Ordered Time Scales

Andrew U. Frank
Dept. for Geo-Information E127
Technische Universität Wien
Gusshausstr. 27-29
A-1040 Vienna Austria
frank@geoinfo.tuwien.ac.at

ABSTRACT

There is a strong request for GIS to include temporal information. Most efforts currently are addressing the incorporation of time qua calendar time. Events are dated according to the ordinary time and calendar, which are effectively measurements on an interval scale. Temporal information available only as relative order between events cannot be incorporated in this framework. Clearly knowledge about temporal order without measurement on the time scale is less precise but nevertheless useful. Human beings use qualitative temporal reasoning all the time.

Qualitative ordinal information about events is typically encountered in archeology, urban development etc. where precise dates for events are not known but the relative order of events can be deduced from observations. Even in legal proceedings about parcel data, ordinal relations are often all what matters. These are among the disciplines which have asked for the inclusion of facilities to deal with temporal data in GIS.

This paper gives specifications for ordinal temporal reasoning using qualitative methods. It differentiates between different time models, each having slightly different properties: models with or without events at the same time, models with total or partial order. It discusses the introduction of tolerances (without recourse to measurements of an epsilon value) and how it affects reasoning.

The semantics are given as formal specification, expressed in an algebraic notation which can be executed. An example from a parcel subdivision is used throughout and results from various computations are compared with human logical deduction.

INTRODUCTION

There is a strong request for GIS to include temporal information and research in the last years has addressed this issue increasingly[Barrera, and Al-Taha, 1990; Langran, and Chrisman, 1988]. Most efforts currently are addressing the incorporation of time qua calendar time. Events are dated according to the common ordinary time and calendar, which are effectively measurements on an interval scale. This makes the full set of methods from real arithmetic available for the analysis of such data.

In some situations, temporal information is only given as relative order of events, without time-stamps which relate the events to a fixed time scale. Clearly such knowledge is less precise than measurements on the interval time scale, but nevertheless useful. Human beings use such qualitative temporal knowledge all the time. Several of the key disciplines which have requested the incorporation of temporal data in a GIS [Barrera, and Al-Taha, 1990] typically deal with event sequences without precise data, for example in archeology, urban development etc.

where precise dates for events are not known. Even in legal proceedings about parcel data, ordinal relations are often all what matters [Al-Taha, 1992].

Among the increasing number of papers on space and time in the GIS context, most consider time only as measurement on a date line and mostly discuss the difficult data base and data storage issues arising [Snodgrass, 1992] or issues of specific applications. Nevertheless, research in the conceptual models used for time are beginning to emerge [Golledge, and Egenhofer, in preparation]. Alternative models for time are necessary and the differences in the concepts used to model time or space matters for GIS. It influences the use of the GIS [Campari, 1991], the user interface [Kuhn, and Frank, 1991; Payne, 1991], the internal representation [Egenhofer, and Kuhn, 1991; Frank, and Kuhn, 1986; Gueting, and Schneider, 1992; Herring, Egenhofer, and Frank, 1990], and it also influences the visualization, including the understanding and visualization of data quality [Beard, and Buttenfield, 1991; Mark, and Frank, 1991].

Different situations require different models of time or space. Humans form concepts of a situation according to the task at hand and do not include more data than necessary (principle of economy). Qualitative reasoning, i.e. reasoning methods which involve only a small number of distinct symbols, are often economical and are providing useful models for human performance. The model of ordinal time is but one model of time and reasoning with time. It deals with relations between events (i.e. ’before’ and ’after’), where some relations are given and others must be deduced.

Discussing models of time (or space) requires consideration of a surprising number of details, and concentration on a specific situation and model or family of models is necessary. Here the focus is on a qualitative reasoning model for time as differentiated from quantitative reasoning. The information considered will be about the order of two events and the calculus will answer arbitrary question about the sequence of events. It is motivated by an example deduced from a subdivision of land in parcels. Even within such a limited model, several variants must be differentiated. Results of temporal reasoning are different for different models and algorithms are necessarily different. Here two models of ordinal time, one with total order, one with partial order are combined with models a model of events at the same time to yield a family of four ordinal time models.

Formalizations for the concepts described are given and permit to clearly point to the differences and dependencies. The semantics are formal, based on mathematical concepts [Birkhoff, and Lipson, 1970; Goguen, Thatcher, and Wagner, 1978]. The formal specification method permits to identify small building blocks and to combine them. The specifications were checked with a compiler and can be executed. The results are then compared with our intuition about the semantics to captured. The language used is Gofer [Jones, 1991], a dialect from the functional programming language Haskell [Hudak *et al.*, 1992]. The advantages of mechanical tests of formal correctness and the possibility of testing outweigh the small notational inconveniences introduced by the language. Of course, specifications are written for optimal clarity, not for fast execution. A number of optimizations are possible and the translation to a ’mainstream’ programming language is straight forward.

