An Interval-based Temporal Algebra Based on Binary Encoding of Point Relations

Vincent J. Kovarik Jr.
Software Technology, Inc.
1225 Evans Road
Melbourne, FL 32904
Vince_Kovarik@mlb.sticomet.com

Avelino J. Gonzalez
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816

ABSTRACT

This paper presents a method for representing temporal interval relations using a bit-encoded form of the relationships between interval end-points. The set of bit patterns for each interval relationship yields a unique, single-byte signature that forms the basis of a binary temporal algebra. Also presented is a matrix multiplication algorithm for computing transitive relations based on the definition of sum and product operations for the bit-encoded relation signatures. This bit-encoding encompasses the representation of unknown relations between endpoints of two intervals and captures ambiguities within a temporal system while providing an efficient binary algebra. Finally, an algorithm to compute the transitive closure over a set of intervals forming a temporal system is presented. The algorithm's complexity is analyzed and is $O(n^3)$, worst case, where n is the number of temporal intervals within the system. Empirical observations indicate that the closure algorithm completes in $O(n^2)$ time, on average. The small memory footprint for the bit-code, the algorithmic transitive relation calculation, and the closure algorithm, together, form an efficient method for providing machine-based temporal reasoning capabilities.

1. Introduction

In [Allen, 1984], a temporal logic based on intervals was proposed. An interval-based approach provides a more accurate model of actions and events in the world. The representation is based on the premise that events take temporal "space" and are not simply point-based, as in [Vilain, 1982]. Restricting a temporal representation to absolute points limits the range of problems that the system is capable of addressing. The interval-based approach has been the catalyst for many other research efforts that view temporal relationships in the same light. This comprehensive treatment evolved out of earlier papers [Allen, 1983a and Allen, 1983b] which first defined the temporal logic. This logic, based on a typed, first-order predicate calculus, consists of six basic relationships, their inverses, plus the identity relationship (=), as illustrated in Figure 1. These interval relationships capture the relationships between two intervals in a logical and qualitative fashion.

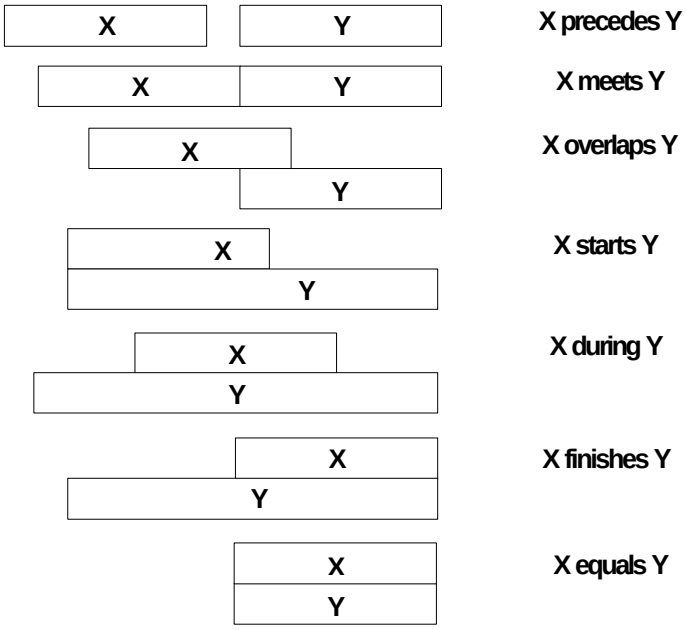

Figure 1. Fundamental Temporal Relationships

Table 1 below provides a reference for each relationship, its inverse and the abbrevations used through out this paper.

Relationship	Abbreviation	Inverse	Abbreviation
Precedes	**p**	Is preceded by	$\mathbf{p}^{-1}$
Meets	**m**	Is Met by	$\mathbf{m}^{-1}$
Overlaps	**o**	Is Overlapped by	$\mathbf{o}^{-1}$
Starts	**s**	Is started by	$\mathbf{s}^{-1}$
During	**d**	Contains	$\mathbf{d}^{-1}$
Finishes	**f**	Is Finished by	$\mathbf{f}^{-1}$
Equal	**e**	Equal	**e**

Table 1: Relationships and their abbreviations

However, ambiguities arise computing the relation when interval relationships do not have discrete values. For example, assume the assertions that interval *A precedes B*, (*A **p** B)*, and *B during C*, (*B **d** C)*. In computing the transitive relationship between A and C, the potential relationships between A and C could be any one of *precedes*, *meets*, *overlaps*, *starts*, and *during*. None of these violate the relationships asserted between A and B or B and C. This ambiguous transitive computation is illustrated in Figure 2.

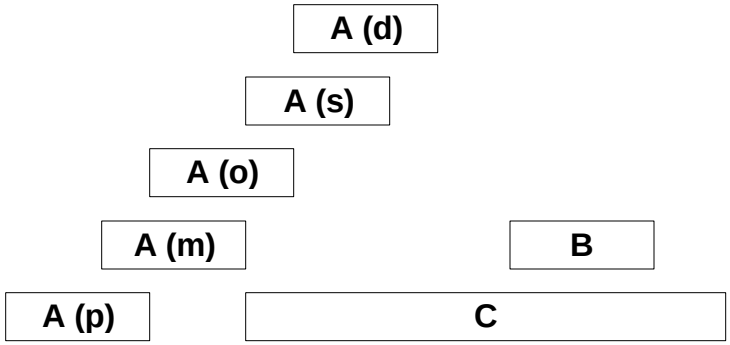

Figure 2. Ambiguous Transitive Computation

When describing complex interactions between entities involving multiple temporal intervals, ambiguous relationships occur frequently. Computing the transitive closure over a set of temporal intervals, becomes intractable for large sets of intervals with moderate ambiguity. For example, take the set of relations, A precedes B ***(A p B)***, B during C ***(B d C)***, and C contains[1] D ***(C d^{-1} D)***. The relationships asserted between **A** and **B,** and **B** and **C** yields the set of possible transitive computations between **A** and **C** previously shown in Figure 2. Computing the transitive relations between **A** and **D**, requires that each of the five possible computed relations between **A** and **C** be evaluated in the transitive computation. Thus, the transitive calculation is performed for each of the five possible relationships between A and C ***(A <p, m, o, s, d> C)*** is computed with the relationship between C and D ***(C d^{-1} D)***. The result is union of the computations obtained for each of the five relationships between A and C and the relationship between C and D. forms the set of potentially valid relationships between **A** and **D**. The algorithm must, in essence, follow divergent paths when it encounters an ambiguous condition.

[1] Contains is the inverse of during. Thus the use of the symbol di to denote the relationship (C di D) is used.

Applying Allen's algorithm for computing transitive closure over a set of temporal relations has previously been shown to be NP-hard [Vilain and Kautz, 1986]. The intractable nature of the full interval-based temporal algebra has resulted in several research directions focusing on alternative representations for temporal systems. Vilain [1982] developed a point-based algebra for temporal systems and [Dechter, Meiri, and Pearl, 1992] developed a representation utilizing directed graphs. These approaches reduced complexity by limiting the depth of the representation to achieve polynomial-time performance. Work by Alur and Henzinger [1994] employs a timed-logic concept that provides a representation tailored more towards real-time systems and event processing. Constraint-based systems, such as found in Dean [1987], form a directed graph in which the nodes represent states[2] and the arcs are labeled with temporal data which constrains the minimum and maximum temporal "cost" to traverse the link. All of the above efforts, however, employ a fundamentally different representation than the interval-based concept in order to achieve less complexity.

Alternatively, [Dorn, 1992] focused on a subset of the temporal algebra which maintains the interval as the basic representational unit but uses a directed graph to represent the collection of interval relationships within a given system. In this approach, the nodes of the graph represent the intervals and the arcs represent the relationships. This is in contrast to constraint-based approaches in which the nodes represent states (or points) and the arcs represent durations (i.e. intervals). The sequence graphs preserve the richness of the temporal algebra but rely on a graph reduction algorithm to minimize the number of arcs within a sequence graph. This algorithm is dependent on the width and length of the interval graph. The width is the maximum number of concurrent intervals and the length is the longest possible sequence chain. Problems arise because the width of a sequence graph is not easily determined and can limit the effectiveness of the representation for systems of highly parallel tasks.

---

[2]These are similar to the states in situational calculus.

2. Bit-encoded Temporal Representation

Our research focuses on finding a tractable means of computing the transitive closure over intervals while maintaining the richness of the temporal algebra. This section presents the bit-encoded temporal algebra that represents a solution to the problem. As mentioned earlier, the encoding is based on the unique signatures of each of the basic temporal relations. The signatures are assigned a binary value based on the set of end-point relationships for the interval relation. The algebra is first defined in discrete mathematical terms in the balance of this section and is then elaborated into a data structure and algorithm in succeeding sections.

When describing the temporal intervals defined by Allen, references are made to the start and end of the intervals in question. Allen and Hayes [1985] provide a stepwise development of the thirteen temporal relationships based on the single relationship "MEETS." They also present the concepts of "instantaneous" events that were called "Nests." These are described as the "beginning or ending" of intervals. The important aspect of nests is that they are of very small duration and, thus, take on the properties of points within the temporal calculus.

