--

# Temporal Reasoning using Tesseral Addressing: Towards an Intelligent Environmental Impact Assessment System

*F. Coenen, B. Beattie, B. Diaz, T.J.M. Bench-Capon and M.J.R. Shave*

Department of Computer Science,
The University of Liverpool,
Chadwick Building, P.O. Box 147,
Liverpool L69 3BX, England.
email: frans@uk.ac.liverpool.compsci
Tel: 051 794 3698
Fax: 051 794 3715

*ABSTRACT*

A temporal reasoning system is described which uses a tesseral addressing referencing system. The tesseral addressing techniques used are directly compatible with raster based Geographic Information Systems (GIS) and by extension vector based GIS. As a result the temporal reasoning mechanism can be integrated into GIS systems and provide a temporal reasoning capability for such systems. Although the target application of the system described is the production of Environmental Impact Assessment reports it is considered that the advantages offered have a much wider field of application. The temporal reasoning approach advocated has similarities with Kowalski's event calculus in that the principal entities under consideration are temporally displaced events which have temporal attributes associated with them. Events are considered to be instantaneous; non-instantaneous events are described in terms of instantaneous start and end events. The temporal attributes which can be associated with events comprise sets of one or more ranges. Ranges may define a time period during which an event can be said to occur, in which case they are defined in terms of their earliest and latest times (tesseral address), or they may be used to allocate an absolute reference to an event, i.e. a range comprising a single tesseral address. Experiments to date have indicated that the system provides an excellent foundation on which to base further, multi-dimensional reasoning, work.

KEYWORDS: temporal reasoning, tesseral addressing, integration of KBS and GIS

**Temporal Reasoning using Tesseral Addressing: Towards an Intelligent Environmental Impact Assessment System**

*F. Coenen, B. Beattie, B. Diaz, T.J.M. Bench-Capon and M.J.R. Shave*

Department of Computer Science,
The University of Liverpool,
Chadwick Building, P.O. Box 147,
Liverpool L69 3BX, England.
email: frans@uk.ac.liverpool.compsci
Tel: 051 794 3698
Fax: 051 794 3715

## 1. INTRODUCTION

Environmental impact assessment is an important aspect of many engineering projects. A current EEC directive (Council Directive 85/337/EEC) requires that all major development projects which will have a major impact on the environment must submit, together with the planning application, an Environmental Impact Assessment (EIA) report. A significant proportion of the content of these reports comprises, or is derived from, geographic information. This in turn is usually obtained with support from computer systems collectively referred to as Geographic Information Systems (GIS). We can define GIS as computer systems that contain *"spatially referenced data that can be analysed and converted to information for a specific set of purposes or applications"*[1]. A severe limitation of these systems is that they support little or no reasoning capability concerning the spatial (and temporal) entities represented. This is because the representation techniques used by GIS (e.g. vector or rasta) are not directly compatible with the "KBS" representations used by existing spatio-temporal reasoning techniques (e.g. predicate logic). This limitation substantially restricts the usage of GIS, not only with respect to the production of EIA reports, but also with respect to many other applications.

A research project currently in progress at the University of Liverpool, the dynamic Geographic Knowledge Based Information System (dGKBIS) project, is investigating the application of KBS techniques to GIS. The aim is to develop spatio-temporal reasoning techniques which can be applied to GIS and thus enhance the potential of such systems. The principal target application for this work is to provide support for the production of EIA reports. However, it is expected that the results of the project will have much wider areas of application. Currently work is focussed on the temporal aspects of the spatio-temporal reasoning techniques to be applied to GIS. This has resulted in a tesseral temporal reasoning system, based on tesseral addressing and using tesseral arithmetic, which offers the advantage that it is directly compatible with existing GIS technology.

We have formalised the technique using the Horn clause subset of classical logic (with negation as failure). The result is an executable logic which we have implemented in the form of a PROLOG system. This system comprises an *event space*, an *eventbase* and an *inference engine*. The event space is a conceptual one dimensional space which is bounded by a start and end tesseral address and in which all events of interest are contained. The eventbase is a conventional database structure in which all information concerning the events which we wish to reason about are stored. Two types of data are represented, *event constraint* data detailing the constraints that exist between events, and configuration data concerned with the temporal attributes that may be associated with the events. The inference engine comprises a set of routines to determine such temporal configurations given a predefined set of event constraints.

All aspects of the system are fully described in this paper. We commence with a general discussion of events, points and intervals in Section 2 and of tesseral addressing in Section 3. We then go on, in Section 4, to describe the range and relational referencing options incorporated into the system and, in Sections 5 and 6, the inferencing strategy. The system's operation is illustrated in Section 7 with an example. Finally, in Section 8, some conclusions are drawn.

## 2. EVENTS, POINTS AND INTERVALS

Temporal reasoning is concerned with the deduction of the interrelationships between events displaced through time. There are two approaches; we can associate events with temporal entities and reason about the temporal entities themselves or we can associate temporal attributes with events and reason about the events. The first approach is exemplified by Allen's Interval Calculus[2] and the second by Kowalski and Sergot's Event Calculus[3].

Temporal entities can be considered to be either instantaneous temporal *points*, or *intervals* which have some duration associated with them. Both point and interval systems have been developed (and some combined systems, for example that of Jixin and Knight[4]). In interval based systems, such as Allen's interval calculus[2] and Freksa's spatial reasoning system[5], intervals are considered to be primitive; points are considered to be secondary and exist only where intervals meet. In point based systems points are considered to be primitive and intervals are considered secondary; intervals are defined in terms of their start and end points. Examples of such systems include those of Bruce[6], Dechter et al.[7] and Ladkin[8]. Advocates of the interval based approach maintain that points do not need to be represented. Allen argued that nothing can be true at a non-dimensional point and hence points need not be included in any general spatio-temporal theory[9]. Freksa makes a similar observation and suggests that (from a cognitive perspective) spatio-temporal entities must have a duration associated with them[5]. Galton[10], however, points out that in classical physics if a ball is thrown vertically upward there is an interval when the ball is travelling upwards and an interval when it is travelling downwards, but there is also a instantaneous point where the ball is stationary - hence any general temporal reasoning mechanism must be capable of representing both intervals and points.

In a similar manner events can be considered to be either instantaneous or have some duration associated with them, the precise length of which may or may not be known. Non-instantaneous events can of course be defined in terms of their instantaneous start and end points. The distinction between events and temporal attributes is that events can only have a single temporal point or interval associated with them, while temporal attributes can have any number of events associated with them.

For the tesseral temporal reasoning system described here we have adopted the second approach; we reason about events which have temporal attributes associated with them (for most applications they will also have non-spatio-temporal attributes). Instantaneous events are related to one another using the following primitive constraints:

```
∀ A B ∈ Ψ (Ta < Tb)    (A before B)
∀ A B ∈ Ψ (Ta = Tb)    (A equal B)
∀ A B ∈ Ψ (Ta > Tb)    (A after B)
∀ A B ∈ Ψ (Ta => Tb)    (A notBefore B)
∀ A B ∈ Ψ (Ta =|= Tb)    (A notEqual B)
∀ A B ∈ Ψ (Ta <= Tb)    (A notAfter B)
```

where $\Psi$ is the set of instantaneous events of interest and `Ta` and `Tb` are temporal attributes (tesseral addresses) associated with the events `A` and `B` respectively. Non-instantaneous events are then defined by relating their start and end points to each other using a before or after constraint. The above constraints can also be used to express all possible constraints between non-instantaneous events (this is described further in Sub-section 4.3).

Event constraints in our tesseral temporal reasoning system are incorporated into the eventbase as sets of facts which are utilised by the inference engine to generate possible temporal configurations for the events under consideration.