The next section reviews the context of this work and section 3 explains the approach used. In Section 4 the example used throughout the paper is presented. The following section first specifies systems where only order is considered. The discussion in Section 6 introduces then a model where equality is also considered and shows how the two models are combined. The conclusion points to topics for further work.

DIFFERENT MODELS OF TIME

An early discussion of types of measurements is found in a landmark paper about the "Theory of Scales of Measurements". Stevens points out that

’The isomorphism between these properties of the numeral series and certain empirical operations which we perform with objects permits the use of the series as a *model* to represent aspects of the empirical world.’ [Stevens, 1946, p. 677]

This points to differences in models as related to measurements (even if observing basically the same phenomena) and explains the differences in terms of comparable operations applicable to the empirical world and the model. The differences are not ascribed to the empirical reality, but rather to differences between empirical and cognitive operations applied to it.

The ability of human beings to switch effortless between different models or move to a more detailed model if a need arises confuses investigations enormously. GIS software incorporates a small number of models and the differences to the ones practically used become painfully obvious. Some of the models are closely related and differences are only small, but observable. It is thus necessary to capture the semantic of different models. To make the task surmountable, we proceed by first identifying small building blocks which are then combined to capture complex semantics.

APPROACH

The method used here is an application of type theory from computer science. The scales of measurement are seen as types and further refined. Specification for small building blocks are given and then shown how they combine to capture complex meaning [Guttag, Horning, and Wing, 1985].The major model of time studied here is based on events and order between events. In particular total and partial order are studied.

A landmark paper "On Understanding Types, Data Abstraction and Polymorphism" explains types simply by: "Sets of objects with uniform behavior may be named and are referred to as types" [Cardelli, and Wegner, 1985, p. 473]. By behavior in this context is meant primarily the behavior of operations on the computational model, in particular the outcome of operations, e.g. the test if event A is before event B.

The mathematical background is found in algebra, where one considers sets of objects and operations applicable to them. (This implies some terminological difficulties: the terms model, type and (multi-sorted) algebra all denote the same notion seen from a different context, namely the application domain, computer science and mathematics.) Integer for example is a set of objects, with the operations plus, minus, and multiplication. Real numbers are another set of objects (another type) with similar operations, but including a division. Discussion of types in programming is important, because " In mathematics as in programming, types impose constraints that help to enforce correctness" [Cardelli, and Wegner, 1985, p. 474]. Objects and operations, for example measurement and other observations applied to them, link the different viewpoints of ’scales of measurements’, conceptual models, computational models and formalizations.

Starting with a minimal set of operations or constraints that capture some temporal aspects, building blocks are sets (generic ordered time, time with equality). These are then combined using inheritance to build more complex types. To combine algebraic specifications of different objects remains currently difficult in many situations. Herring has already showed the importance of category theory[Asperti, and Longo, 1991] for the foundation of GIS types [Herring, 1990 #57]. The problem is especially difficult, when user interfaces and databases are considered. King and Wadler presented recently a promising solution[King, and Wadler, 1993], which can be applied to GIS to specify input/output operations [Peyton Jones, and Wadler, 1993].

Formalizations provide a means to show precisely what is meant with the operations. It defines semantics in a precise sense, based on the concept of denotational semantics [Scott, 1977, Stoy, 1977]. It becomes evident what is common and where different models of time differ. The specifications written in a formal language can be executed and are thus controlled for completeness, correct types etc., and then compared with the intuitive notions, to assure that they correctly capture the intended meaning.

THE EXAMPLE

An example where ordinal temporal information is available, but no dates measured on an interval scale, are patterns of spatial subdivisions. The subdivisions of a piece of land as sketched in Figure 1 provide the data for our tests. From the geometric situation in figure 1, strong evidence for a certain temporal sequence of subdivision events follows. For example, the boundary line labeled g was established after the line labeled ’e’. The interesting question of how spatial and temporal knowledge is linked and how temporal information is deduced from a geometric situation is here only used as an example and not further investigated (for a study from the geological domain, see [Flewelling, 1992 #52]).