The essential quality of nests is that they have the properties of points and, thus, are totally ordered. Thus for any two nests N and M, either $N<M$, $M<N$, or $N=M$. These nests form an initial starting point for the development of the theory presented here.

Although Allen categorically states that time points are unnecessary in understanding temporal relationships [Allen, 1983a] the description of the interval relationships, nonetheless, refer to the start and end of an interval as a discrete point in time. The work presented in this paper is based on the relationships that exist between the start and end points of intervals. These relationships exist between bounding intervals that identify and bound the temporal interval.

Intervals can be ordered based on the sets of relationship values that are asserted between the start and end points of the intervals. The relationship between the start and end bounding intervals for each interval is implicit within the description. In order to facilitate any further

discussion, the following section defines some of the terminology and concepts to be used in this article.

2.1 Definitions

A temporal point, ***p***, has a scalar value representing the magnitude of the temporal value. This magnitude represents the displacement from some arbitrary starting point of the temporal system. The start of the temporal system is referenced as zero and all temporal points in the system have a positive magnitude with respect to the starting point of the temporal system.

Let ***P*** be the set of temporal points $\{p_1, p_2, p_3, \ldots, p_n\}$ where any point, p_i, in set ***P*** has a magnitude representing the time for that point. For any two temporal points, there may be six possible relationships. These are:

$p_i < p_j$,

$p_i <= p_j$,

$p_i = p_j$,

$p_i >= p_j$,

$p_i > p_j$, and

$p_i <=> p_j$

where <=> represents an unknown relation.

An interval ***i*** is represented as a pair of points (p_i, p_j) where p_i is the start of the interval and p_j is the end. Let an interval, ***i***, be represented as such an ordered pair of points, (p_i, p_j), where the relation, $p_i <= p_j$, holds. Let ***I***, then be the set of intervals formed from pairs of points in the set ***P***. Thus, $I = \{i_1, i_2, \ldots i_n\}$ where $i_1 = (p_i, p_j)_1$, $i_2 = (p_k, p_l)_2$, $\ldots$ $i_n = (p_x, p_y)_n$. Thus, an interval may be represented which has a duration of zero magnitude or greater.

Let an interval relationship, ***r***, consists of the set of the relationships existing between each of the points specified by the individual interval, ***i***. Thus, if two intervals, i_1 and i_2, are represented by the ordered points (p_1, p_2) and (p_3, p_4), respectively, then the interval relationship

$r(i_1, i_2)$ consists of the relationships represented by all possible relationship between each of the points, (p_1, p_2, p_3, p_4), which define the intervals.

Finally, let R contain the set of all interval relationships for a given temporal system. Thus $R = \{r_1, r_2, \ldots r_n\}$ where r_1 is the set of relationships between all points defined by the intervals that comprise the interval relationship, $r_1 = (i_n, i_m)$.

The relationship between two intervals may be represented using a directed graph. This form a node in the graph represents an end point of an interval. The arcs in the graph represent the ordering of the point according to the definitions above. The directed graph for the relationship A overlaps B is shown below in Figure 3.

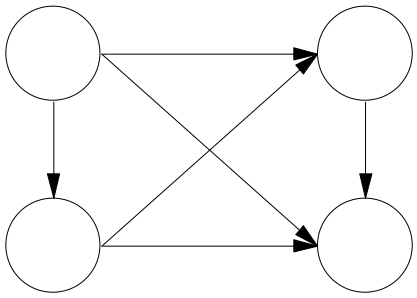

Figure 3: Representing an Interval Relationship as a Directed Graph

If we take the above graphical representation and add to it the graph for the relationship B overlaps C, the graph presented in Figure 4 is produced.

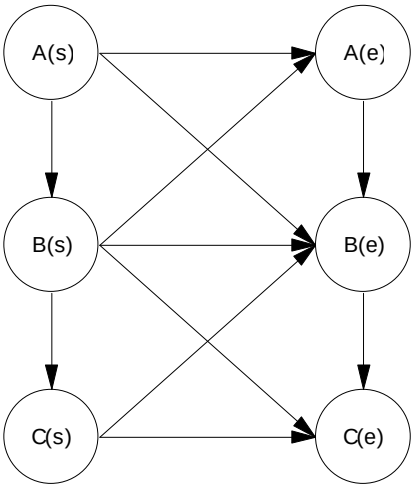

Figure 4: Directed Graph of A overlaps B and B overlaps C

Thus, the calculation of the transitive relationship between intervals A and C is the reduction of the directed graph in figure 4 to a graph containing only the nodes for intervals A and C. To perform this reduction, we must identify paths between each node of interval A and each node of interval C through each node of interval B or from each node of interval C to each node of interval A including each node of interval B. Stated differently, in order to derive the transitive relationship between A and C each possible path must be identified between A and C through B.

Upon inspection, it can be seen that for each of the end points of A and C there exists at least one path between each end point of one to each end point of the other with the exception of Ae and Cs. This confirms the fact that for the relationships A overlaps B and B overlaps C, the transitive relationship is ambiguous with respect to the relation between the end of interval A and the start of interval C.

2.2 Representing Interval Relations – The Relations Matrix

The relation "signature" between two intervals ***A*** and ***B*** can be represented by a matrix of relationships between of each of the interval end-points. Figure 4 displays the graphical relationship between intervals **A** and **B** when (***A o B***). As shown in Figure 5, the relationship of

the endpoints of the two for the relationship *overlaps* result in the overlaps interval signature matrix. This representation is similar to the matrix representation described by Sheppard and Simpson [1992] to represent end-point relations and perform basic transitive computations. However, where the axes of Shepard and Simpson's matrix includes all points for both intervals, the matrix specified herein only represents the essential relationship between the two intervals. Thus, while Shepard and Simpson's matrix contains all relationships between all points, it contains four sub-matrices. Two of these sub-matrices are the identity relationship for an interval with itself. The other two relation matrices are represent the relationship between the intervals and the inverse relationship. Since the identity relationship of an interval with itself is always constant, there is no value added in their inclusion in transitive calculations. Further, the inverse relationship may be easily computed for any given relationship. Thus, it is only necessary to maintain the 2x2 matrix representing the actual relationship between the intervals.

The next step is to compute the transitive relation between this interval relationship and (***B o C***), which is also depicted in Figure 4. That is, what is the relationship between intervals **A** and **C**? This is first performed using the graphical technique of mapping endpoints to time lines. As shown in Figure 5, however, a problem arises in that an ambiguity is raised for the endpoints A_e and C_s.

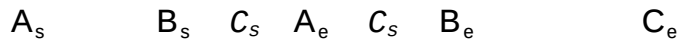

Figure 5. Graphical description of (A *overlaps* B) and (B *overlaps* C) with Endpoint Identification

		B	
A		**start**	**End**
	start	<	<
	end	>	<

		C	
B		**start**	**End**
	start	<	<
	end	>	<

Figure 5. Matrix Representation of (*A overlaps B*) and (B *overlaps* C) Relationship Signatures

The information expressed in the form of endpoint relationships is explicit enough to prove that both A_e and C_s are between B_s and B_e. However, there is not sufficient information to explicitly state the relationship that exists between A_e and C_s. Thus, there are three potential resolutions of the ambiguity, $A_e < C_s$, $A_e = C_s$, or $A_e > C_s$. In other words, any one of the three possible assignments of a relationship between A_e and C_s could be used without violating the primary relationships of (***A o B***) and (***B o C***).

Figure 6 shows the same relationship between **A** and **C** that is shown in Figure 5 between **A** and **B** and between **B** and **C**. Note, however that the relationship $\mathbf{A_e}$ and $\mathbf{C_s}$ is identified in the representation as ambiguous. This representation captures, in a concise fashion, the fact that interval A may *precede*, *meet*, or *overlap* interval C.

		C	
A		**start**	**End**
	start	<	<
	end	< = >	<

Figure 6. Matrix Representation of the Transitive Relation between A and C.

The implication of this statement is that time is unidirectional and that the completion or end point of some event X, which is represented by the interval, occurs some time after the start of X. Using these descriptions, we can then develop a set of bounding interval relationships. The bounding interval relationships for each of the thirteen temporal relationships specify a set of conditions that must be true in order for the relation to be valid. We can illustrate these

relationships by taking the matrix signature representing the relationship between two intervals and "unwrapping" it in a column-major fashion. This would then give us the relation between endpoints as follows: $\mathbf{A_s\text{-}B_s/A_e\text{-}B_s/A_s\text{-}B_e/A_e\text{-}B_e}$. These are illustrated in Table 2.

Table 2. Interval Relationship Signatures

Relation	Signature
Precedes (<)	<<<<
Meets (m)	<=<<
Overlaps (o)	<><<
Starts (s)	=><<
During (d)	>><<
Finishes (f)	>><=
Overlap Inverse (oi)	>><>
Meets Inverse (mi)	>>=>
Precedes Inverse (>)	>>>>
Starts Inverse (si)	=<<>
During Inverse (di)	<>>>
Finishes Inverse (fi)	<><=
Equal (=)	=><=

This notion of a unique signature for each relation is a key part of the approach for this research. It provides a method for assigning a unique code that identifies a specific *relation type*.