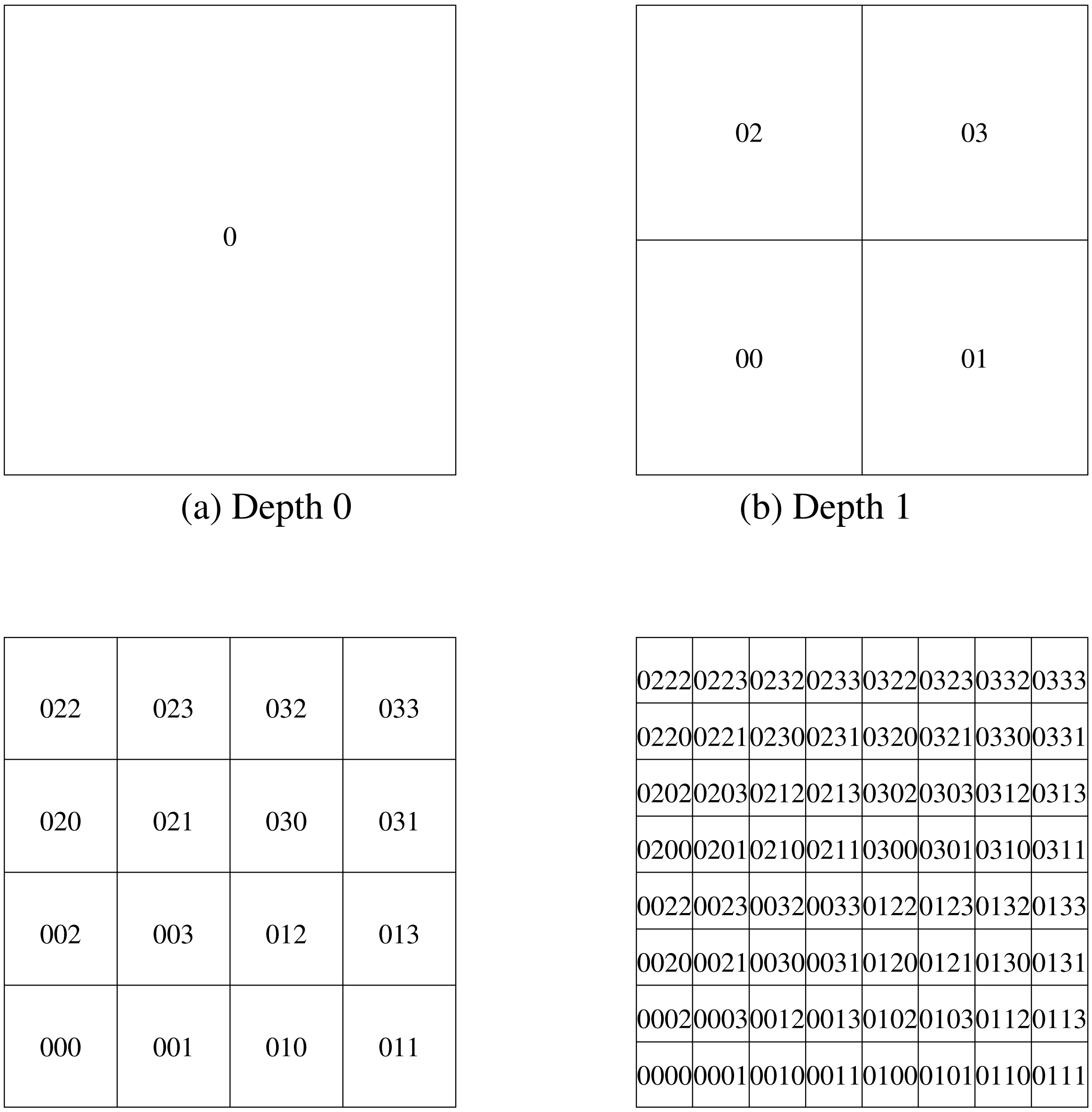
## 3. TESSERAL ADDRESSING

As noted in Section 1 spatial entities are represented in GIS using either polyline (vector) or raster techniques. Using polyline techniques the entities are represented using sets of connected vectors which in turn are defined using a standard Cartesian referencing system. Using the raster technique entities are represented using a uniform grid of cells or pixels. Polyline based systems are popular where the data to be represented can be considered to comprise many polygons defined in terms of points while raster systems are used where every feature in a given area needs to be recorded. Vectors can of course be defined in terms of unique cells or pixels and thus the two techniques can be integrated. It should also be noted that the cells or pixels used in raster based systems are themselves referenced, usually using some numbering system which is analogous to the Cartesian referencing system.

One numbering technique applicable to raster GIS is *tesseral addressing*. This is based on ideas concerning hierarchical clustering developed in the 1960s and 1970s to improve data access times[11], and atomic isohedral (same shape) tiling strategies developed in the 1970s and 1980s concerned with group theory[12]. The technique was first suggested as a suitable representation for GIS in the early 80s when the two strands were brought together and a *tesseral arithmetic* applied[13,14,15].

Tesseral addresses are generated in such a way as to reflect the tessellation process. One of the most straightforward tessellation techniques is illustrated in Figure 1. Here, we commence with a (2-dimensional) event space to which we assign the tesseral address 0 (Figure 1(a)). We then quarter it giving each quadrant a unique number (Figure 1(b)).

The process is continued in a hierarchical manner by quartering the quadrants and assigning addresses to these as shown in Figure 1(c) and then to those shown in Figure 1(d) and so on down to the desired grain size.




(a) Depth 0 (b) Depth 1

(c) Depth 2 (d) Depth 3

**Figure 1:** *Tesseral addressing*

Points to note:

- Each tesseral address associated with a tile is unique.
- All tesseral addresses (given a 2 dimensional space) are founded on the alphabet {0 .. 3}.
- Addresses can be stored with attributes in a database where each address acts as a single key to data.
- Tesseral addressing techniques map precisely onto raster GIS representations (and by extension vector representations).
- Tesseral addressing can be extended to three or four dimensions.
- The approach is supported by an arithmetic.

If, in Figure 1, we consider the X axis to represent time then we can move through the temporal dimension, at any specified depth of tesselation, simply by adding or subtracting the tesseral number 1 (`1`, `01`, `001`, to represent time we would need to add or subtract the tesseral number 2 (`2`, `02`, `002` ...). Extending this to a third dimension we would apply the number 4 (`4`, `04`, `004` ...), and given a fourth dimension the number 8 (`8`, `08`, `008` ...).

Remaining with two dimensions we can see that a "place" notation develops if there is a necessity to move outside the initial event space of interest. The immediate neighbours of tile `0` are numbered `1`, `2` and `3` in accordance with the tesseral pattern and the same pattern is then repeated at the next larger grain size (an event space for times as large). This is illustrated in Figure 2.

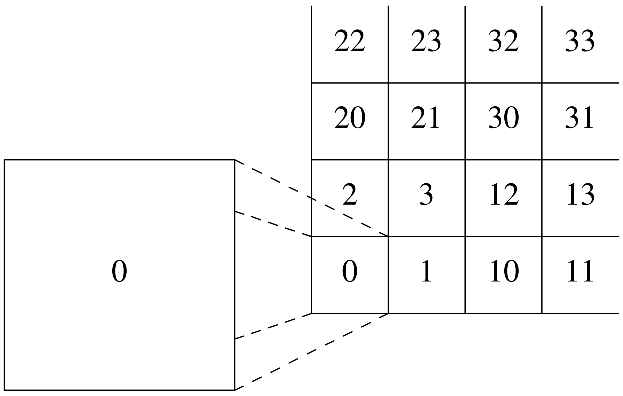



**Figure 2**: *Translating beyond the current event space*

We can make use of this concept to present a tesseral addition table as shown in Table 1. The table is entered with the desired operation and the operand to which the operation is supplied (note that the table also provides for translation along the prime-diagonal by adding or subtracting 3). Both the operations and the operand must be expressed as tesseral addresses and the operator must be interpreted at the level of precision of the operand. Thus, in Figure 1(d) if we wish to determine the tesseral address that will result if we move two tiles in the X direction from the address `0210` we must add the tesseral

| | +0 | +1 | +2 | +3 | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | |
| 1 | 1 | 10 | 3 | 12 | |
| 2 | 2 | 3 | 20 | 21 | |
| 3 | 3 | 12 | 21 | 30 | |

**Table 1:** *Addition Table*

number `0010` (the tesseral equivalent of 2 tiles to the "east"). Thus initially we add `0` to `0` (the righ- most digit of the operand) which, from Table 1, will result in `0`. Thus the right most digit of the result will be `0`. We then add the second digit of the operation, `1` to the second digit of the operand, `1`. We enter the addition table with the operation `+1` and the operand `1` and the value `10` is obtained. This indicates that the second digit of the result will also be `0` and that we need to add `1` to the third digit of the operand. Referring to Table 1 again this will result in `3`. The final result is thus `0300` as can be confirmed by reference to Figure 1(d). We can present this as follows:

```
Operand    0210
Operation   +10
           ----
Result     0300
           ====
Carry        1
```

Further examples:

```
Operand     0013      0023      0211
Operation     +1        +2        +3
            ----      ----      ----
Result      0102      0201      0302
            ====      ====      ====
Carry        11        22        11
```

To implement tesseral subtraction we would use a similar table and technique.

## 4. POINT REFERENCING

There are four principal methods whereby we can reference temporal points: absolute referencing, pseudo referencing, range referencing and relational referencing. These will be considered in more detail in the following Sub-sections.