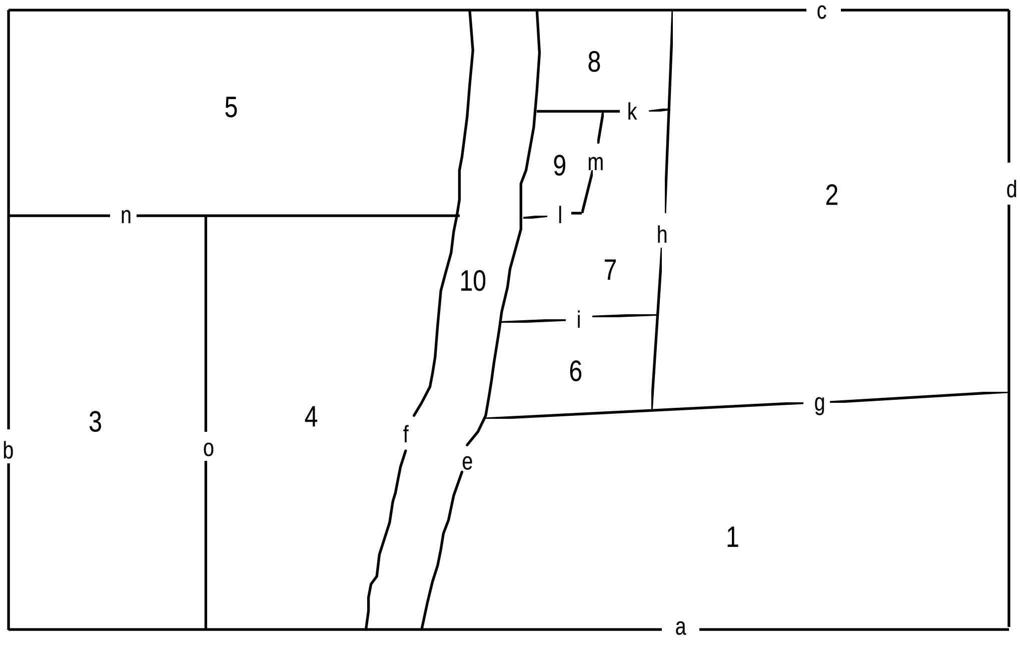

Figure 1: Land-subdivision

The events considered are the establishment of boundary lines. Events are labeled with lower case characters in lieu of some more meaningful data. The following facts are used ('a' :< 'b' states that event a is before event b, 'x' := 'y' asserts that the events x and y are contemporaneous). The temporal relations can also be shown as a Hasse diagram (figure 2) where events higher are before events lower and the edges between events describe the available facts. The facts are collected in four lists:

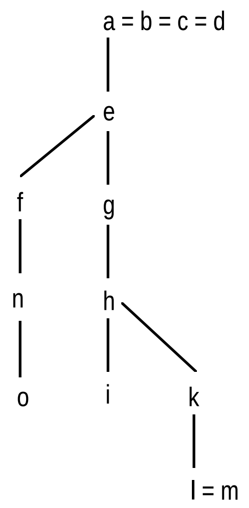

Figure 2: Hasse diagram of temporal relations

```
facts1 = [ 'a' :< 'e', 'e' :< 'g', 'g' :< 'h', 'h' :< 'i']
facts2 = [ 'e' :< 'f', 'f' :< 'n', 'n' :< 'o']
facts3 = [ 'h' :< 'k', 'k' :< 'l']
facts4 = [ 'a' := 'b', 'a' := 'c', 'a' := 'd',  'l' := 'm']
```

Assumptions

The objects considered are events, seen as points in time, and collections of such events with relations between them. On these collections, deductive reasoning operations are applied. Points in time are abstractions like Euclidean points, with no extension. Facts are given as A before B (implying A strictly before B and not at the same time).

To simplify treatment and stay within the limits of traditional logic, the set of facts will be considered constant and given at the beginning. The extension to a system of monotonic reasoning, where new facts can be added to a collection, but the new facts must not invalidate any of the previously true statements[Davis, 1990] is easy and requires only the addition of tests to assure that the information provided is not contradictory. These tests for consistency are left out to keep the example simpler.

The "closed world assumption" is applied, which states that everything about the universe of discourse is known and the absence of information allows to infer that the negative is true. This is a standard assumption for database query languages, even if it is not always appropriate in a GIS context [Reiter, 1984].

ORDINAL MODELS FOR EVENTS

1 ORDER RELATIONS

Order relations should fulfill the axioms of asymmetry and transitivity, which are the two properties at the base of any concept of time. Mathematicians usually use a *smaller or equal* relation '<=', which is reflexive, antisymmetric and transitive, to construct order relations. For "*before or equal*" we will write A <= B with the properties:

A <= B = true	reflexive
A <= B and B <= A implies A = B	antisymmetric
A <= B and B <= C implies A <= B	transitivity

From the *before* relation follows the converse *after* relation (A *before* B is the same as B *after* A) and also the strict order relations (before but not at the same time) defined in terms of >=:

A < B = (A <= B) and not (A >= B)

A > B = (A >= B) and not (A <= B)

A relation *'immediately before'* (<<) and the converse *'immediately after'* (>>) is defined in terms of the known facts: A is said to be immediately after B if there is no event C between A and B (within the facts known).