2.3 Mapping Relation Signatures to Bit Representations

Based on the assertion that there may be, at most, four possible relationships, *greater than*, *less than*, *equal*, and *unknown* asserted between any two intervals, these four relationships may be mapped to specific bit representations. For the purposes of our work, the <= and >= relationships can be said to be subsumed by the others, and thus unnecessary. These bit-encoded relations are summarized in Table 3.

Table 3. Bit Encoded Temporal Relationships

Relation	Bit Code Representation

Equal (=)	00
Less Than (<)	01
Greater Than (>)	10
Unknown (<=>)	11

Mapping these bit-encoded representations for each relationship now provides a distinct, 8-bit representation for each of the temporal intervals defined in Allen's Algebra. These are illustrated in Table 34.

These relation signatures can be presented using the relation matrix form used previously. Figure 7 illustrates the bit code representation used to represent an ***o*** relationship between two intervals, **A** and **B**, in matrix form.

Table 4. Bit Code Assignment of Interval Signatures

Relation	Signature	Bit Encoding
Precedes (<)	<<<<	01010101
Meets (m)	<=<<	01000101
Overlaps (o)	<><<	01100101
Starts (s)	=><<	00100101
During (d)	>><<	10100101
Finishes (f)	>><=	10100100
Overlap Inverse (oi)	>><>	10100110
Meets Inverse (mi)	>>=>	10100010
Precedes Inverse (>)	>>>>	10101010
Starts Inverse (si)	=<<>	00010110
During Inverse (di)	<>>>	01101010
Finishes Inverse (fi)	<><=	01100100
Equal (=)	=><=	00100100

B

A		**start**	**end**
	start	01	01
	end	10	01

Figure 7. Matrix Representation of A overlaps B Using Bit Codes

2.4 Temporal System Matrix Representation

The relation matrix adequately describes a relation between two intervals. However, it is also necessary to represent in a similar fashion the global relations between all the intervals in the temporal system. This is done through the temporal system matrix.

In this matrix, both axes contain each of the temporal intervals that compose the temporal system. The diagonal, of course, represents the relationship of each interval to itself, and is therefore, insignificant. However, all other elements of the matrix represent the relationships between intervals. The relations represented in the upper triangular part of the temporal system matrix represent the relation between the vertical and the horizontal axes, in that order. The relations represented in the lower triangular part of the matrix, on the other hand, represents the relations between the horizontal and vertical axes, or in other words, the inverse relation of that shown in the corresponding location in the upper triangular matrix. How these inverses are computed is described in detail in section 3.3 below. This temporal system matrix can thus represent all the relations explicitly defined as well as all computed transitive relations.

Each element of the matrix consists of a string of four decimal digits between 0 and 3, where 0 through 3 correspond to the underlying bit encoding of the relationship between the end points. Thus, 0 ≡ 00, 1 ≡ 01, 2 ≡ 10, and 3 ≡ 11. An example of a temporal system where the transitive closure between intervals has already been calculated is shown as figure 8.

```
     00    01    02    03    04    05    06    07    08    09    10    11
00 (----)(1121)(1121)(1111)(1111)(3131)(3333)(1111)(1111)(2120)(3131)(3131)
01 (2122)(----)(3123)(1133)(1133)(3133)(3333)(1133)(1133)(2122)(3133)(3133)
02 (2122)(3123)(----)(1101)(1111)(3121)(3323)(1111)(1111)(2122)(3131)(3131)
03 (2222)(2323)(2022)(----)(1133)(2121)(2323)(1133)(1133)(2222)(3133)(3133)
04 (2222)(2323)(2222)(2323)(----)(2323)(2323)(1101)(1111)(2222)(3121)(3121)
05 (3322)(3323)(3122)(1122)(1133)(----)(2122)(1133)(1133)(3322)(3133)(3133)
06 (3333)(3333)(3133)(1133)(1133)(1121)(----)(1133)(1133)(3333)(3133)(3133)
07 (2222)(2323)(2222)(2323)(2022)(2323)(2323)(----)(1120)(2222)(2121)(2121)
08 (2222)(2323)(2222)(2323)(2222)(2323)(2323)(2120)(----)(2222)(2121)(2121)
09 (1120)(1121)(1121)(1111)(1111)(3131)(3333)(1111)(1111)(----)(3131)(3131)
10 (3322)(3323)(3322)(3323)(3122)(3323)(3323)(1122)(1122)(3322)(----)(0121)
11 (3322)(3323)(3322)(3323)(3122)(3323)(3323)(1122)(1122)(3322)(0122)(----)
```

Figure 8. Example of temporal system matrix consisting of 12 intervals

Thus, in Figure 8, the entry at row 00, column 01 represents the relationship between interval 00 and 01. The value of 1121 in this entry represents the end point relationship matrix in row-major order. Thus, 1121 corresponds to a bit signature of 01011001 represented in matrix form below.

		01	
		start	**end**
00	**start**	01	01
	end	10	01

Figure 9: Matrix Representation of Overlaps Relationship Between Intervals 00 and 01

The signature for this matrix is 01100101 or (<><<) which is the signature for the overlaps relationship. So, therefore, the above entry states that interval 00 overlaps interval 01.

2.5 Summary

The above section describes how to represent temporal events as bit-encoded strings. Calculation of the closure is performed by logical bit operations on the signature relations between the two relation-interval pairs over which the closure is desired. Using the example of (**A *o* B**) and (**B *o* C**), the computation of the transitive closure (**A,C**) is described in the following section.

3. Transitive Relation Computation

Computing the transitive relationship between any two relationships can now be calculated by defining the Sum and Product operations within the domain of the bit-encoded forms and applying the procedure of matrix multiplication.

3.1 Binary Matrix Operations

There are two basic operations that can be performed in the interval algebra. These are the sum (⊕) and product (⊗) operations. The sum operation in the algebra is defined as a binary inclusive *and* of the two bit codes. The product is defined to be the binary *or* of the two bit codes representing the relations. The sum and product operators shown are also consistent with symbolic versions of relation operators defined by Sheppard and Simpson [1992].

3.1.1 Product Operation. The relation product is shown in Table 12 below. The product operation is a binary *or* over the end point relation codes. A logical *or* is applied because the product of two terms in the binary algebra represents the combination of all possible alternatives of the two terms. The *or* operation provides this logical combination.

Table 5. Product Table for the Temporal Algebra

⊗	**00**	**01**	**10**	**11**
00	00	01	10	11
01	01	01	11	11
10	10	11	10	11
11	11	11	11	11

3.1.2 Sum Operation. As previously mentioned, the sum of two terms in the algebra is formed by taking the logical *and* of the terms. The binary sum of the two terms represents the reduction of a set of possible relationships to the relationships that are consistent between the two terms. Thus, the *and* operation is used to perform this calculation.

Table 6. Addition Table for Temporal Algebra

⊕	**00**	**01**	**10**	**11**
00	00	00	00	00
01	00	01	00	01
10	00	00	10	10
11	00	01	10	11

Both the sum and product operators in this algebra are commutative.

The next section presents the transitive closure operation and how it will be computed using the representation described above.

3.2 Transitive Computation

The key, limiting factor in all temporal reasoning systems is the ability to efficiently compute full transitive closure over a system of intervals. This process ensures consistency across all temporal relations within the system and asserts transitively computed relations for those that have no primitive relation asserted. The transitive computation takes two relations, *R1[]* and *R2[]*, which are relations between three intervals, ***A***, ***B***, and ***C*** such that (***A*** *R1[]* ***B)*** and (***B*** *R2[]* ***C),*** and where the relationships and intervals are represented as matrices such as found in figures 7 and 9 above . Computation of the transitive relation, (***A*** *Result[]* ***C),*** is performed as a simple matrix product applying the sum and product operations defined earlier in this chapter. The following algorithm describes this procedure.

In the algorithm below R1 and R2 represent a relationship between two intervals for which a transitive computation will be performed. Thus, the second interval of R1 is the first interval of R2. In the function specification relation names are shown. However, the computation references the entries in the relation matrix.

```
ComputeTransitive(R1(A,B)[],R2(B,C)[])
begin
      Result(A,C)[] = 0
      Result[1,1] ¬ (R1[1,1] Ú R2[1,1]) Ù (R1[1,2] Ú R2[2,1])
      Result[1,2] ¬ (R1[1,1] Ú R2[1,2]) Ù (R1[1,2] Ú R2[2,2])
      Result[2,1] ¬ (R1[2,1] Ú R2[1,1]) Ù (R1[2,2] Ú R2[2,1])
      Result[2,2] ¬ (R1[2,1] Ú R2[1,2]) Ù (R1[1,2] Ú R2[2,2])
      If isAmbiguous(Result(A,C)[]) then
            AmbiguousRelations ¬ Result(A,C)[]
      return(Result(A,C)[])
end;
```

Transitive products may result in an ambiguity. These ambiguous relations are the key transitive computation process since they are the only relations which may be changed through the transitive computation without invalidating the relationships within the matrix. Thus, transitive computations are tested for ambiguity and, if found to be ambiguous, are stored in the set *AmbiguousRelations*. This set is referred to during the transitive closure computation to identify other relationships that may be recomputed during the process.

