed several times for each participating tuple.

This document describes the *possible tuples* data model. The model adds valid time historical 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 historical 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 theoretically equivalent to SQL, our ideas can be applied to both languages.

The next section introduces the example that will be used throughout the paper. We then examine the representation of historical indeterminacy, and explore what it means to retrieve information from a database with historical indeterminacy. Emphasis is placed on providing a simple and intuitive retrieval method. Next we outline syntactic extensions to TQuel, followed by a formal tuple calculus semantics for the retrieve statement. The sixth section is a discussion of implementation issues. We show specifically how to represent historical indeterminacy in minimal space, and how to efficiently implement the ideas presented in the previous sections. The final sections trace related work, summarize our approach, and discuss future work.

2 Motivating Example

An historical geological database is shown in Figure 1. The database consists of the relation `Fossils`, a collection of fossil samples that have been dated using a radioactive dating technique, and the relation `Classifications`, a history of geologic epochs. The valid time for each tuple is on a scale of millions of years in the past. Since radioactive dating techniques are inherently imprecise, the age of each fossil is indeterminate. For instance, the Trilobite fossil was formed sometime between 209–202 million years ago. For this relation we will assume that each *possible*

8 Summary and Future Work **29**

Contents

1 Overview **1**

2 Motivating Example **1**

3 Extending the Data Model with Indeterminacy **3**

3.1 Time . . . 3

3.2 Indeterminate Events . . . 4

3.3 Indeterminate Intervals . . . 6

3.4 Indeterminate Tuples . . . 8

3.5 Other Kinds of Indeterminacy . . . 8

4 Retrieving Possible Tuples in TQuel **9**

4.1 Review of TQuel . . . 9

4.2 Extensions for Historical Indeterminacy . . . 10

5 Semantic Extensions **13**

5.1 Determinate TQuel semantics . . . 13

5.2 Supporting Range Credibility . . . 14

5.3 Supporting Ordering Plausibility . . . 16

5.4 Coalescing . . . 20

5.5 Semantics of the Example Query . . . 21

5.6 Query Reducibility . . . 23

6 Implementation **24**

6.1 Representing Indeterminacy . . . 25

6.2 Query Evaluation Algorithms . . . 26

7 Related Work **27**

Copyright © Curtis E. Dyreson and Richard T. Snodgrass 1992

Historical Indeterminacy

Historical Indeterminacy

Curtis E. Dyreson[1]
Richard T. Snodgrass[1]

TR 91-30a

Revised April 21, 1992

Abstract

In *historical 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 the *possible tuples* data model, in which each indeterminate event is represented with a set of possible events that delimits 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 historical data and in the user's plausibility in the relationships among that data. We provide a formal tuple calculus semantics, and show that this semantics reduces to the determinate semantics. We outline an efficient representation of historical indeterminacy, and efficient query processing algorithms, demonstrating the practicality of our proposed approach.

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