- *A << B = (A < B) and (not exist C such that A < B < C).*

2 THE GENERIC FORMAL MODEL FOR ORDINAL TIME

The class Time uses a class Set,with the customary operations. The keyword 'class' starts the definition of a class. The class name (with type parameters) may be preceeded by a list of classes inherited before the symbol =>. Classes describe generic types with type parameters. Local variables are defined in where clauses. Theboolean operators are '&&' for and, '||' for or and '==' is the test for equality. [] is an empty list, [a] is a list with single item a, both are used here as a set.For all the operations the signatures are given in the form:

- operation name :: (list of argument types) -> return type

As usual in algebra, all operations are functions, returning a single value. (To connect easier with regular mathematics, the functions are written in the 'uncurried' form. Technically, they have a single argument, which is a n-tuple.) Following the operation signatures are definitions given for all operations which can be defined in terms of the base relation *immsucc* and *immpred*. Operation definitions are regular equations.

The *class Time* is specified with the operations

new:	to create a new time collection,
bef:	to insert a time fact in the form of A < B
succ, *prec*:	to determine all events succeeding/preceding the given event (including the given event),
immsucc, *immpred*:	to determine the immediate successor/ predecessor (not including the given event)

and tests for the relations between two events: before, after, immediately before, immediately after.

For technical reasons the two additional operations *immsucc'* and *immpred'* are necessary to transform the arguments passed to the closure operation.

```
class SetClass t =>  Time l t where
   new                         :: (t, t) -> l t
   bef                         :: (t, t, l t) -> l t

   succ , pred                 :: (t, l t) -> Set t
   immsucc , immpred           :: (t, l t) -> Set t
   before,  after              :: (t, t, l t) -> Bool
   immedbefore, immedafter     :: (t, t, l t) -> Bool
```

The converse relations after and immediately after are defined in terms of corresponding relations for before. '*A before B*' is defined as 'B is element in the set of all event succeeding A'. The test for '*A is immediately before B*' is additionally testing that there is no point after A and before B. The successors or predecessors of an event are computed as the transitive closure of the immediate successors (resp. predecessor). As the given event is included in the set of the successors (A is a successor of itself), the relations for before and after are symmetric ('A before A' and 'A after A' are both true).

Please observe a subtle difference in the definition of the notion of immediate successor and the relation of 'immediately after'. Immediate successor is based on the facts collected: an event A is immediate successor of B, if there is a fact*A < B*' included in the collection (and correspondingly for the predecessor). The collection may include several time facts, stating that*A > C* and *B > C*. In this case both, A and B, are immediate predecessors of C (even if a fact A > B is also available). To deduce that A is immediately before C requires a test for A being a predecessor of C and the intersection of all predecessors of C (here A and B) and the successors of A (here C) is empty (Figure 3), which is the translation of*immedbefore a, b = a < x and not exist x, a < x and x < c.*

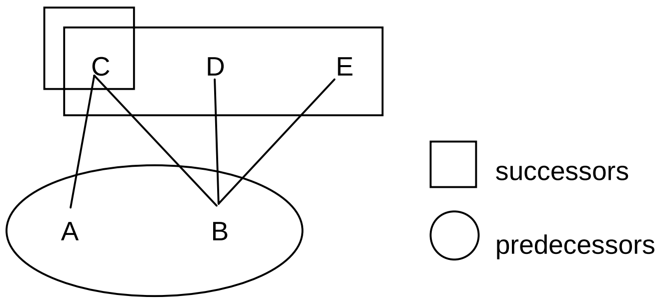

Figure 3: Visualization of 'A is immediately before C'

```
immedafter  (a, b, t) =  immedbefore  (b, a, t)
after  (a, b, t) =   before  (b, a, t)
before  (a, b, t) = memberInSet (b, succ  (a, t))
```

```
immedbefore  (a, b, t) =
   memberInSet (a, imp) &&  emptySet (intersectSet (ims,    imp))
   where                  ims = immsucc  (a,t)
                          imp = immpred  (b,t)

succ  (a, tl) = closure ([a], immsucc' , tl)
pred  (a, tl) = closure ([a], immpred' , tl)

immsucc  (a, tl) =  immsucc' ([a],[], tl)
immpred  (a, tl) =  immpred' ([a],[], tl)

where a and b are variables of the type event and tl is a  variable of
type timeline.
```