The result of computing the transitive closure of intervals A and C from the relations described in Figures 7 and 9 is represented as a matrix of the form found in Figure 10.

C

A		**start**	**end**
	start	01	01
	end	11	01

Figure 10. Matrix Representation of (A overlaps C) Using Bit Codes

Applying the above algorithm, the combination of relationships R1 and R2 (Figs. 7 and 9) can be accomplished by following the standard ordering for matrix multiplication between two matrices of equal size. This is illustrated in Figure 11. Row 1 of the first relationship matrix, consisting of the relationships between the start point of the first relationship and the start and end points of the second relationship, is matrix multiplied with the left column of the second relationship matrix.

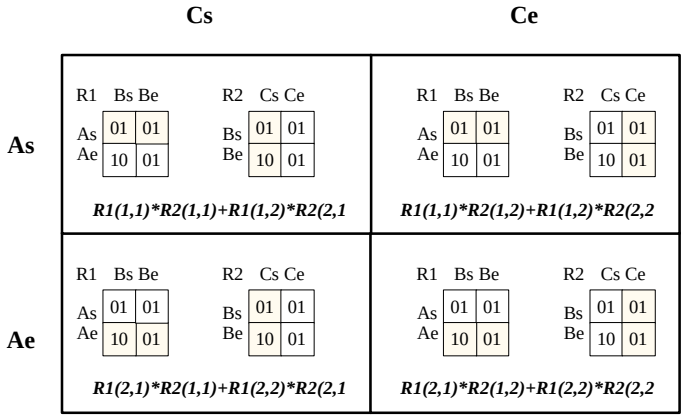

Figure 11. Matrix operation ordering for transitive computation

Thus, the transitive computation is performed following the standard procedure for matrix multiplication. This result of the matrix product operation is shown below.

$$\begin{bmatrix} (R1[1,1] \otimes R2[1,1] \oplus R1[1,2] \otimes R2[2,1]) & (R1[1,1] \otimes R2[1,2] \oplus R1[1,2] \otimes R2[2,2]) \\ (R1[2,1] \otimes R2[1,1] \oplus R1[2,2] \otimes R2[2,1]) & (R1[2,1] \otimes R2[1,2] \oplus R1[2,2] \otimes R2[2,2]) \end{bmatrix}$$

Figure 10 shows the result of the bit-calculation for the two relations. The cell corresponding to the relationship between the end of interval **A** and the start of interval **C** shows 11 as the value. This is the value resulting from the transitive computation algorithm and represents and unknown or ambiguous relationship between the endpoints.

The problems associated with the computation of transitive closure over a set of intervals fall into two primary areas, consistency and performance. In the former, the issue is to identify as early as possible in the closure process any conflicts that will arise due to primitive relation assignments. The latter is concerned with the performance of the closure algorithm in terms of computing a complete and average set of closures given a set of intervals and primitive relationships.

3.3 Inverse Relation Computation and Matrix

The inverse, R^{-1}, of any relationship may be computed in a simple and straightforward manner. This holds for all the relationships within the temporal algebra. The inverse of a relationship is isomorphic with the definition of the inverse relationships identified by Allen. The interval matrix is divided along the diagonal by a set of equal relations corresponding to the identity operation (i.e. A = A). The two triangular sub-matrices are symmetrical along this axis with the lower triangular matrix being an inverse of the values in the upper triangular matrix (i.e. $R(i,j) = R^{-1}(j,i)$) where i = the row and j = the column of the interval relation.

The inverse operation can be thought of as viewing the relationship via a timeline vector 180 degrees out of phase. This changes the direction of the paths between the end points of the two intervals. This has the effect of inverting the relationship (i.e. $>$ *and* $<$ $<$ *and* $>$) with the exception of the equality (=) and unknown (<=>) endpoint relations. Furthermore, the change in viewing direction or perspective is captured within the individual relationship matrix representation by inverting the relationships and then transposing the matrix about the diagonal.

```
ComputeInverse(R[])
begin
     Result[] = 0;
     If R[1,1] = 01 or 10 then
          Result[1,1] ← 11 ∇ R[1,1]
          else
          Result[1,1] ← R[1,1];
```

```
        If R[2,1] = 01 or 10 then
             Result[1,2] ← 11 ∇ R[2,1]
             else
             Result[1,2] ← R[2,1];
        If R[1,2] = 01 or 10 then
             Result[2,1] ← 11 ∇ R[1,2]
             else
             Result[2,1] ← R[1,2];
        If R[2,2] = 01 or 10 then
             Result[2,2] ← 11 ∇ R[2,2]
             else
             Result[2,2] ← R[2,2];
        return(Result[]);
end;
```

The ComputeInverse operation performs this inversion of the operators and transposition of the matrix. The inversion is performed by taking the exclusive-or, XOR, (shown as the symbol $\tilde{N}$) of the end point relationship if the relationship is either < or > (i.e. 01 or 10).

3.4 The Bit-mapped Closure Algorithm

As previously noted, Allen's constraint propagation algorithm has been shown in [Vilain and Kautz, 1986] to perform on average $O(n^3)$. This is the same order of complexity as Warshall's algorithm for transitive closure, also shown in [Vilain and Kautz, 1986].

The following section presents a representation of temporal relations based on a bit-encoded representation of interval end points or "nests" referred to by [Allen, 1985]. This representation presents a terse coding of the temporal relations. The algebra defined in [Kovarik 1994] provides for the addition of the ambiguous relation, (<=>), not defined in Allen's description. This enables the temporal system to represent ambiguous relationships between intervals

This representation forms the basis for computing transitive relations based solely on the interval algebra operations defined earlier. This allows for a straightforward algorithm

computation of transitive relations without any table lookup. Finally, the complexity of the transitive closure algorithm developed is dependent on the number of relationships within the matrix that are ambiguous and not upon the total number of intervals.

The bit-encoded transitive closure algorithm is described in this section. The foundation of the algorithm is the computation of the transitive relation over three intervals described in section 3.2 above. This computation is based on the graphical foundation presented in the previous sections. This operation is illustrated below.

The algorithm maintains the relationships in a temporal system matrix. As discussed previously, the lower triangular matrix entries in a temporal system matrix are the inverse of the upper triangular matrix entries. Therefore, the lower entries are not calculated as part of the algorithm. Instead, the algorithm limits the computations performed to the upper diagonal matrix and the lower diagonal entries are simply calculated using an inverse operation as described in section 3.3 above.

The ComputeTransitive operation refers to an IsAmbiguous operation that tests whether a relationship is ambiguous, and returns true if it is or false if not. This operation is shown below.

```
IsAmbiguous(R[])
begin
     If
          R[1,1] = 11 or
          R[1,2] = 11 or
          R[2,1] = 11 or
          R[2,2] = 11
     then
          return(true)
     else
          return(false);
end;
```

Simply stated, a relation is ambiguous if any endpoint relationship is unknown. Thus, the above operation checks for the existence of the ambiguous code, 11, in any of the four entries and returns true or false accordingly.

When adding an interval to the system, at least one relationship must be specified with an interval previously in the system in order to calculate the corresponding relationships for the new relationship and the others. This is performed by computing the transitive relationship for each entry in the column corresponding to the new interval. This operation is shown below.

```
ComputeColumn(i,j)
begin
For k ← (i - 1) to 0 step -1 do
     begin
     Table[k,j] ← ComputeTransitive(Table[k,i],Table[i,j]);
     Table[j,k] ← ComputeInverse(Table[k.j]);
     end;
For k ← (i + 1) to j step 1 do
     begin
     Table[k,j] ← ComputeTransitive(Table[k,i],Table[i,j]);
     Table[j,k] ← ComputeInverse(Table[k.j]);
     end;
end;
```

Again, since the lower diagonal matrix is simply the inverse of the upper diagonal, only the column entries need be calculated. The corresponding row in the lower diagonal matrix is populated by computing the inverse for each column entry computed. Thus, for a new interval, a relation is asserted and then the entries above the asserted relation in the column are computed and then the entries below the entry are computed.

Relations are asserted into the temporal system matrix through the *Add* operation. This operation asserts a relation *R[]* for an interval pair <*i,j*> within the matrix. The *Add* operation is shown below.

```
Add(R[]<i,j>)
begin
If Table[i,j] is null then
     begin
          Table[i,j] ← R[]<i,j>;
          Table[j,i] ← ComputeInverse(R[]);
          ComputeColumn(i,j);
          return;
     end;
If isAmbiguous(Table[i,j]) then
     If isCompatible(R[],Table[i,j]) then
          begin
               Table[i,j] ← R[]<i,j>;
               Table[j,i] ← ComputeInverse(R[]);
               Remove <i.j> from AmbiguousRelations;
               ChangedRelations ← <i,j>;
               ComputeClosure();
               return;
          end;
     else
          return(Exception("Relation is not compatible"));
else
     return(Exception("Relation already defined"));
end;
```

There are only three possible states that can occur. Either,

1. the relation is being asserted for a new relation and, therefore, the entry in the matrix is null,

2. the relation is being asserted over an existing relation that is ambiguous, or

3. the relation is being asserted for a location in the matrix which already contains a non-ambiguous relationship.