### 4.1. Absolute and Pseudo Referencing

Absolute referencing involves the assignment of "real" values to a spatio-temporal entity. In a temporal sense this will be a time; we say that the temporal entity has been "time stamped". Absolute referencing has the following general advantages:

1. The referencing scheme is constant thus allowing simple numeric comparisons of references. For example to determine the distance between two references we subtract one from the other.
2. It results in a dense representation; a third reference can always be inserted between any two other references.

The disadvantage associated with absolute referencing schemes is that the necessary information must be available. Pseudo referencing is similar to absolute referencing except that some arbitrary referencing system is used, for example a set of sequential numbers. This still supports the concept of a dense representations but does not allow meaningful comparison between references.

The tesseral referencing technique used by the temporal reasoning system described here falls somewhere between these two types of referencing. The tesseral referencing system maps directly onto conventional notions of time. Thus the tesseral system still allows comparison between points, however, it has the inherent grain size dependent disadvantage that a third point cannot always be inserted between any two existing tesseral addresses. Where this occurs, i.e. the atomic level of tiling has been chosen incorrectly and a finer resolution is required, we have two options; we can use fractional tesseral addresses[16] or we can recast the tiling one level down. However, the issue is not addressed further in this paper.

### 4.2. Range Referencing

Given an absolute or pseudo referenced system any "reasoning" that we may wish to apply concerning attributes associated with spatio-temporal entities is reduced to simple database type queries. A much more powerful reasoning system will be able to reason using incomplete knowledge. This can be achieved using a range referencing system where a temporal range or *interval* is identified during which some event of interest may be said to occur. The ranges themselves must of course still be defined using absolute or, as in our case, pseudo references.

In the tesseral temporal reasoning system advocated here range information is contained in the event base in predicates of the form:

```
range(A,Ra).
```

where `A` $\in \Psi$, $\Psi$ is the set of events under consideration and `Ra` is the range associated with the event `A`. The ranges are defined in terms of a start and end tesseral address. In the absence of any information to the contrary the range is assumed to encompass the entire event space from a tesseral address of `0` to some maximum. This is expressed as follows:

```
range(A,[[0,Max]]).
```

where `Max` is the largest permitted tesseral address given a predefined event space.
It may be that a range comprises only a single tile, this is expressed thus:

```
range(E,[T1]).
```

where $T1 \in T$ and `T` is the set of tesseral addresses between `0` and `Max`. In this manner point information can be included in our tesseral temporal referencing system. Using ranges it is also possible to associate a number of ranges with an event. For example:

```
range(A,[[0,T1],T2,[T3,Max]]).
```

where $T1,\ T2,\ T3 \in T$. The above should be interpreted as stating that the event `A` occurs within the tesseral temporal range `0` to `T1`, or at `T2` or within the tesseral temporal range `T3` to `Max`.

### 4.3. Relational Referencing

In the absence of any other information, points may still be referenced by relating them to other points. We refer to this process using the term relational referencing. We define a relational reference as a *primitive temporal connection between two temporal entities that cannot be expressed by conjuncting and/or disjuncting other primitive connections.* A great deal of work has been done on one dimensional relational referencing, mostly concerned with temporal reasoning theories and techniques[2,17,18,19,20]. From this work we can identify four categories of relation references:

1. Point-point relations.
2. Point-interval relations.
3. Interval-point relations.
4. Interval-interval relations.

These categories are well established although there is still some argument as to how many relations exist under each heading[4]. However, given the above definition, we can demonstrate that there are three unique point-point relations, five point-interval relations, five interval-point relations and thirteen interval-interval relations. These are illustrated in Table 2. The labelling convention used for the interval-interval relations in Table 2, although in some cases ambiguous, is that first proposed by Allen[2] and subsequently adopted by many other authors in the field. We have thus also adopted this labelling convention for the purpose of this paper. The labels for the remaining relation categories have been chosen so as to achieve compatibility with Allen's labels.

It should be noted that the relations given in Table 2 also define all the possible constraints that can exist between events (see Section 2). Care should therefore be taken that event constraints and temporal relations are not confused, event constraints hold between events while temporal relations hold between temporal attributes. It should also be noted that all the above relationships can be expressed in terms of the three point-point relations.

## 5. INFERENCING

The inference engine considers each event constraint contained in the eventbase in turn and generates all possible temporal configurations accordingly. As this procedure progresses the possible configurations are refined, in a depth first manner, until there are no further constraints to consider. Possible configurations are presented in the form of a list:

```
[Ranges for event A, Ranges for event B ... Ranges for event N].
```

which is associated with a predicate *configuration*. The ranges associated with individual events are included using the notation given in Section 4.2. For example:

```
configuration([[A,[[020,031]]],[B,[021]] .... ]).
```

Given a constraint between two events `A` and `B` there are four possibilities:

1) Neither event exists.
2) Only event `A` exists.
3) Only event `B` exists.
4) Both events exist.

The term "exists" used here implies that the event indicated already has some temporal data associated with it by virtue of it being part of some previously considered constraint. Where an event has not been previously considered, i.e. it does not exist, in the absence of any information to the contrary it has a *start* range associated with it of `[0,Max]`.

Thus in the first of the above cases both events are initially assumed to have a *start* range of `[0,Max]` associated with them. In the case of an `equals` constraint there are no possible permutations to consider. In all other cases all permutations (see Section 6) must be considered so that all possible valid configurations can be generated. In the second two cases where an `equals` constraint is under consideration the temporal attributes of the previously unknown event will be identical to the existing event. In the case of a `notEquals` constraint the unknown event will have all those temporal attributes associated with it that are not associated with the recorded event. In the remaining cases the unknown event is assumed to have a *start* range of `[0,Max]` associated with it. This range, together with the temporal attributes associated with the known event, are then permuted so that all possible configurations for the constraint in question can be determined.

```
X before Y: X < Y                   X after Y: X > Y
X equals Y: X = Y
                        Point-point relations

X before PQ: X < (P < Q)            X after PQ: (P < Q) < X
X starts PQ: ((X = P) < Q)          X finishes PQ: (P < (X = Q))
X during PQ: (P < X < Q)
                       Point-interval relations

MN before Y: (M < N) < Y            MN after Y: Y < (M < N)
MN startedBy Y: ((M = Y) < N)       MN finishedBy Y: (M < (N = Y))
MN contains Y: (M < Y < N)
                       Interval-point relations

MN before PQ: (M < N < P < Q)       MN after PQ: (P < Q < M < N)
MN meets PQ: (M < (N = P) < Q)      MN metBy PQ: (P < (Q = M) < N
MN overlaps PQ: (M < P < N < Q)     MN overlappedBy PQ: (P < M < Q < N)
MN during PQ: (P < M < N < Q)       MN contains PQ: (M < P < Q < N)
MN starts PQ: ((M = P) < N < Q)     MN startedBy PQ: ((M = P) < Q < N)
MN finishes PQ: (P < M < (N = Q))   MN finishedBy PQ: (M < P < (N = Q))
MN equals PQ: ((M = P) < (N = Q))
                      Interval-interval relations

Where:
   {M, N, P, Q, X, Y} ∈ T
   T = The set of tesseral addresses between 0 and Max.
   (M < N) and (P < Q) define the intervals MN and PQ
```

**Table 2:** *Relational references*

Where both events exist and they are linked by an `equals` event constraint and the temporal attributes are identical then there is no need for any further processing. Otherwise, where this is not the case or where some other event constraint is under consideration, all permutation, given the ranges associated with the two events, must be determined so that all valid configurations are generated.

We can formalise the above process as a set of Horn clauses as shown in Table 3. In the table the connective should be read as a material implication. The  and  connectives have the obvious interpretation.