A time line is generated from an empty line, with a mark *New*. Facts are inserted, to form strings as *Bef ('a', 'c', Bef ('c', 'd', Bef ('a', 'z', New)))* etc. (Constructive types are introduced with 'data' statements, followed by alternative constructors, which can be used to generate the type. Definitions of operations are written as equations '*op (x,y) = x + y*'. Instances for generic types are necessary and for those operations, which are expressed in terms of the constructors, they are defined only then.) For this data, the operations declared but not yet defined (new, bef, immsucc and immpred) are now given. *new* and *bef* generate the time line. A new timeline is created with two bounding events, arbitrarily labeled 'a' and 'z'. The observers *immsucc'* and *immpred'* are collecting the immediate successors (or predecessors) by recursively following the time line.

```
data   TimeFacts  t = Bef  (t, t, TimeFacts  t) | New

instance (SetClass t) => Time  TimeFacts  t where
    new  (a,b) = Bef  (a,b, New )
    bef  (a,b, tl) = Bef  (a, b, tl)

    immsucc' (a, s, New ) = s
    immsucc' (a, s, Bef  (e, f, tt)) = if memberInSet (e,a)
                               then immsucc' (a, insertInSet  (f,s), tt)
                               else immsucc' (a, s,tt)
    immpred' (a, s, New ) = s
    immpred' (a, s, Bef  (e, f, tt)) = if memberInSet (f,a)
                               then immpred' (a, insertInSet e,s), tt)
                               else immpred' (a, s,tt)
```

This provides all the necessary apparatus to deal with ordinal times and captures the intuition about time in general. The following subsections demonstrate the changes necessary to generate different models of order time.

3 TOTAL ORDER

A naive concept of time assumes that there is only a single time line, and all events are located in this time. This image of time is based on a single person's experience, where A happens to me before B. For any two events C and D it can be determined if C is before D or D is before C. Technically, such relations are known as 'total order', because *for any A, B A< B or A = B or A > B.*

Such a time model is not only representing a persons experience, but also the events that apply to a single object, e.g. a building. It is first constructed, then changed, inhabited by family Smith and later by family Scott, and finally destroyed. This is a linear ordered time, and for any point in space or any object, such a totally ordered timeline can be constructed.

Information can be added to this totally ordered timeline only if it does not violate this restriction. In the example from figure 1, only the history of a single parcel (or a sequence of histories for a parcel and its ancestors) can be captured on such a strict timeline (e.g. for parcel 4: *a < f, f < n, n < o*). This is assured, if the only operation to add facts is of the form 'B is between A and C' is available. This implies that 'A is immediately before C' was true (otherwise the error message 'first not immediately before third event' is produced) and inserts the two facts *'A :< B'* and *'B :< C'*.

Formally, this *TimeTO* is derived from the previously defined generic time. Only the new operations for *'between'* must be defined, everything else is inherited from above.

```
class Time l t => TimeTO l t where
  between, between' :: (t, t, t, l t) -> l t
  between  (a, b, c, lt) =  if  immedbefore (a, c, lt)
                            then between' (a, b, c, lt)
                            else error 'first not before third event'

instance  TimeTO TimeFacts t where
      between' (a, b, c, (lt)) =   (bef  (a,b, bef  (b,c, lt)))
```

A database is loaded with the facts for parcel 1, 2, and 6 and tested with queries.

4 PARTIALLY ORDERED TIME

The general case is that time facts are not restricted to a single sequence of events. Information about events happening in different locations or to different persons, is only giving the order among subsets, but is not sufficient to determine total order. Deduction of information about relative order within one or the other sequence of events is possible, but not the determination of temporal relations between events that are happening independently in different locations.

Such a relation forms a partial order, because the relation A before B is only a partial function mapping to Boolean values. It can be true or false or not determined; the later situation is sometimes expressed as ’A and B are not comparable’.

This time is deduced from the generic*Time*. The only operation added is used to test if two facts have a known temporal relation or not (incomparable), which is directly defined as not (a<=b or a>=b).

```
class  Time l t => TimePO l t where
        incomparable :: (t, t, l t) -> Bool
        incomparable (a, b, tl) = not (before (a,b, tl) || after (a,b,
tl))

instance TimePO TimeFacts Char
```

Here all the ’before’ relations from figure 1 can be introduced in the database and be tested. Interesting is only to observe, that indeed for some points A and B neither *A before B* nor *A after B* is true, for example ’f’ and ’g’.

ORDINAL MODEL WITH EQUALITY FOR EVENTS

1 EQUALITY ALONE

Temporal information can be available as two events A and B happening at the same time (here written as =). Such a model by itself is not very useful, but it forms a minimal building blocks from which the complex model is constructed. Equality must be reflexive, symmetric and transitive:

- *A = A*
- *A = B implies B = A*
- *A = B and B = C implies A = C*

and there is a connection between the equality model and the ordered model: *A = B* is *A >= B* and *B >= A*.