In the first case, the relationship can be simply asserted within the matrix, the inverse of the asserted relationship is calculated and inserted in the matrix, and the remainder of the new interval's relationships can be computed via the *ComputeColumn* operation.

In the third case, a relationship is specified for an interval pair which already has a non-ambiguous relation asserted. Thus, for any relationship that is not equal to the existing relationship in the table, an error is raised.

In the second case, the matrix entry contains an ambiguous relation and a new, potentially more specific relationship is being asserted. The *IsCompatible* operation is called to check if the relationship to be asserted is compatible with the existing relationship. If so, the newly refined relationship is removed from the set of ambiguous relationships, if it is non-ambiguous, and it is inserted into the *ChangedRelations* queue. If the relationship asserted is not compatible with the existing relationship then a *"Relation is not Compatible"* exception is raised.

```
isCompatible(RAssert[],RExist[])
begin
If
     (RAssert[1,1] = RExist[1,1]
          or RExist[1,1] = 11
          or Rassert[1,1] = 11)
     and
     (RAssert[1,2] = RExist[1,2]
          or RExist[1,2] = 11
          or Rassert[1,2] = 11)
     and
     (RAssert[2,1] = RExist[2,1]
          or RExist[2,1] = 11
          or Rassert[2,1] = 11)
     and
     (RAssert[2,2] = RExist[2,2]
          or RExist[2,2] = 11
          or Rassert[2,2] = 11)
     then
          return(true)
     else
          return(false);
end;
```

If the relation being asserted replaces a previously ambiguous relationship then, in addition to being inserted into the *ChangedRelations* queue, the *ComputeClosure* function is called.

The ComputeClosure function removes the first entry from the ChangedRelations queue and checks it against the remaining ambiguous relations. If it can be used to calculate the transitive relation for an ambiguous relationship, then the new transitive relation is computed and asserted into the matrix.

```
ComputeClosure()
begin
     while ChangedRelations is ¬null do
     begin
     <i,j> ← first element of ChangedRelations;
     For (each pair <k,l> in
          (row i | column i | row j | column j)
          where <k,l> is in the upper triangular matrix
          and <k,l> is ambiguous)
     do
     begin
          If i=k then
          begin
               Table[k,l] ← ComputeTransitive(i,j,l);
               Table[l,k] ← ComputeInverse(R[k,l]);
               ChangedRelations ← <k,l>;
          end;
          If j=l then
          begin
               Table[k,l] ← ComputeTransitive(k,i,j);
               Table[l,k] ← ComputeInverse(R[k,l]);
               ChangedRelations ← <k,l>;
          end;
          If i=l then
          begin
               Table[k,l] ← ComputeTransitive(k,j,i);
               Table[l,k] ← ComputeInverse(R[k,l]);
               ChangedRelations ← <k,l>;
               Remove <k,l> from AmbiguousRelations;
          end;
          If j=k then
          begin
               Table[k,l] ← ComputeTransitive(j,i,l);
               Table[l,k] ← ComputeInverse(R[k,l]);
```

```
                ChangedRelations ← <k,l>;
                Remove <k,l> from AmbiguousRelations;
            end;
        end;
        end;
    end;
```

As shown above, the **ComputeClosure** algorithm forms the heart of the transitive closure computation for this algebra. The function is initiated whenever a relation is asserted into the matrix which refines a previously ambiguous relation in the matrix. The relation is added to the matrix, the inverse is also inserted into the matrix and the **ComputeClosure** routine is initiated.

The basic approach of this routine uses the *ChangedRelations* variable as an input queue to the closure computation process. The first interval pair is removed from the *ChangedRelations* and is checked against each of the interval pairs in *AmbiguousRelations*. If either of the intervals in the *ChangedRelation* matches one of the intervals of the *AmbiguousRelation* being inspected then the transitive relation for the *AmbiguousRelation* is recomputed using the *ChangedRelation*. If the newly computed relationship is not ambiguous then it is queued to *ChangedRelations* and removed from *AmbiguousRelations*.

The next section performs an analysis of an implementation of the closure algorithm and identifies worst case performance. It also presents an implementation of the algorithm and discusses the realization of the algorithm in code.

4. Implementation and Evaluation of Bit-mapped Transitive Closure Computation Technique

It is important that the technique presented here be evaluated for its effectiveness. The evaluation described consists of two main parts: 1) a theoretical complexity analysis of the worst and the average cases, and 2) an average case empirical test. The latter is composed of several different types of tests. A number of significant data were collected. These data covered the different aspects of the algebra, its implementation, and performance. Since the algorithm is

critically dependent on the number of ambiguous relations in the system, a test routine was developed which generated random interval assignments to provide some insight into the number of occurrences of ambiguous relationships in a totally random system.

Next, timing data was gathered to verify the claims of the algorithm's performance. This data was gathered in two forms. The first was an iteration count for the transitive closure computation. This instrumentation counted the number of cycles executed within the compute closure function. The second was a collection of timing data for a set of test cases. This provided two insights into the algorithms and its implementation. First, the cycle iteration provided a validation of the complexity analysis of the algorithm. Second, the timing data provided both, a verification of the algorithm's performance curve with respect to the number of ambiguous relationships in the system, and provided a cursory indication of the speed of the bit-mapped transitive relation computation.

4.1 Analysis of Worst-case Complexity

The transitive closure algorithm, ComputeClosure, presented earlier is composed of two iterative loops. The inner For loop iterates over each relationship in the row and column of the changed relation. The outer While loop iterates over a queue of changed relations until the queue is empty. The inner loop of the algorithm iterates over the row and column of each interval in the changed relation queue rather than the entire set of interval pairs as in Allen's closure algorithm. This yields, at worst, 2n computations for each ambiguous relationship. The outer loop consists continues until there are no further ambiguous relationships in the queue. The worst case is a totally ambiguous matrix where the number of ambiguous relations is equal to the number of

entries in the matrix[3] or n^2 where n = the number of intervals in the system. Thus, the order of magnitude of the algorithm's complexity is $n^2 * 2n$, or $O(n^3)$.

The *changedRelations* queue is initialized to the value of the interval pair for which a new, unambiguous relation is asserted. This value is dequeued and then each of the relations in the $<i,j>$ columns of the relation disambiguated is inspected to see if any of the relations can be recomputed based on the new relation. Thus, the worst case for performance is where a relation is asserted, the set of ambiguous relations is inspected and no additional relationships are disambiguated. This results in $4n$ comparisons for each relation in the *ambiguousRelations*.

It should be noted that as an unambiguous relation is asserted over an existing relation, the relation is removed from the *AmbiguousRelations* set and inserted into the *ChangedRelations* queue. The number of maximum comparisons performed by the closure algorithm is $4n*n(n-1)/2$ where n is the number of intervals in the system. Thus, the worst case performance of the algorithm is $O(n^3)$.

4.2 Empirical Validation

Four different tests were performed to provide empirical data to validate the algorithm and its performance. The first test provides an opportunity to observe how the interactive construction of a temporal system works using the presented technique. The objective of the second test was to establish a relationship between the number of intervals and the number of resulting ambiguous relations when the transitive closure was computed. The third test tries to determine the effect of scaling the algorithm to a large number of intervals to determine its scalability. Test #4 uses timing data from the other tests to provide an empirical indication of run

[3]Actually the maximum number of entries will be n^2-n because the diagonal of the matrix will always be the identity (=) relation and, therefore. unambiguous.

time and cycles. This is done as a way to verify the theoretical complexity of the algorithm derived in section 4.1 above.

4.2.1 Test 1: Interactive Construction of Temporal System

This test explored the use of the temporal reasoning system through an interactive construction. The goal was to provide a detailed trace of actions and results through the stepwise insertion of new intervals and relations and the refinement of ambiguous relations. A twelve interval temporal system was constructed and then refined.

The intervals (0 through 11) were created and then relationships asserted between them. The interval pairs and the relationships asserted are shown in Table 6.