There are two further points to consider when generating configurations. Firstly we do not wish to produce duplicate results. Consider the following situation: the set of events `E = {A B C}` which are contained in a 4x4 event space of the form given in Figure 1(c) (i.e. there are four temporal slots in the temporal dimension which we will indicate by the identifiers `T0`, `T1`, `T2` and `T3`), and which have two possible configurations associated with them:

```
if (A equals B)  ¬(A Exists)  ¬(B exists)  (range(A,[[0,Max]]))
       (range(B,[[0,Max]])

if (A Constraint B)  ¬(Constraint = equals)  ¬(A Exists)
       ¬(B exists)  (range(A,permutations([[0,Max]],[[O,Max]])))
       (range(B,permutations([[0,Max]],[[0,Max]])))

if (A equals B)  (A Exists)  ¬(B exists)  (range(A,RangeA))
       (range(B,RangeA))

if (A notEquals B)  (A Exists)  ¬(B exists)  (range(A,RangeA))
       (range(B,E-RangeA))

if (A Constraint B)  ¬(Constraint = equals)  (A Exists)
       ¬(B exists)  (range(A,permutations(RangeA,[[O,Max]])))
       (range(B,permutations(RangeA,[[0,Max]])))

if (A equals B)  ¬(A Exists)  (B exists)  (range(A,RangeB))
       (range(B,RangeB))

if (A notEquals)  ¬(A Exists)  (B exists)  (range(A,E-RangeB))
       (range(B,RangeB))

if (A Constraint B)  ¬(Constraint = equals)  ¬(A Exists)
       (B exists)  (range(A,permutations([[O,Max]],RangeB)))
       (range(B,permutations([[0,Max]],RangeB)))

if (A equals B)  (A Exists)  (B exists)  (RangeA = RangeB)
       (range(A,RangeA))  (range(B,RangeB))

if (A equals B)  (A Exists)  (B exists)  ¬(RangeA = RangeB)
       (range(A,permutations(RangeA,RangeB)))
       (range(B,permutations(RangeA,RangeB)))

if (A Constraint B)  ¬(Constraint = equals)  (A Exists)
        (B exists)  (range(A,permutations(RangeA,RangeB)))
       (range(B,permutations(RangeA,RangeB)))
```

**Table 3:** *Configuration generation process*

```
configuration([[A,[[T0,T1]]],[B,[T2]],[C,[[T2,T3]]]])
configuration([[A,[[T0,T1]]],[B,[T2]],[C,[T3]]])
```

Then if we add the event constraint `B notEquals C` and permuting the first configuration this will produce the configuration:

```
configuration([[A,[[T0,T1]]],[B,[T2]],[C,[T3]]])
```

However, the same result will be produced when considering the second configuration. To prevent duplicate results whenever a new configuration is generated a check is made to ensure that it is not a duplicate. We can illustrate the above using a set of range tables as shown in Figure 3 where tables (a) and (b) show the two initial configurations and table (c) the final configuration that can be produced from each of the start configurations. In each table the columns represent the sequence of temporal addresses T0 through to T3. The possible temporal locations for the events A and B are then indicated by the shaded tesselations.

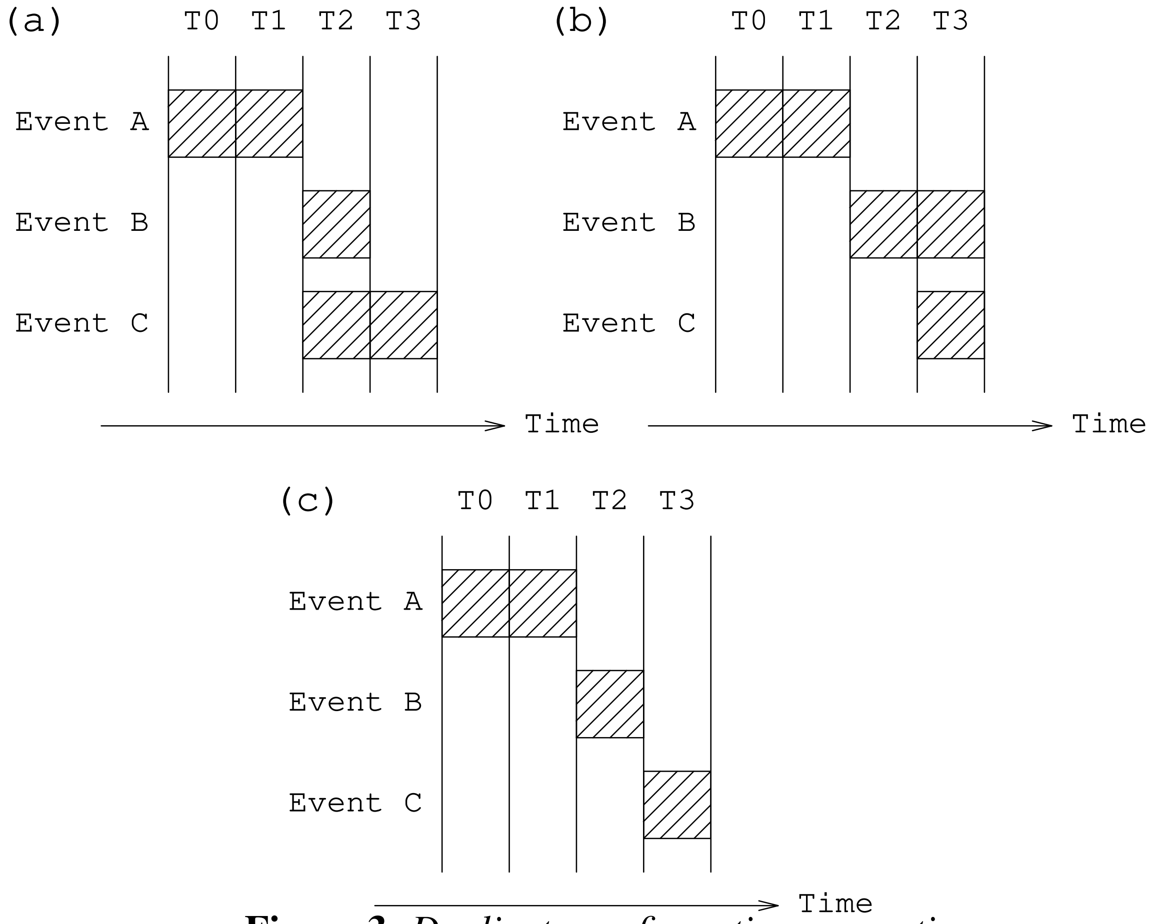



**Figure 3:** *Duplicate configuration generation*

Secondly whenever a set of ranges associated with an event is adjusted or added to we must ensure that any other events with which this event is linked through an `equals` constraint are also updated, i.e. the adjustment must be propagated through to these events. For example, given the following two constraints:

```
A equals B
B equals C
```

Any changes made to the temporal attributes associated with event C by virtue of some other constraint must be propagated through to events B and A.

## 6. RANGE PERMUTATIONS

To determine all possible configurations we can allocate each available tesseral address in the set T to each event in the set E in turn and test this against all the existing constraints using a traditional "generate and test" paradigm (see Bratko[17] for examples). Given that the number of possible tesseral addresses that may be associated with an event may be very large this is not computationally desirable. The use of ranges allows us to restrict the number of permutations; for example in the absence of any information to the contrary an event A is assumed to have only one eligible range associated with it, `event(A,[[0,Max]])`. We can further restrict the number of permutations that need to be generated by adopting a number of "heuristics" to reduce the search "space" according to the event constraint under consideration and the relational references that can exist between pairs of ranges (see Table 2). It is also anticipated that, given a real life problem, the number of temporal attributes associated with events will be severely restricted. In the following Sub-sections we will consider the possible permutations that can be generated given a specific event constraint by considering each event constraint supported by the system in turn.

To illustrate the permutation generation mechanism in each case we will use range tables of the form introduced in the previous Section, however, we will assume an 8x8 event space as shown in Figure 1(d). Thus each table will comprise eight columns representing a sequence of temporal units expressed in the form of a sequence of tesseral addresses indicated by the identifiers T0 through to T7 which might indicate any of the horizontal tesseral sequences given in Figure 1(d); for example 0000, 0001, 0010, 0011, 0100, 0101, 0110 and 0111. The possible temporal locations for events are again indicated by shading the appropriate tesseral slots.

### 6.1. Before

Consider the two ranges for two hypothetical events A and B given in Figure 4. Using the notation defined in Sub-section 4.2, the ranges associated with each can be expressed as:

```
range(A,[[T0,T1],[T3,T4],[T6,T7]]).
range(B,[[T1,T3],T5,T7]).
```

The possible range permutations, given the knowledge that event A occurs temporally before event B, can be deduced by moving through the range of tesselations in a positive direction and continuously partitioning the ranges in such a way that at least one range exists on the left hand side of the partition for event A and at least one range on the right hand side of the partition for event B. Thus partitions which will produce an empty range for at least one of the events involved are not valid. Further, partitions which result in a duplicate result, or a result that is a more specialised form of a previously generated result (or a result that will be generated) are invalid. In the figure partitions are indicated by the double headed arrows.