A timeline to store such facts inherits much from the base time and the implementation follows the same pattern as before. An operation to add the fact that A is at the same time than B and a corresponding test if A is equal to B are added. There is also a need for an operation to find all events equal (i.e. at the same time) then a given one, which is computed as the closure of the immediately equal events.

```
class Time l t => TimeE l t where
       eq :: (t, t, l t) -> l t
       equal  :: (t, t, l t) -> Bool
       immequal :: ([t], [t], l t) -> [t]
       allEqual :: ([t], l t) -> [t]



instance TimeE TimeFacts t where
   eq  (a,b, tl) = Eq  (a, b, tl)
   equal (a, b, tl) = memberInSet (b, allEqual ([a], tl))
   allEqual (a, tl) = closure (a, immequal , tl)

   immequal  (s, rs, New ) = rs
   immequal  (s, rs, Eq  (e, f, tt)) =
       if memberInSet (e,s) then immequal  (s,
                                      insertInSet(f,rs), tt)
          else if memberInSet (f,s) then immequal (s,
                                                insertInSet (e,rs),tt)
                          else immequal  (s, rs, tt)
   immequal (s, rs, Bef (x, y, tt)) = immequal (s, rs, tt)
```

2 EQUALITY AND ORDER

An expressive temporal system combines observation of '*equal*', *before*' and '*after*'. This is achieved by merging (inheriting) the behavior from Time with Equality and generic Time. Interesting is the interaction between the equality and the predecessor and successor relations. If *A is before B* and *B is at the same time than C* then *A is before C,* thus in every step of searching for immediate successors, all events which are known to be at the same time must be added to the set of successors (another limiation in the specification language makes it necessary to use a 'tag' TEO here to differentiate objects of this time model from the ones of the original generic model).

```
data TimeFactsEO t = TEO (TimeFacts t)

instance (TimeE TimeFacts t, Time TimeFacts t) =>   Time
                                                        TimeFactsEO
t where
   new (a,b) = TEO (new (a,b))
   bef (a,b, TEO (tl)) = TEO (bef (a, b, tl))
   immsucc' (a, s, TEO (tl)) = allEqual (a2, tl)
                                      where a2 = immsucc' (a1, s, tl)
                                                a1 = allEqual (a, tl)
```

```
    immpred′ (a, s, TEO (tl)) = allEqual (a2, tl)
                                            where a2 = immpred′ (a1, s, tl)
                                                     a1 = allEqual (a, tl)
  instance  TimeE TimeFactsEO t where
    eq (a,b, TEO (tl) ) = TEO (eq (a,b, tl))
    equal (a,b, TEO(tl)) = equal (a, b, tl)
    immequal (a, b, TEO (tl)) = immequal (a, b, tl)
    allEqual (a,  TEO (tl)) = allEqual (a,  tl)
```

Two different time models are now deduced, one for total order and equality (*TimeTOE*) and one for partial order and equality (*TimePOE*). For this time model, relations *'strictafter'* and *'strictbefore'* are meaningful and are added.

```
  instance (TimeTO TimeFacts t ) => TimeTO TimeFactsEO t where
    between′ (a, b, c,  TEO (tl)) = TEO ( between′ (a,
                                                     b, c, (tl)))

  class (TimeE l t, TimeTO l t) => TimeTO′ l t where
    strictafter , strictbefore  :: (t, t, l t) -> Bool
    strictafter  (a, b, tl) = after  (a,b, tl) && not
                                                     (equal (a, b, tl))
    strictbefore  (a, b, tl) =  before  (a, b, tl) && not
                                            (equal(a, b, tl))

  instance (TimeTO′ TimeFactsEO t) => TimeTO′ TimeFactsEO t
```

The code for the time model with partial order and equality is even shorter:

```
  instance (TimePO TimeFacts t ) => TimePO TimeFactsEO t
```

This shows how new, more expressive, models of time can be combined from simpler models. It should also demonstrate the power of inheritance, which brings together behavior from different sources with minimal coding (and some of the code above is due to a limitation in the used language rather than inherent in the method).

3 EQUALITY WITH TOLERANCE

When are two events at the same time? This depends on the resolution of our observation system. Real measurement systems have some inherent error and results are not completely precise. Thus a strict definition is not practical and some notion of tolerance is needed, for example allowing a small difference epsilon between two measurements that are considered equal. As the time model investigated here does not provide for the computation of difference between events, this is not feasible.

A pragmatic observation system that only determines ’before’, ’equal’ or ’after’ for two events may find that A is before B and B is before C and also find that A is equal to D and B is also equal to D (figure 4 indicates this situation on a time line).