Table 7. Table of Bit-Encoded Relationships

Interval Pair and Relation	Bit Code Asserted	Number of Ambiguous Relations
0 overlaps 1	(1 1 2 1)	0
0 overlaps 2	(1 1 2 1)	1
2 meets 3	(1 1 0 1)	2
2 precedes 4	(1 1 1 1)	4
3 during 5	(2 1 2 1)	8
5 overlaps inverse 6	(2 1 2 2)	13
4 meets 7	(1 1 01)	17
7 finishes inverse 8	(1 1 2 0)	21
0 finishes 9	(2 1 2 0)	23
7 during 10	(2 1 2 1)	31
10 starts 11	(0 1 2 1)	39

As can be seen, the number of ambiguous relationships for this example is slightly larger than $n^2/2$ [4]. The full matrix including all transitive computation is shown in figure 8, and shown below again as figure 12 for convenience. The next step is to assert an unambiguous relationship over one of the ambiguous relations in the matrix. The first such assertion is the during relationship (2 1 2 1) over the interval pair (2, 5). The existing relation computed for (2, 5) is (3

```
     00    01    02    03    04    05    06    07    08    09    10    11
00 (----)(1121)(1121)(1111)(1111)(3131)(3333)(1111)(1111)(2120)(3131)(3131)
01 (2122)(----)(3123)(1133)(1133)(3133)(3333)(1133)(1133)(2122)(3133)(3133)
02 (2122)(3123)(----)(1101)(1111)(3121)(3323)(1111)(1111)(2122)(3131)(3131)
03 (2222)(2323)(2022)(----)(1133)(2121)(2323)(1133)(1133)(2222)(3133)(3133)
04 (2222)(2323)(2222)(2323)(----)(2323)(2323)(1101)(1111)(2222)(3121)(3121)
05 (3322)(3323)(3122)(1122)(1133)(----)(2122)(1133)(1133)(3322)(3133)(3133)
06 (3333)(3333)(3133)(1133)(1133)(1121)(----)(1133)(1133)(3333)(3133)(3133)
07 (2222)(2323)(2222)(2323)(2022)(2323)(2323)(----)(1120)(2222)(2121)(2121)
08 (2222)(2323)(2222)(2323)(2222)(2323)(2323)(2120)(----)(2222)(2121)(2121)
09 (1120)(1121)(1121)(1111)(1111)(3131)(3333)(1111)(1111)(----)(3131)(3131)
10 (3322)(3323)(3322)(3323)(3122)(3323)(3323)(1122)(1122)(3322)(----)(0121)
11 (3322)(3323)(3322)(3323)(3122)(3323)(3323)(1122)(1122)(3322)(0122)(----)
```

1 2 1) which corresponds to the relation set (d s o). Thus, the relation asserted is compatible with the existing ambiguous relation.

Figure 12. Temporal system matrix consisting of 12 intervals

After the relation is asserted, the transitive closure algorithm is initiated. The number of iterations performed by this function is 38 or, the number of ambiguous intervals remaining after (2, 5) is removed. In this case, no additional ambiguities are resolved and there are 38 ambiguous intervals remaining after the transitive closure calculation.

The next relation asserted is the finishes relation over the interval pair (4, 6). In this case, when the transitive computation completes it has refined four additional relations yielding a total of 33 remaining ambiguous relations. The algorithm performed 37 iterations for the initial

---

[4]Note that the actual total number of ambiguous relations is 78. Since the matrix is inversely reflective about the diagonal, it is sufficient to maintain the set of ambiguous intervals in the upper diagonal.

asserted and then performed 33 iterations for each of the additional four relations. Thus, the original assertion resulted in the disambiguation of four additional relations but none of these produced further refinements. The total number of cycles for this assertion was 4 * 33 + 37 or 159. The resultant matrix is shown here as figure 13.

```
00    01    02    03    04    05    06    07    08    09    10    11
 00 (----)(1121)(1121)(1111)(1111)(3121)(3121)(1111)(1111)(2120)(3131)(3131)
 01 (2122)(----)(3123)(1133)(1133)(3123)(3123)(1133)(1133)(2122)(3133)(3133)
 02 (2122)(3123)(----)(1101)(1111)(2121)(2121)(1111)(1111)(2122)(3131)(3131)
 03 (2222)(2323)(2022)(----)(1133)(2121)(2123)(1133)(1133)(2222)(3133)(3133)
 04 (2222)(2323)(2222)(2323)(----)(2121)(2120)(1101)(1111)(2222)(3121)(3121)
 05 (3122)(3123)(1122)(1122)(1122)(----)(2122)(1123)(1133)(3122)(3123)(3123)
 06 (3122)(3123)(1122)(1123)(1120)(1121)(----)(1101)(1111)(3122)(3121)(3121)
 07 (2222)(2323)(2222)(2323)(2022)(2123)(2022)(----)(1120)(2222)(2121)(2121)
 08 (2222)(2323)(2222)(2323)(2222)(2323)(2222)(2120)(----)(2222)(2121)(2121)
 09 (1120)(1121)(1121)(1111)(1111)(3121)(3121)(1111)(1111)(----)(3131)(3131)
 10 (3322)(3323)(3322)(3323)(3122)(3123)(3122)(1122)(1122)(3322)(----)(0121)
 11 (3322)(3323)(3322)(3323)(3122)(3123)(3122)(1122)(1122)(3322)(0122)(----)
```

Figure 13. Resultant Temporal system matrix for Test #1.

In the first example, the 38 iterations took 36.75 seconds to complete. In the second example, which refined an additional four relations, the closure computation took 163.46 seconds to perform 159 iterations. The rough performance, in terms of time is one second per interval check. This appears to be true throughout the remainder of the twelve-interval example. Thus, while the number of ambiguous relationships may be a square of the number of intervals, the number of iterations and transitive closure execution time is linearly dependent on the number of ambiguous relationships. The balance of the twelve-interval system is provided in [Kovarik, 1994].

4.2.2 Test #2 - Distribution of Ambiguous relations

Since the performance of the algorithm is dependent on the number of ambiguous relationships in the system, a test was developed which selected a random number of intervals between two and twenty as the set of intervals in the system, and then generated random relation assignments between successive pairs in the system. As each pair had an interval relation randomly selected, the system computed the rest of the entries for the interval using the compute-

column function. A test generator was used to develop a set of temporal systems consisting of a random number of intervals between 2 and 25. One temporal relation for each column in the temporal system matrix was selected at random and the resultant closure was calculated.

The number of resulting ambiguous intervals was then identified, and associated with the number of intervals in that particular execution. This test was run 500 times to obtain an indication of the relationship between the number of intervals in the temporal system and the resultant ambiguous intervals.

Thus, a completely populated temporal system matrix of relationships was calculated for the intervals in the system. These data were then plotted on a graph to provide a visual presentation of the distribution of the ambiguous intervals. A raw data plot of the number of intervals in the system versus the number of ambiguous relationships is shown in Figure 14.

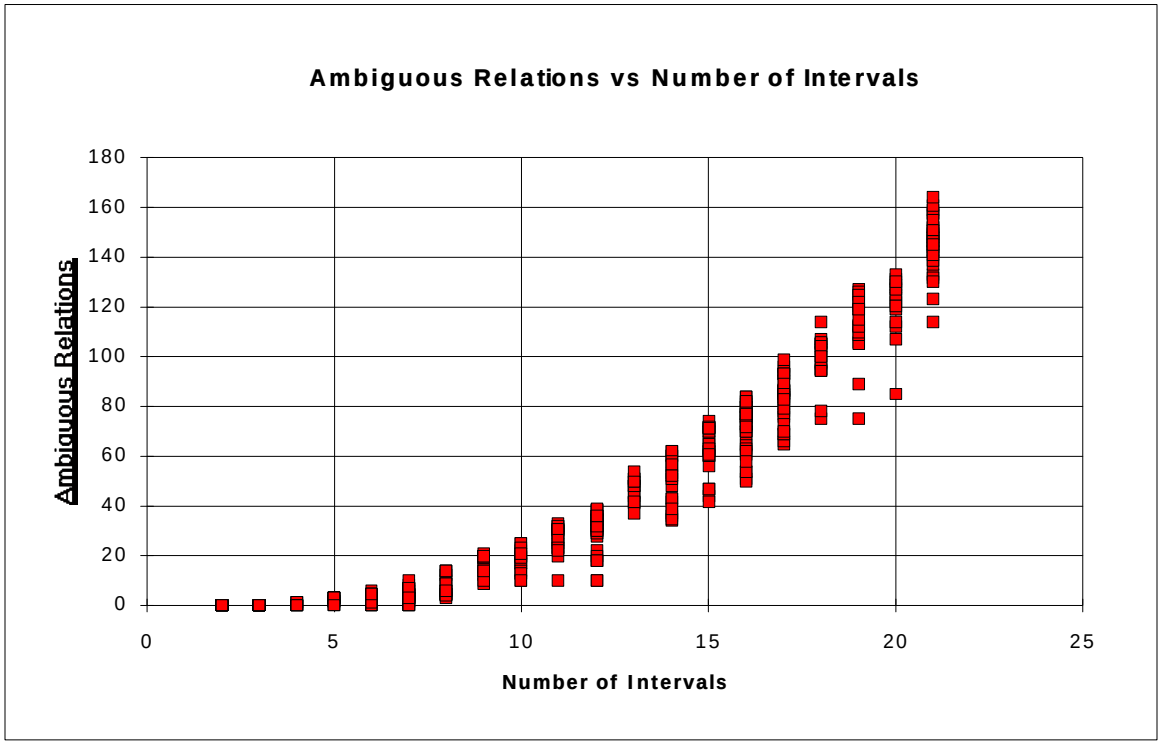

Figure 14: Number of Intervals versus Ambiguous Relations

As the plot shows, the number of ambiguous intervals within the system appears to be polynomially dependent upon the total number of intervals in the system. The range of the dependency is less than n^2, where n is the number of intervals in the system. This is because n^2 is the worst case where there are no unambiguous relationships within the system.