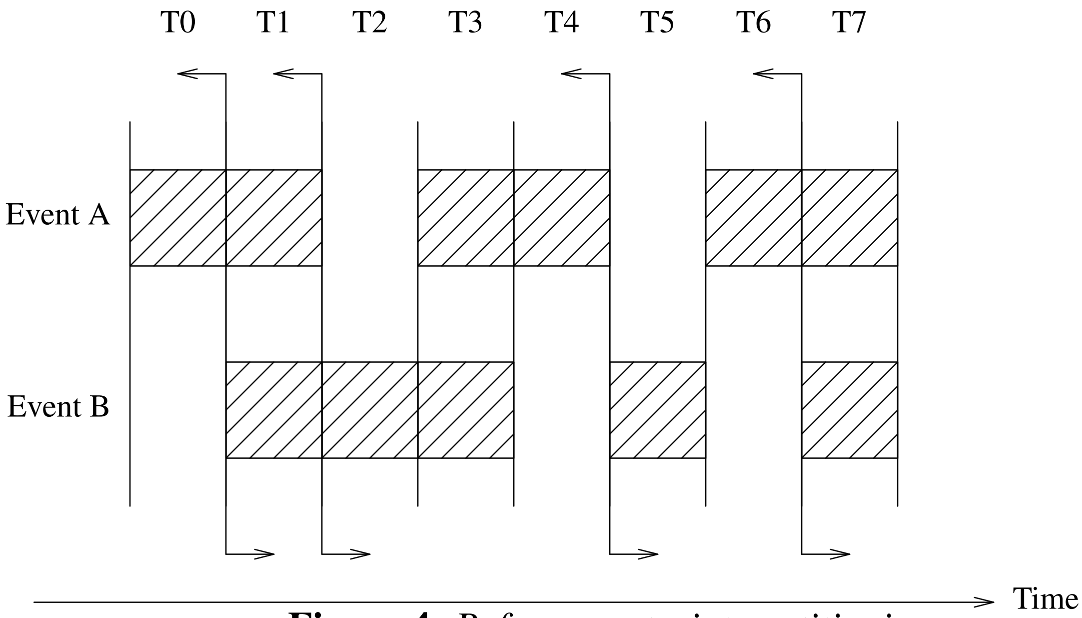



**Figure 4:** *Before constraint partitioning*

Thus, referring to Figure 4, the first valid partition will be at the leading edge of tesselation `T1`, this will have the effect of producing the following configuration for events `A` and `B` (given the constraint `A before B`):

```
range(A,[T0]).
range(B,[[T1,T3],T5,T7]).
```

The next valid partition will then be located at the leading edge of tesselation `T2` which will result in the configuration:

```
range(A,[[T0,T1]]).
range(B,[[T2,T3],T5,T7]).
```

If we now partition at the following tesselation, `T3`, this will result in a range of `[[T0,T1]]` for event `A` and a range of `[T3,T5,T7]` for Event `B`. Inspection of the range for event `B` indicates that this is simply a more specialised form of the previously generated permutation. The permutation is therefore superfluous and thus considered to be invalid. If we now partition at `T4` inspection of Figure 4 indicates that the result will be a more specialised form of the result that would be obtained if we partitioned at `T5`. This partition is also superfluous. The next valid partition is therefore at the leading edge of tesselation `T5` which will produce ranges of:

```
range(A,[[T0,T1],[T3,T4]]).
range(B,[T5,T7]).
```

Continuing the process, the remaining valid partition is at the leading edge of tesselation `T7` which will result in the following sets of ranges:

```
range(A,[[T0,T1],[T3,T4],T6]).
range(B,[T7]).
```

The simplest method of implementing this partitioning is in accordance with the possible relational references that can exist between points and/or intervals as discussed in Sub-section 4.3. Using relational referencing techniques we can formalise the partitioning using a set of Horn clauses of the form presented in Appendix 1.

### 6.2. After

We can adopt a similar approach for the `after` event constraint to that described in the previous Sub-section for the `before` constraint. Again partitions which will produce an empty range for at least one of the events involved, or which result in a duplicate set of ranges or a more specialised set of ranges than that of a previously deduced set of ranges (or a more specialised set of ranges than that of a later set of ranges) are invalid. Thus if we consider the two ranges given in Figure 5 for the events `A` and `B`, and given the constraint `A after B`, the first valid partition will occur at the leading edge of tesselation `T3`. This will produce the ranges:

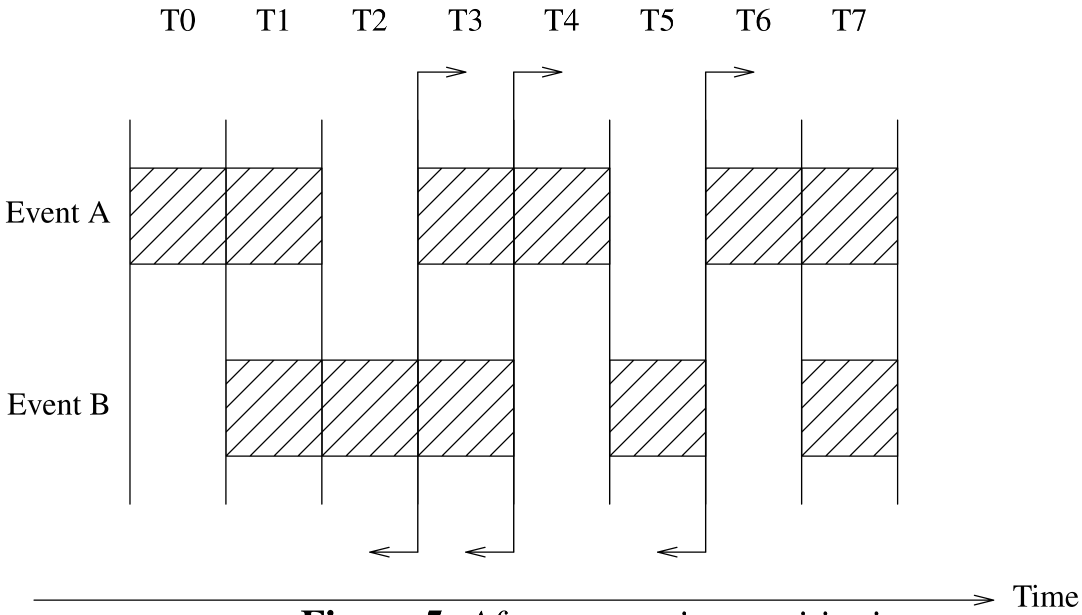



**Figure 5:** *After constraint partitioning*

```
range(A,[[T3,T4],[T6,T7]]).
range(B,[[T1,T2]]).
```

Continuing the process the following valid partitions will occur at the leading edges of tesseral addresses `T4` and `T6`, giving rise to the following two sets of ranges respectively:

```
range(A,[T4,[T6,T7]]).
range(B,[[T1,T3]]).

range(A,[[T6,T7]]).
range(B,[[T1,T3],T5]).
```

Again can formalise this process using a set of Horn clauses such as those given in Appendix 2.

### 6.3. Equals

Given two sets of ranges associated with two events `A` and `B` and the knowledge that the events are related through an `equals` constraint there is only one possible set of ranges that can satisfy this constraint, this is the intersection of the two sets, `(RangeA ∩ RangeB)`. Thus given the two ranges given in Figure 6 there are three tesseral addresses which occur in both ranges (indicated in the Figure by upright rectangles). As a result the ranges for events `A` and `B` will become respectively:

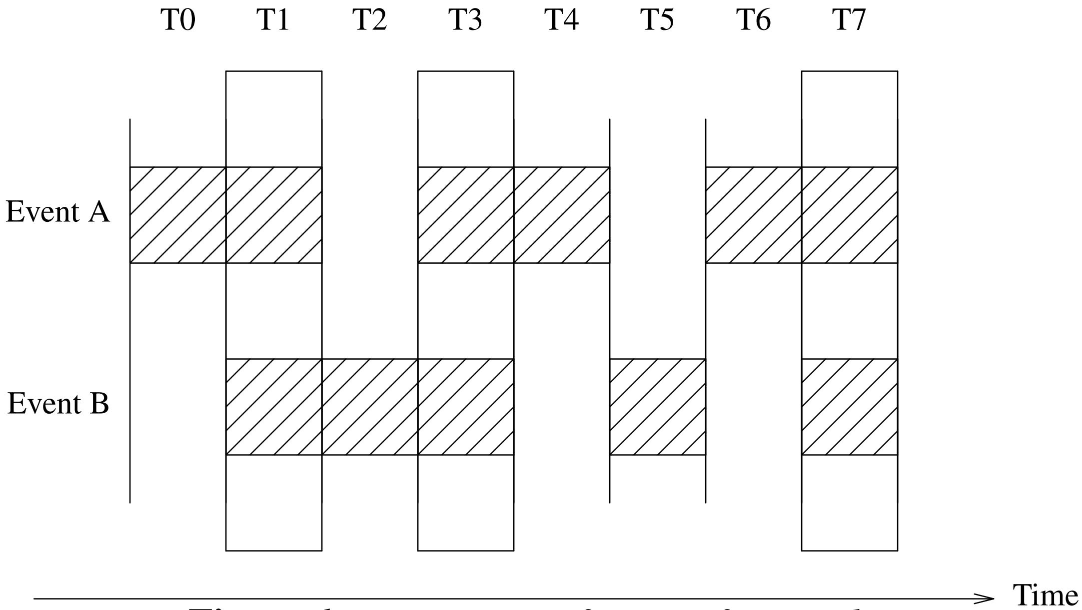



**Figure 6:** *Intersection of ranges for equal constraint*

```
range(A,[T1,T3,T7]).
range(B,[T1,T3,T7]).
```

A point to note here is that the equal event constraint will usually reduce the number of locations that can be attached to an event and will never serve to increase the number of range permutations. This makes the constraint much more economic than the `before` and `after` constraints described in the foregoing two subsections. The generation of intersections is also comparatively simple to implement.