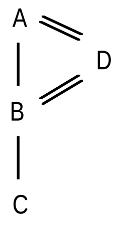

Figure 4: Time line of a<b, b<c,a=d, b=d

This shows that D is closer to A and also closer to B than what can be differentiated. If data of this nature must be processed, then the regular equality leads to logical contradiction and semiorders must be used. Measurement theory is providing the theoretical base for a solution. It introduces the concept of tolerance as "the just noticeable difference" [Krantz *et al.*, 1971; Scott, and Suppes, 1958]. Luce has defined semiorders as

"Let S be a set and < and = be two binary relations defined over S. (<, =) is a semiordering of S if for every a, b, c, and d in S the following axioms hold:

- S1. exactly one of a<b, b<a, or a=b obtains,
- S2. a=a.
- S3. a<b, b=c, c<d imply a<d,
- S4. a<b, b<c, b=d imply not both a=d and c=d." [Luce, 1956] p. 181]

These observations can applied to graphs [Roberts, 1969; Roberts, and Suppes, 1967] and lead to tolerance geometry [Robert, 1973] (Figure 5). Whenever facts are added in such a tolerance based system, one must check that none of the axioms is violated. The axioms S3 and S4 can be used for deduction. Deductions proceed regularly (using the same rules than for the time with equality and total or partial order).

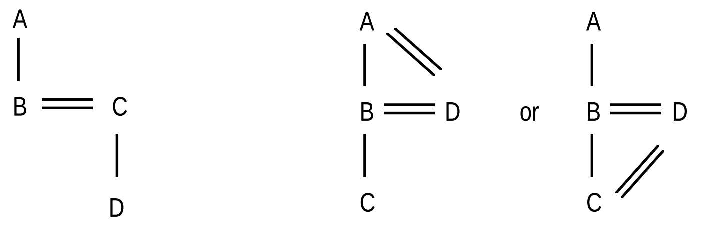

Figure 5: Time lines for axioms S3 and S4

CONCLUSIONS

Information given as order between events but without precise dates measured on a time line should be incorporated in a GIS. Many of the sciences which have asked for temporal data in a GIS have exactly this kind of imprecise, but nevertheless interesting data. Most current efforts to incorporate temporal data in GIS consider calendar dates, but here was shown that with a more restricted kind of data meaningful reasoning is possible.

Constructing a model for ordered time as a special case of the ’ordinal scale’ reveals that there are several variants, which differ in their definition in details but the effects of these small differences become visible when conclusions from a data set are drawn. Depending on the circumstances, one or the other model is appropriate.

The models were specified in an algebraic notation using a functional programming language. This produces an unambiguous definition of semantics, based on the concept of ’denotational semantics’ but also allows to execute the specification to test if the semantics captured correspond to the intentions.

From the ordered time models intervals over time can be constructed (again without a need for more precise time measurements) and the standard relations between intervals tested. Intervals based on total order are the standard case, but also intervals over partial order are possible. One can also construct cyclic ordered time, which by itself is trivial, because for any A and B, A is before B and A is after B. Based on ordered cyclic time, meaningful intervals are defined and allow the same type (but not exactly the same) of relations between intervals as in the linear ordered time model.

The many ways how these building blocks can be combined demonstrate the power of algebraic specifications with inheritance as a combination method and allow to understand the differences in the interaction of nearly identical models.

ACKNOWLEDGMENTS

Sabine Timpf made helpful suggestions to improve the presentation. I owe much to Mark Jones and Kevin Hammond who have invented the Gofer system and made it a delightful environment[Hammond, 1993] to experiment with formal specifications. Support from Intergraph Coorporation and for the NCGIA under a grant from NSF is gratefully acknowledged.

REFERENCES

Allen, J.F. “Maintaining Knowledge about Temporal Intervals.” Communications of the ACM 26 (1983): 832-843.

Allen, J.F. “Towards a general Theory of Action and Time.” Artificial Intelligence, 23 (1984): 123-154.

Al-Taha, Khaled. “Temporal Reasoning in Cadastral Systems.” Ph.D., University of Maine, 1992.

Asperti, Andrea, and Longo, Giuseppe. *Categories, Types and Structures*. Foundation of Computing Science, Cambridge, MA: MIT Press, 1991.

Barrera, R., and Al-Taha, K.K. *Models in Temporal Knowledge Representation and Temporal DBMS.* University of Maine, 1990.

Beard, Kate, and Buttenfield, Barbara. *Visualization of the quality of spatial data - Initiative 7 Position Papers.* NCGIA, 1991.

Birkhoff, G., and Lipson, J. D. “Heterogeneous Algebras.” *Journal of Combinatorial Theory* 8 (1970): 115-133.

Campari, Irene. “Some Notes on Geographical Information Systems. The relationship between their practical application and their theoretical evolution.” In *Cognitive and Linguistic Aspects of Geographic Space*, ed. Mark, D.M., and Frank, A.U. 419-434. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991.

Cardelli, Luca, and Wegner, Peter. “On Understanding Types, Data Abstraction, and Polymorphism.” *ACM Computing Surveys* 17 (4 1985): 471 - 522.

Davis, Ernest. *Representation of Commonsense Knowledge.* San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.

Egenhofer, Max J., and Kuhn, Werner. “Visualizing Spatial Query Results: The Limitations of SQL.” In *IFIP Workshop on Visual Databases in Budapest, September 30 - October 3, 1991*, edited by Knuth, Elöd, 1991.