These intervals and ambiguous relationships were also plotted on a logarithmic scale. This is shown in Figure 15. Based on the shape of the plot, it can be seen that the number of ambiguous relationships is roughly in the order of n^2, but the placement of the curve indicates that it is less than a full n^2 by some relative multiplier.

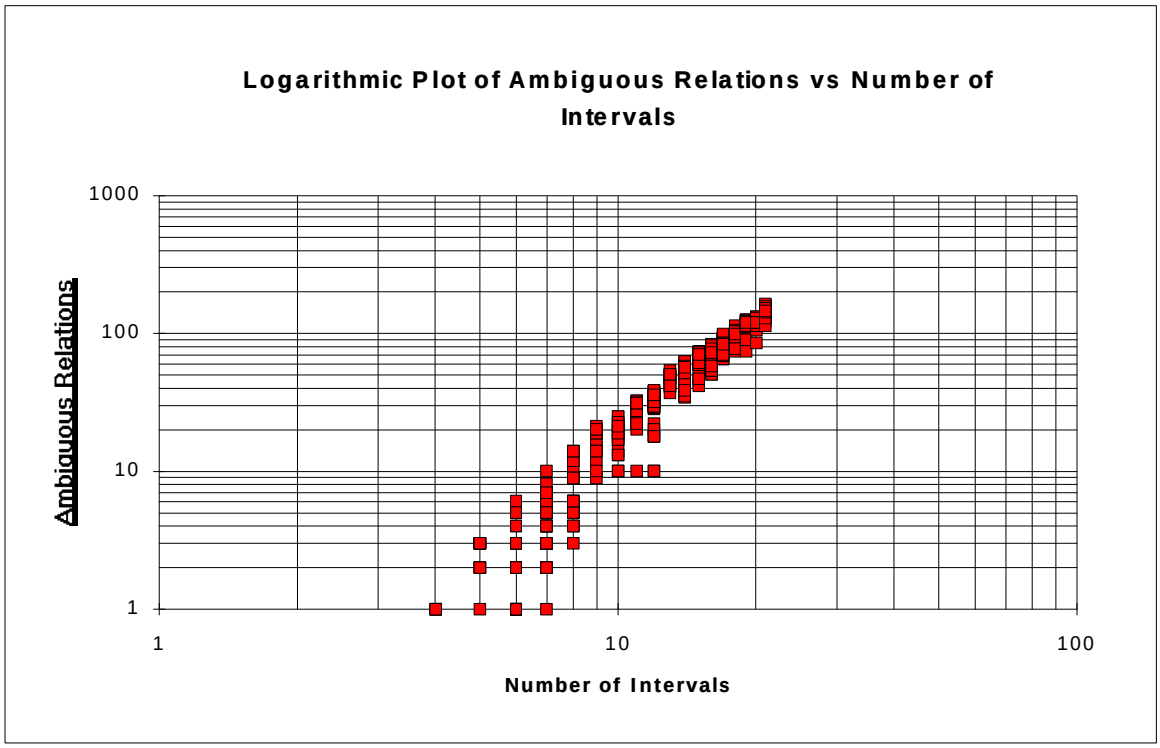

Figure 15: Logarithmic Plot of Number of Intervals versus Number of Ambiguous Relations

As the above plots show, the number of ambiguous relationship within the system is related to the number of total intervals in the system. Furthermore, as a general limit, the number

of ambiguous relationships would appear to be n^2. However, as the plot in Figure 15 indicates, the common limit appears to be somewhat less than half the square of the number of intervals,. $n^2/2$. Furthermore, as relationships are refined, the number of ambiguous relationships decrease. Thus, the $O(n^2)$ of the general limit only applies for initial matrices. Subsequent computations and refinements of the relationships within the matrix reduce the number of ambiguous relationships, thereby reducing the complexity of the transitive closure computation.

4.2.3 Test #3- Scalability and Large Interval sets

This set of tests addressed the issue of scalability. Would the number of ambiguous relationships be different for large temporal systems, or would the number be consistent with the data gathered for the smaller examples?. A secondary issue was also investigated through this set of tests. Specifically, when a set of unambiguous relations is asserted over an ambiguous relation in the matrix, triggering the transitive closure computation, what was the number of other ambiguous relations clarified during the computation?

To accomplish this, a set of five interval sets was generated consisting of 20, 40, 60, 80, and 100 intervals. These sets resulted in a relation matrix of 400, 1,600, 3,600, 6,400, and 10,000 relations, respectively. A single relation was randomly selected for intervals $(0, 1)$ and asserted. This was repeated for each interval pair (I_i,I_{i+1}) to $n-1$, where n is the number of intervals in the test set. As each relation was asserted for the interval pair, the resultant column computation was performed to complete the entries in the matrix.

After logging the initial set of ambiguous relations in these large runs, a test driver was initiated automatically which iterated n times, where n = the number of intervals in the system[5],

[5] While the iteration process could be continued until the temporal system is fully disambiguated, the intent of this process was to gather some empirical data on the relative disambiguation of a temporal system with respect to the number of iterations. Hence, the number of interval was arbitrarily chosen.

taking the next interval pair off the ambiguous relations set, and asserted into the matrix the first primitive relation which was compatible with the ambiguous relation. This was done to progressively disambiguate the system.

This initiated the transitive closure computation once again, which was instrumented to print the number of ambiguous relations at the start of each transitive closure computation and the number remaining at the end of each closure computation. In addition, the actual closure computation was instrumented to provide a comparative CPU time to execute the closure algorithm. While this execution time will vary depending on the execution platform, the intent was to provide some indication of any correlation between the performance of the closure algorithm and the number of ambiguous intervals.

The resultant matrices for the above test sets showing the initial number of ambiguous relations and the number of ambiguous relations remaining after refining *n* relations (where *n* is the number of intervals) and the number of ambiguous relation removed is presented below.

Number of Intervals	Number of Relations	Ambiguous Relations	Ambiguous Relations Remaining	Ambiguous Relations Removed
20	400	107	2	105
40	1,600	651	305	346
60	3,600	1,629	1,291	338
80	6,400	2,953	2,566	387
100	10,000	4,521	3,926	595

Table 7. Summary for large test case set.

The first item of note is the number of ambiguous relations in each of the test sets. These relations within these sets were generated randomly and in all cases the number of ambiguous relations averaged less than $n^2/2$. An interesting observation is the general pattern of $5n$ ambiguous relations that were removed during the test where *n* is the number of ambiguous relations that were removed by the test suite.

In all cases, the number of iterations performed for each ambiguous relation removed was directly dependent on the number of remaining ambiguous relations. Thus, for a system with xR_a ambiguous relations, at most *x-1* iterations are performed for the removal of an ambiguous relation. It was noted that in some cases, several ambiguous relations were removed as part of the transitive closure algorithm. So, for some portion of the test data, less that *x-1* iterations were performed.

4.2.4 Test #4 – Timing Tests

Test #4 uses timing data from the other tests to provide an empirical indication of run time and cycles. Two essential types of timing data were gathered for the algorithm validation. These were Cycle Iterations and Clock Time. The purpose of the first data was to validate the performance and complexity analysis of the algorithm presented in 4.1 above.

Cycle iterations were calculated by counting the number of cycles performed over the set of ambiguous relations for each new relation asserted into the system. This was found to be equal to the number of ambiguous relations. The full test data can be obtained from [Kovarik, 1994]

Clock time was also collected for the 12 interval system of Test #1, but the large scale tests of Test #3 provide more sizable examples. These tests were the systems of 20, 40, 60, 80, and 100 intervals mentioned earlier. In each case, the number of ambiguous intervals present at the start of the transitive closure, the time required to perform the closure computation, and the number of ambiguous intervals present at the end of the closure computation was captured.

Results showed that the overall performance of the algorithm was somewhat better than expected. The limits of the algorithm, in terms of its complexity and related performance were in the range expected. For all test cases, the range of ambiguous relationships were less than n^2 where n = the number of intervals in the system and, consequently, the dimension of the matrix.

For large test cases where the relations were selected at random, the number of ambiguous intervals was nearly n^2. However, in tests where the relations asserted where defined through an

example, such as temporal relations in a story, the number of ambiguous intervals ran somewhat less than n^2. The output from these test cases can be found in [Kovarik 1994].

4.3 Analysis Summary

Based on the data gathered from the test suites developed, the bit-encoded temporal relation representation performed at the predicted efficiency. The algorithm exhibits no signs of exponential growth in time as the number of intervals in the temporal system grows. The main limitation is the space demands for maintaining a full binary matrix. The overall performance was consistently less than the worst case analysis of $O(n^3)$ and was closer to $O(n^2)$ on average.