### 6.4. NotBefore

To determine the possible range permutations for the `notBefore` event constraint we partition the ranges associated with the two events under consideration in a similar manner to that used to determine the possible range permutations associated with the `before` and `after` event constraints described earlier. The necessary partitioning given constraint `A notBefore B` is illustrated in Figure 7. As a result the following sets of ranges will be obtained:

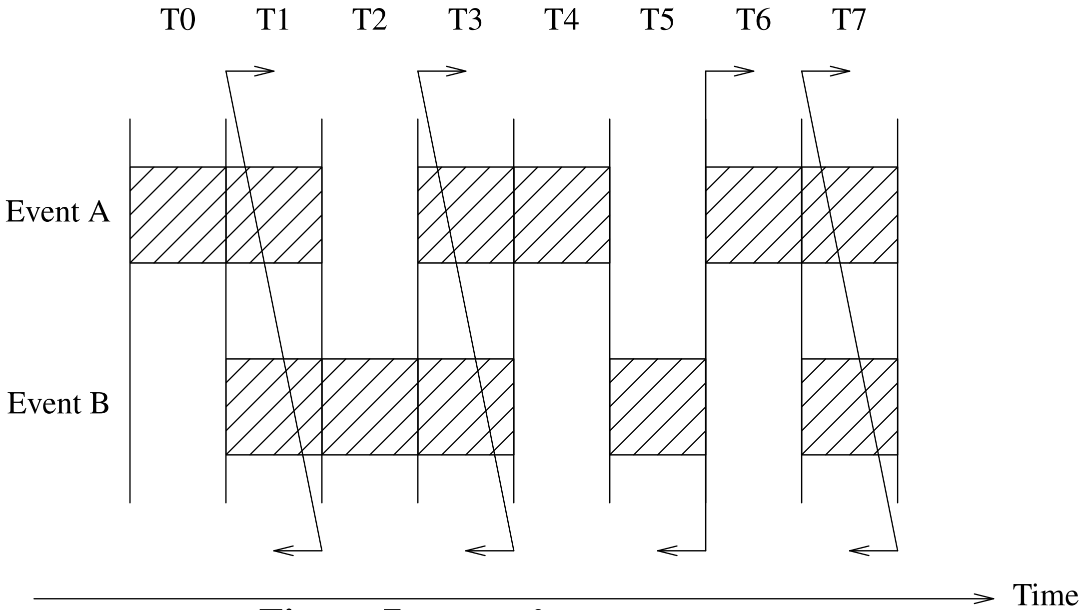



**Figure 7:** *NotBefore constraint partitioning*

```
range(A,[T1,[T3,T4],[T6,T7]]).
range(B,[T1]).

range(A,[[T3,T4],[T6,T7]]).
range(B,[[T1,T3]]).

range(A,[[T6,T7]]).
range(B,[[T1,T3],T5]).

range(A,[T7]).
range(B,[[T1,T3],T5,T7]).
```

### 6.5. NotAfter

The possible range permutations for the `notAfter` event constraint given `A notAfter B` are shown in Figure 8. The resulting permutations are as follows:

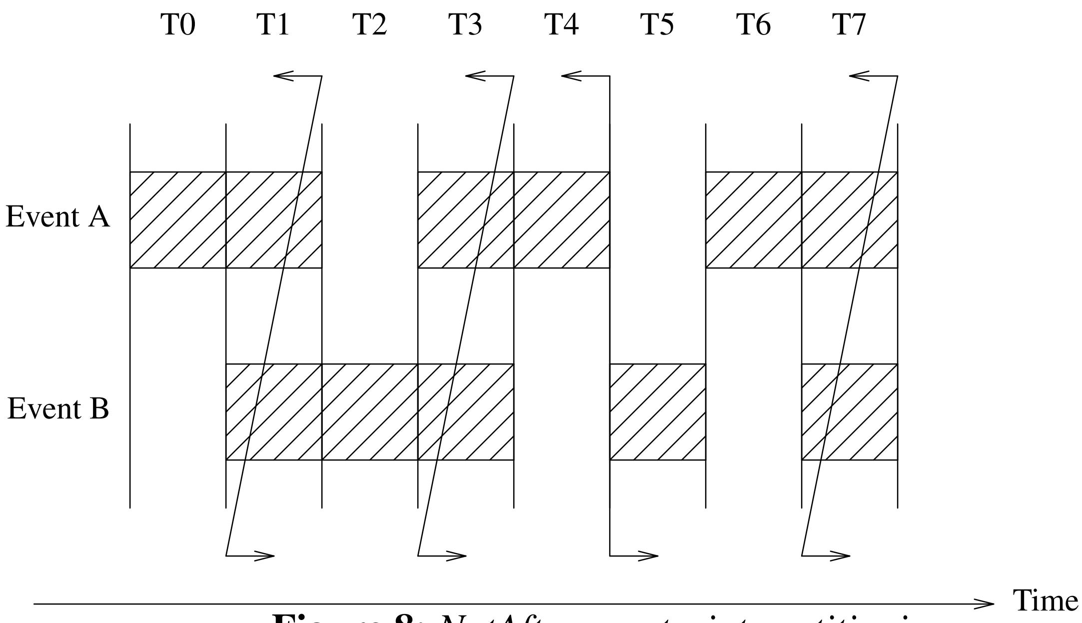



**Figure 8:** *NotAfter constraint partitioning*

```
range(A,[[T0,T1]]).
range(B,[[T1,T3],T5,T7]).

range(A,[[T0,T1],T3]).
range(B,[T3,T5,T7]).

range(A,[[T0,T1],[T3,T4]]).
range(B,[T5,T7]).

range(A,[[T0,T1],[T3,T4],[T6,T7]]).
range(B,[T7]).
```

### 6.6. NotEquals

The `notEquals` constraint is the most expensive for which to determine all possible range permutations. Essentially given a constraint `A notEquals B` this states that the event `A` can have any set of tesseral ranges (in the same dimension) associated with it other than those already associated with the event `B`. However, returning to Figure 6, we can note that we need only consider the subset of ranges equal to the intersection `(RangeA ∩ RangeB)`. From Figure 6:

```
RangeA ∩ RangeB = [T1,T3,T7]
```

This reduces the number of possible ranges for event `A`, however the number of partitions is still substantial, namely:

```
[T1,T3,T7]
[T1,T3]
[T1,T7]
[T3,T7]
[T1]
[T3]
[T7]
[]
```

The possible sets of ranges for events `A` and `B` can be determined using the following identities:

```
NewRangeA = RangeA - (RangeA ∩ RangeB) +
                        permutations(RangeA ∩ RangeB)
NewRangeB = RangeB - permutations(RangeA ∩ RangeB)
```

Returning to Figure 6 the possible range permutations which will result are as follows:

```
range(A,[[T0,T1],[T3,T4],[T6,T7]]).
range(B,[T2,T5]).

range(A,[[T0,T1],[T3,T4],T6]).
range(B,[T2,T5,T7]).

range(A,[[T0,T1],T4,[T6,T7]]).
range(B,[[T2,T3],T5]).

range(A,[T0,[T3,T4],[T6,T7]]).
range(B,[[T1,T2],T5]).

range(A,[[T0,T1],T4,T6]).
range(B,[[T2,T3],T5,T7]).

range(A,[T0,[T3,T4],T6]).
range(B,[[T1,T2],T5,T7]).

range(A,[T0,T4,[T6,T7]]).
range(B,[[T1,T3],T5]).

range(A,[T0,T4,T6]).
range(B,[[T1,T3],T5,T7]).
```