Flewelling, D., Egenhofer, Max J. and Frank, Andrew U. “Constructing Geological Cross Sections with a Chronology of Geologic Events.” Fifth International Symposium on Spatial Data Handling / Charleston, SC., U.S.A., Vol. 2: 544-553, 1992.

Frank, A.U., and Kuhn, W. *Cell Graphs: A Provable Correct Method for the Storage of Geometry*. Second International Symposium on Spatial Data Handling//Seattle, Wa., July 5-10, 1986//pp. 411 - 436: orig, 1986.

Goguen, J.A., Thatcher, J.W., and Wagner, E.G. “An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types.” In *Current Trends in Programming Methodology*, ed. Yeh, R. Englewood Cliffs, NJ: Prentice-Hall, 1978.

Golledge, R., and Egenhofer, M., ed. *Time in Geographic space*. in preparation.

Gueting, R.H., and Schneider, M. *Realms: A Foundation for Spatial Data Types in Database Systems*. FernUniversitaet Hagen, 1992. Informatik-Report 134.

Guttag, J.V., Horning, J.J., and Wing, J.M. *Larch in Five Easy Pieces.* Digital Equipment Corporation, Systems Research Center, 1985.

Hammond, Kevin. *MacGofer, Version 0.21*. Dept. of Computing Science, Glasgow University, 1993. (available by anonymous FTP)

Herring, J., Egenhofer, M.J., and Frank, A.U. “Using Category Theory to Model GIS Applications.” In *4th International Symposium on Spatial Data Handling in Zurich, Switzerland* , edited by Brassel, K., International Geographical Union IGU, Commission on Geographic Information Systems, 820-829, 1990.

Hudak, P *et al.* . “Report on the functional programming language Haskell, Version 1.2.” *SIGPLAN Notices* 27 (1992):

Jones, Mark P. *An Introduction to Gofer*. Yale University, 1991. (available by anonymous ftp)

King, David J., and Wadler, Philip. “Combining Monads.” In *Functional Programming - Workshop Glasgow 1992*, ed. Ayr, S., Launchbury, J, and Sansom, P.M. Berlin: Springer Verlag, 1993.

Krantz, D.H. *et al.* . *Foundations of Measurement.* Vol. 1. New York: Academic Press, 1971.

Kuhn, W., and Frank, A.U. “A Formalization of Metaphors and Image-Schemas in User Interfaces.” In *Cognitive and Linguistic Aspects of Geographic Space*, ed. Mark, D.M., and Frank, A.U. 419-434. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991.

Langran, Gail, and Chrisman, Nicholas. “A Framework for Temporal geographic Information.”*Cartographica* 25 (3 1988): 1-14.

Luce, R. Duncan. “Semiorders and a theory of utility discrimination.” *Econometria* 24 (1956): 178 - 191.

Mark, David M., and Frank, Andrew U. *User Interfaces for GIS*. NCGIA, 1991.

Payne, S. J. “Interface Resources.” In *Designing Interaction*, ed. Caroll, J. M. 128-153. Cambridge: Cambridge Univ. Press, 1991.

Peyton Jones, Simon L, and Wadler, Philip. “Imperative functional programming.” In *ACM Symposium on Principles of Programming Languaes (POPL) in Charleston*, ACM, 71 - 84, 1993.

Reiter, R. “Towards a logical reconstruction of relational database theory,.” In *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, ed. M. M. L. Brodie Schmidt, M. L. Brodie, Mylopolous and Schmidt. 191-233. New York: Spri nger Verlag,, 1984.

Robert, Fred S. “Tolerance Geometry.” *NDJFAM* 14 (1 1973): 68-76.

Roberts, Fred S. “Indifference Graphs.” In *Proof Techniques in Graph Theory*, ed. Harary, F. 139 - 146. New York: Academic Press, 1969.

Roberts, Fred S., and Suppes, Patrick. “Some Problems in the Geometry of Visual Perception.” *Synthese* 17 (1967): 173 - 201.

Scott, Dana, and Suppes, Patrick. “Foundational Aspects of Theories of Measurements.” *Journal of Symbolic Logic* 23 (2 1958): 113 - 128.

Scott, D. S. “Logic and Programming Languages.” *Commuincation of ACM* 20 (9 1977):

Snodgrass, R.T. “Temporal Databases.” In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, ed. Frank, A.U., Campari, I., and Formentini, U. 22-64. 639. Heidelberg-Berlin: Springer-Verlag, 1992.

Stevens, S. S. “On the theory of scales of measurement.” *Science* 103 (2684 1946): 677 - 680.

Stoy, J. *Denotational Semantics*. Cambridge MA: MIT Press, 1977.