Limiting the closure algorithm's traversal of intervals to the rows and columns of the interval indexes representing the changed relation significantly improves performance. If no other ambiguous intervals are found in the interval row and column, the closure algorithm halts in linear time. Similarly, if ambiguous intervals are found but they are not refined as part of the closure computation then the closure algorithm halts in linear time.

In the next section, the conclusions are summarized for this research and the results are compared with the original goals.

5. Conclusions

This paper presents an efficient method of computing transitive closures over a set of temporal interval relationships. A sub-algebra of the interval-based temporal algebra of Allen was defined. The mechanism to accomplish this goal was the definition and validation of a bit-mapped algebra for representing temporal interval relations as a set of end point relationships between two intervals. This bit-encoded signature was then used as a basis for computationally deriving the transitive relation from two interval relations.

This algebraic algorithm for transitive relation calculation then formed the basis for a transitive closure algorithm which computed the closure of a temporal system matrix in $O(n^3)$ at the worst case. A key feature of the algebra is the elimination of any dependency on a transitive lookup table. All the relation values for the transitive relation are computed directly from the representation.

Mixing the time points of the interval structures with the temporal relationships in the relationship matrix provided a synergy between the two approaches by supporting the validation of interval relationships via time point values on the intervals and visa versa. This proved to be of significant value for engineering applications of the system.

This hybrid approach allows a temporal plan to be completely specified in terms of the interval relationships between the actions in the plan represented as intervals. As the plan is executed, the initiation of actions within the plan can be tagged with discrete time points as they occur. These time points can then be used to validate the plan by comparing the point relationships between the actual values to see if the actual point relationships yield the same signature between two intervals as the interval relation asserted in the plan matrix.

As with any engineering implementation of a mathematical or physical system, there are inherent limitations imposed on the realization of the system. In this case, the limitations are primarily the reduced algebra representation. It should be noted, however, that the reduced algebra represented by this implementation captures all interval set combinations within the transitivity table defined by Allen. Thus, the limitations of the bit-algebra are those cases where a hypothetical situation is represented by the full algebra. For example, representing the statement, "The light was turned on before or after Vince left the house." is a plausible natural language statement. However, what is being asserted by this statement is a logical or of two hypothetical scenarios.

From an implementation standpoint, there exists some limitation in the subsets of intervals that can be represented using a two-bit value for the endpoint relationship encoding. There are

subsets, such as (p m), of other sets in the algebra, such as (p m o), which are valid. Currently, because the bit-encoding representation utilizes two bits, it cannot capture these subsets uniquely.

The bit algebra was developed based on the premise that the only relations that can be asserted between the endpoints of intervals are less than, greater than, equal, and unknown. This results in a set of ambiguous relationships that provide a representation where the unknown endpoint relationship could be one of less than, equal, or greater than.

In fact, there are opportunities for a finer granularity of representation. For example, when computing the transitive relation for (***A overlaps B***) and (***B overlaps C***), the resultant ambiguous relationship represents the fact that, without additional information, it is impossible to ascertain whether the relationship between **A** and **C** is *precedes*, *meets*, or *overlaps*. This is accurate when computing the transitive relation from two overlaps relationships. However, it is also valid to state that (***A overlaps*** *or* ***meets C***).

Using a two-bit code for endpoint relations cannot capture this finer grained ambiguity. The potential exists, at the expense of a bit more space to represent the interval relations, to expand the bit code representation of the end point relations such that it is possible to capture the finer grained ambiguities of the algebra. For example, using a four bit code for end point relationships would allow the use of individual bits for each of the end point relationship types.

6. References

All83a Allen, J.F., Maintaining Knowledge about Temporal Intervals, *Communications of the ACM 26 (11), November 1983, 832-843.*

All83b Allen, J.F. and Koomen, J.A., Planning Using a Temporal World Model, Proceedings of the 8th International Joint Conference on Artificial Intelligence (Karlsruhe, W. Germany), Morgann Kaufmann 1983, 741-747.

All84 Allen, J.F., Towards a General Theory of Action and Time, *Artificial Intelligence 23 (2), July 1984, 123-154.*

All85a *Allen, J.F. and Kautz, H.,* A Model of Naive Temporal Reasoning*, in Hobbs, J.R. and Moore, R.C., editors, Formal Theories of the Commenses World, Ablex 1985.*

All85b *Allen, J.F. and Hayes, P.J.,* A Commensense Theory of Time*, Proceedings of the 9th International Joint Conference on Artificial Intelligence (Los Angeles, California), Morgan Kaufmann 1985, 528-531.*

All87 *Allen, J.F. and Hayes, P. J.,* Short Time Periods*, Proceedings of the 10th International Conference on Artificial Intelligence (Milano, Italy), Morgan Kaufmann 1987, 981-983.*

All89a *Allen, J.F.,* It's Time for Planning*, Proceedings ot the Seventh Conference of the Society for the Study of Artificial Intelligence and Simulation of Behavior, London, UK, 1989, 227 (Abstract only).*

All89b *Allen, J.F. and Hayes, P.J.,* Moments and Points in an Interval-Based Temporal Logic*, Computational Intelligence, Vol. 5, No. 4, November 1989, 225-238.*

All91 *Allen, J.F.,* Time and Time Again, The Many Ways to Represent Time*, International Journal of Intelligent Systems, (6) 4, July 1991, 341-355.*

Dec92 *Dechter, R., Meiri, I., and Pearl, J.,* Temporal Constraint Networks*, Artificial Intelligence 49, 1991, 61-95.*

Dor92 *Dorn, J.,* Temporal Reasoning in Sequence Graphs*, Proceedings of AAAI-92, San Jose, CA, 1992, pp. 735-740.*

Gal91 *Galton, A.,* Reified Temporal Theories and How To Unreify Them*, Proceedings of the Twelfth International Conference on Artificial Intelligence, 1991, Volume 2, 1177-1182.*

Gol92 *Golumbie, M. C., and Shamir, R.,* Algorithms and Complexity for Reasoning About Time*, Proceedings of AAAI-92, San Jose, CA, 1992, pp. 741-747.*

Hal86 *Halpern, J.Y. and Shoham, Y.,* A Propositional Modal Logic of Time Intervals*, Proceedings of the Symposium on Logic in Computer Science, 279-292, IEEE Computer Society Press, 1986.*

Kov94 *Kovarik, V. J. Jr.,* A Computationally Efficient Method for Representing and Computing Transitive Closure Over Temporal Intervals*, Ph.D. Dissertation, University of Central Florida, 1994.*

Lad86a *Ladkin, P.B.,* Time Representation: A Taxonomy of Interval Relations*, Proceedings of AAAI-86, 360-366, Morgan Kaufman, 1986.*

Lad86b *Ladkin, P.B.,* Primitives and Units for Time Specification*, Proceedings of AAAI-86, 354-359, Morgan Kaufman, 1986.*

Lad87a *Ladkin, P.B.,* Constraint Satisfaction in Time Interval Structres I: Convex Intervals*, Kestrel Institute Technical Report KES.U.87.11, 1987.*

Lad87b *Ladkin, P.B.,* The Logic of Time Representation*, Ph.D. Thesis in Logic and Methodology Science, University of California, Berkeley, 1987.*

Lig90 *Ligozat, Gerard,* Weak Representations of Interval Algebras*, Proceedings of AAAI-90, Boston, MA, 1990, pp. 715-720.*

She92 *Sheppard, J. W. and Simpson, W. R.,* Fault Diagnosis Under Temporal Constraints*, IEEE Autotestcon Conference Record, 1992, 151-159.*

Van89 *Van Beek, P.,* Approximation Algorithms for Temporal Reasoning*, Proceedings of the 11th IJCAI, 1291-1296.*

Van90 *Van Beek, P.,* Reasoning About Qualitative Temporal Information*, Proceedings AAAI-90, Boston, MA, 1990, 728-734.*

Vil82 *Vilain, M.,* A System for Reasoning About Time*, Proceedings of AAAI-82, Pittsburgh, PA, August 1982.*

Vil86 *Vilain, M. and Kautz, H.,* Constraint Propagation Algorithms for Temporal Resoning*, Proceedings of AAAI-86, Morgan Kaufmann 1986, 377-382.*

Vince:

I have gone over the paper with a fine tooth comb. I had a little trouble understanding it. So, I guess the comments made by the reviewers were for the most part valid. I started looking at the comments, but then I decided that I first needed to understand what was in the paper before I could try to respond to the comments. I was hoping not to have to ask you as many questions as I have, but I think you are in a better position to answer them than I am.

My primary goal was to structure the paper. Although I'm not certain I have done that correctly, specially concerned about section 4, the testing. I am not really certain how many types of tests you actually ran, but my count came to 4. Please verify that the text that I cut and pasted goes with the correct test. I have made several other smaller changes, and not all of them are obvious. I suggest that you answer my questions first, and send the paper back to me. I can then work on it a bit more, and probably generate some new questions, but hopefully fewer and less momentous.

I like the paper very much, but it needs help in clarity. Can you get this back to me within a month or two? I would like to get it off to the journal by mid-summer if at all possible.

Let me know.

Avelino