## 7. OPERATION

Most EIA report problems involve more than one dimension and a substantial event space. It is therefore not possible to describe the system's operation using a genuine problem. We will thus use a contrived scenario. We will consider a 4x4 event space of the form given in Figure 1(c) where the temporal dimension is indicated by the X-axis. In particular we will assume that the minimum tesseral address of an event is `000` and the maximum is `011`. Thus in the absence of any information to the contrary an event that takes place in the event space can occur over a range of four tesseral addresses - `000`, `001`, `010` and `011`. Let us now assume the existence of the following events:

$\{A\ B\ C\ D\ E\ F\} \in \Psi$

which are related through the following event constraints:

```
A before B
C after B
D equals A
E notBefore A
F notAfter C
F notEquals B
```

Although the tesseral temporal reasoning system that we have developed operates in a depth first manner, for illustrative purposes we will consider its operation in a breadth first style commencing with the first constraint and finishing with the last. There will therefore be six "layers" to the derivation of possible temporal configurations as shown in Figure 9. In the Figure we again use range tables of the form introduced in Sections 5 and 6. In each table the columns represent, from left to right, the tesseral addresses `000`, `001`, `010` and `011`. The rows represent each event with the shaded sections representing possible tesseral addresses/temporal locations for the events.

Thus commencing with the first constraint the possible configurations are given in Figure 9(a). Initially, of course, we know nothing about either event `A` or `B` thus both are assumed to have a range of tesseral addresses covering all the addresses available in our event space, i.e. `000` to `011`. Then using the routine presented in Appendix 1, this will give rise to the permutations given in Figure 9(a).

Applying the following constraint, `C after B`, where `C` is a new event which thus has the default range `000` to `011` associated with it, we obtain a set of configurations as shown in Figure 9(b). The first two configurations are derived (using the routine given in Appendix 2) from the first configuration given in Figure 9(a) and the third configuration in Figure 9(b) from permuting the second configuration in Figure 9(a). Note that the third configuration in Figure 9(a) is no longer valid because the given range for event `B` can not be permuted in any way to allow the inclusion of event `C`.

If we now consider the following constraint, `D equals A`, this can simply be interpreted as event `D` has the same tesseral addresses associated with it as event `A`. The resulting configuration is given in Figure 9(c).

The following event constraint is, `E notBefore A`, where `E` is a new event. The resulting configurations are as shown in Figure 9(d). The third and fourth configurations given in the figure are permutations derived, from the third configuration given in Figure 9(c). Note also that where the range associated with event `A` has been adjusted/permuted this adjustments/permutations has been propagated through to event `D`.

If we now consider the following constraint, `F notAfter C`, which introduces the event `F`, there are five valid configurations as shown in Figure 9(e). The first two are permutations generated from the first configuration in Figure 9(d). The following three configurations are extensions from the last three configurations in Figure 9(d) respectively.

The final constraint to be considered is `F notEquals B`. Both these points exist and thus the ranges for both must be permuted to arrive at the possible configurations. The (final) result is that given in Figure 9(f).

## 8. CONCLUSIONS

A tesseral temporal reasoning system has been described. This can be used to reason about temporally displaced entities with the particular aim of providing support for EIA report production but is expected to have a much wider impact for GIS applications. It should be noted that the reasoning impinges on only those GIS entities of interest, we are not suggesting that the system be used to manipulate all the slots in a GIS. Experiments to date have indicated that the system described here forms an excellent foundation on which to build. Currently work is in progress to extend the system to permit two

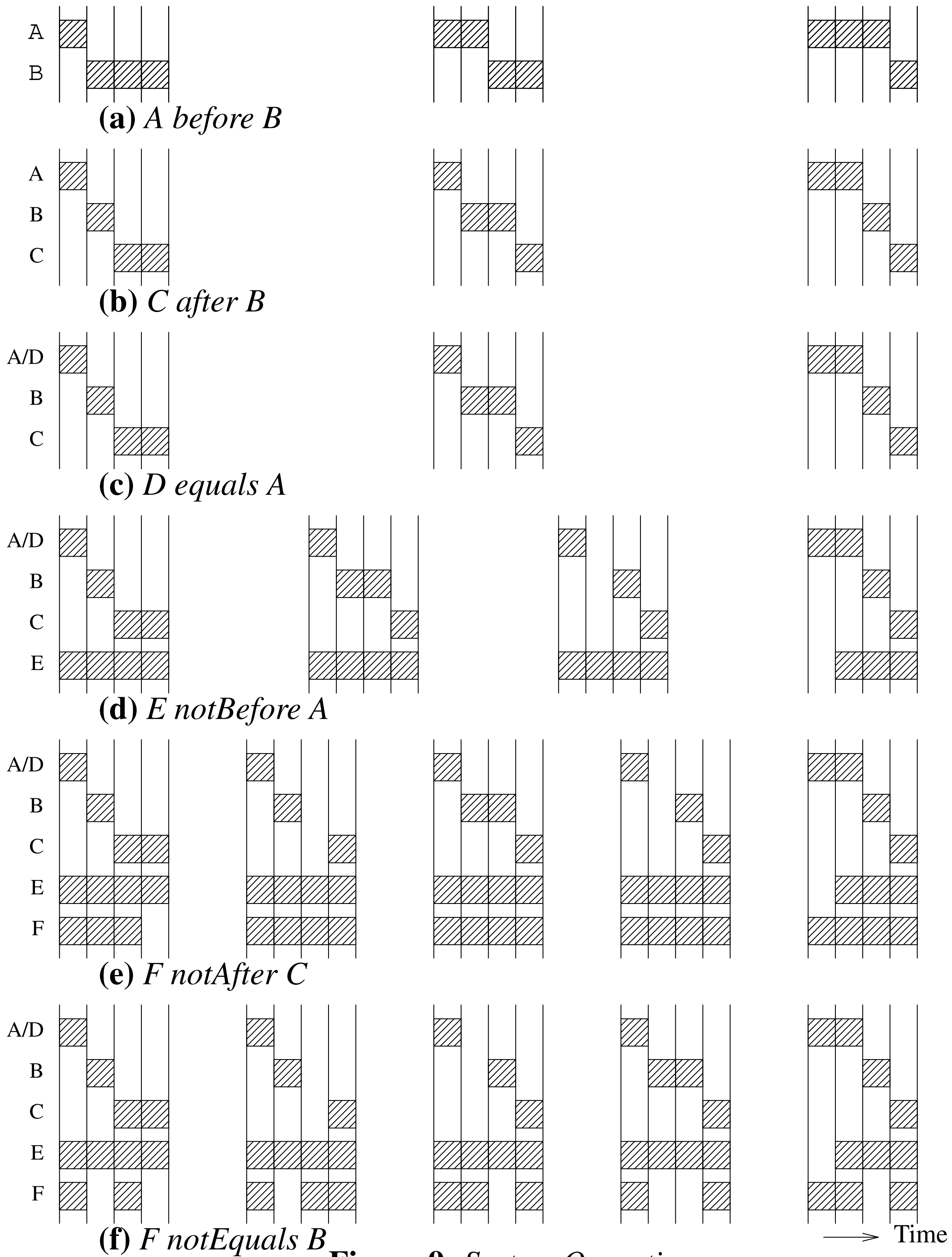



**Figure 9:** *System Operation*

dimensional reasoning and to incorporate the concept of magnitudes into the labelling mechanism so that points can be constrained as being some fixed "distance" before/after other points. The latter will then allow the system to reason about non-instantaneous events of a fixed duration.

## APPENDICES

In the following two appendices routines to implement the permutation techniques described in Sections 6.1 and 6.2 to determine `before` and `after` event constraint permutations are presented (there is not sufficient space in this paper to present routines to determine the `equals`, `notBefore`, `notAfter` and `notEquals` permutations). The two routines are expressed in terms of Horn clauses (with negation as failure) and should be interpreted in the normal "prolog" manner. Points to note:

1. The appendices include a function `relation` which is used to identify the nature of the relational references between two individual "ranges", i.e. point-point, point-interval, interval-point or interval-interval.
2. The function `NewRange` included in some of the clauses converts a range comprised of the same start and end tesseral addresses into a single tesseral reference. Thus `NewRange(T0,T0)` will return `T0`, while `NewRange(T0,T1)` will return `[T0,T1]`.
3. All addition and subtraction is tesseral addition and subtraction (see Section 3).
4. Whenever we are interested in only the first element contained in a list of temporal attributes associated with a particular event we will adopt the declarative programming convention for destructing and constructing lists of using an infix operator to divide the list, this will be indicated using the UNIX pipe symbol |. For example:

   ```
   range(A,[[0,T1]|Tail]).
   ```

   Further, where the precise nature of an element in a range list is irrelevant we will indicate this using the universal variable symbol _ (underscore).
5. Where we wish to distinguish between (say) the point-interval relation `starts` and the interval-interval relation `starts` we will do so using the labels `pir_starts` and `iir_starts`.
6. The => connection should be interpreted as a material implication. The and `as` connectives have the obvious interpretation.

## APPENDIX 1 - A before B

```
1)  before(1,[],RangeB,[],RangeB).

2)  before(_,[Ra|RangeA],[Rb|RangeB],[Ra|NewRangeA],NewRangeB)
        (relation(Ra,Rb) = before)
        before(1,RangeA,[Rb|RangeB],NewRangeA,NewRangeB).

2)  before(1,[Ra|_],[Rb|RangeB],[],[Rb|RangeB])
        (relation(Ra,Rb) = after  metBy  overlappedBy
        during  starts  startedBy  finishes  equals).

3)  before(_,[Ra|RangeA],[_|RangeB],NewRangeA,NewRangeB)
        (relation(Ra,Rb) = after  metBy  finishedBy  pir_finishes)
        before(0,[Ra|RangeA],RangeB,NewRangeA,NewRangeB).

4a) before(_,[[M,N]|_],[[P,Q]|RangeB],[NewRange(M,P-1)],[[P,Q]|RangeB])
        (relation([N,M],[P,Q]) = meets  overlaps  iir_contains
        iir_finishedBy).

4b) before(_,[[M,N]|_],[Y|RangeB],[NewRange(M,Y-1)],[Y|RangeB])
        relation([N,M],Y) = ipr_finishedBy  ipr_contains).

5)  before(_,[Ra|RangeA],[[P,Q]|RangeB],NewRangeA,NewRangeB)
        (relation([N,M],[P,Q]) = meets  overlaps  iir_contains
        iir_finishedBy  equals)
        before(0,[Ra|RangeA],[NewRange(P+1,Q)|RangeB],NewRangeA,NewRangeB).

6a)  before(_,[[M,N]|RangeA],[[P,Q]|RangeB],NewRangeA,NewRangeB)
        (relation([N,M],[P,Q]) = overlappedBy  during  iir_starts
        startedBy  iir_finishes)
        before(0,[[M,N]|RangeA],[NewRange(M+1,Q)|RangeB],NewRangeA,NewRangeB).

6b) before(_,[X|RangeA],[[P,Q]|RangeB],NewRangeA,NewRangeB)
        (relation(X,[P,Q]) = pir_starts  pir_during)
        before(0,[[M,N]|RangeA],[NewRange(X+1,Q)|RangeB],NewRangeA,NewRangeB).
```

## APPENDIX 2 - A after B

```
1)  after(1,RangeA,[],RangeA,[]).

2)  after(_,[Ra|RangeA],[Rb|Range2],NewRangeA,[[P,Q]|NewRangeB])
        (relation(Ra,Rb) = after)
        after(1,[Ra|RangeA],Range2,NewRangeA,NewRangeB).

3)  after(1,[Ra|RangeA],[Rb|_],[Ra|RangeA],[])
        (relation(Ra,Rb) = before  meets  overlaps  contains
        starts  startedBy  finishedBy  equals).

4)  after(_,[Ra|RangeA],[Rb|Range2],NewRangeA,NewRangeB)
        (relation(Ra,Rb) = before  meets  ipr_starts  pir_during
        ppr_equals)  after(0,RangeA,[Rb|Range2],NewRangeA,NewRangeB).

5)  after(_,[[M,N]|RangeA],[[P,Q]|_],[[M,N]|RangeA],[NewRange(P,M-1)])
        (relation([M,N],[P,Q]) = metBy  overlappedBy  irr_during
        iir_finishes).

    after(_,[X|RangeA],[[P,Q]|_],[X|RangeA],[NewRange(P,X-1)])
        (relation([X,[P,Q]) = pir_during  pir_finishes).

6)  after(_,[[M,N]|RangeA],[Rb|Range2],NewRangeA,NewRangeB)
        (relation([M,N],Rb) = metBy  overlappedBy  iir_during
        iir_starts  startedBy  finishes  iir_equals)
        after(0,[[NewRange(M+1,N)]|RangeA],[Rb|Range2],NewRangeA,NewRangeB).

7a) after(_,[[M,N]|RangeA],[[P,Q]|Range2],NewRangeA,NewRangeB)
        (relation([M,N],[P,Q]) = overlaps  iir_contains  finishedBy)
        after(0,[[NewRange(P+1,N)]|RangeA],[[P,Q]|Range2],NewRangeA,NewRangeB).

    after(_,[[M,N]|RangeA],[Y|Range2],NewRangeA,NewRangeB)
        (relation([M,N],Y) = ipr_contains)
        after(0,[[NewRange(Y+1,N)]|RangeA],[Y|Range2],NewRangeA,NewRangeB).

8)  after(_,[Ra|RangeA],[Rb|Range2],NewRangeA,NewRangeB)
        (relation(Ra,Rb) = ipr_finishedBy  ipr_contains)
        after(0,RangeA,[Rb|Range2],NewRangeA,NewRangeB).
```

## REFERENCES


1. **Parent, P J** *Geographic Information Systems: Evolution, academic involvement and issues arising from the proliferation of information* Master's thesis, University of California, Santa Barbara, (1988)
2. **Allen, J F** 'Maintaining knowledge about temporal intervals' *Communications of the ACM* Vol 26 No 11 (1983) pp832-843
3. **Kowalski, R and Sergot, M** 'A Logic-based Calculus of Events' *New Generation Computing* Vol 4(1986) p67-95
4. **Jixin, M and Knight, B** 'A General Temporal Theory' *The Computer Journal* Vol 37 No 2 (1994) pp114-123
5. **Freksa, C** 'Temporal reasoning based on semi-intervals' *Artificial Intelligence* Vol 54 (1992) pp199-227
6. **Bruce, B C** 'A model for temporal references and application in a question answering program' *Artificial Intelligence* Vol 3 (1972) pp1-25
7. **Dechter, R, Meiri, I and Pearl, J** 'Temporal constraint networks' *Artificial Intelligence* Vol 49 (1991) pp61-95
8. **Ladkin, P** 'Effective solution of qualitative interval constraint problems' *Artificial Intelligence* Vol 52 (1192) pp105-124
9. **Allen, J F and Hayes, P J** 'Moments and points in an interval-based temporal logic' *Computat. Intell. (Canada)* Vol 5 (1989) pp225-238
10. **Galton, A** 'A critical examination of Allen's theory of action and time' *Artificial Intelligence* Vol 42 (1990) pp159-188
11. **Morton, G M** *A Computer Oriented Geodetic Data Base, and a New Technique in File Sequencing* IBM Canada Ltd. (1966)
12. **Grunbaum, B and Shephard, G C** *Tilings and Patterns* Freeman, New York (1987)
13. **Holroyd, F C** 'The Geometry of Tiling Hierarchies' *Ars Combinatoria* Vol 16B (1983) pp211-244
14. **Bell, S B M, Diaz, B M, Holroyd, F C and Jackson, M J J** 'Spatially Referenced Methods of Processing Raster and Vector Data' *Image and Vision Computing* Vol 1 No 4 (1983) pp211-220
15. **Diaz, B M and Bell, S B M** *Spatial Data Processing using Tesseral Methods* Natural Environment Research Council publication, Swindon, England (1987)
16. **Diaz, B M** 'Tabular Tesseral Methods and Calculators' in **Diaz, B M and Bell, S B M (eds)** *Spatial Data Processing using Tesseral Methods* Natural Environment Research Council publication, Swindon, England, (1987), pp832-843
17. **Hamblin, C L** 'Instants and intervals' *Studium Generale* Vol 24 (1971) pp127-134
18. **Vilain, M B** *A system for reasoning about time* Proceedings AAAI'82, (1982) pp197-201
19. **Van Benthem, J F A K** *The logic of time* Reidel, Dordrecht, Netherlands (1983)
20. **Vilain, M B and Kautz, H** *Constraint propagation algorithms for temporal reasoning* Proceedings AAAI'86, Morgan Kaufmann, Los Altos, CA, (1986) pp377-382

21. **Bratko, I** *PROLOG Programming for Artificial Intelligence* Addison-Wesley, New York (1986)