An efficient method for representing and computing transitive closure over temporal relations

1994

Vincent J. Kovarik
AN EFFICIENT METHOD FOR REPRESENTING AND COMPUTING TRANSITIVE CLOSURE OVER TEMPORAL RELATIONS

By
Vincent J. Kovarik Jr.

1994

UCF
AN EFFICIENT METHOD FOR REPRESENTING AND COMPUTING TRANSITIVE CLOSURE OVER TEMPORAL RELATIONS

by

VINCENT J. KOVARIK JR.
B.S., State University of New York at Stony Brook, 1974
M.S., George Mason University, 1986

DISSERTATION
Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Engineering
Department of Electrical and Computer Engineering
College of Engineering
University of Central Florida
Orlando, Florida

Summer Term
1994
ABSTRACT

The need for temporal reasoning is found throughout the engineering disciplines. James Allen introduced a representation for temporal reasoning based upon the concept of intervals. This approach provides a rich set of temporal relations for reasoning over events and changes in state. The full temporal algebra is NP-complete however. The algorithm developed by Allen executes in $O(n^3)$ time but only ensures consistency between any three intervals.

This research presents an approach to representing interval relations as a bit-encoded form which captures the relationships between the end-points of the intervals. A bit-algebra is then defined which provides an algorithmic method for computing transitive relations without requiring the table lookup of Allen's algorithm. By reducing the set of ambiguous interval representations to the set of relationships which have unknown temporal extent, a robust subset of the full algebra is defined which maintains the direct computation of transitive relationships.

A transitive closure algorithm is then developed which has the property of being $O(n^3)$ as the worst case rather than the average. Data gathered shows the closure algorithm to run in $O(n^2)$ time for most cases. The bit code for a relationship is a single byte. Thus, the storage requirements for a temporal system of $n$ intervals is $n^2$, a relatively small memory demand. The small memory footprint, coupled with the efficient transitive relation calculation and closure algorithm together form an efficient method for providing temporal reasoning capabilities to a wide range of applications.
ACKNOWLEDGEMENTS

Upon achieving a significant goal or completing a substantial project such as this, the natural reaction is to bask in the glow of your accomplishment. After the initial blush of self-congratulation fades, however, it becomes clear that you did not accomplish the goal alone. All along the way were friends, relatives, co-workers, and other individuals who helped contribute to your success. I would like to take this opportunity to thank those individuals.

There are more people which have contributed to this work than I have the memory or space to thank here. My thanks goes out to each of you. There are, however, several individuals which have made significant differences in my life and this research. First of all, I would like to extend my thanks to Dr. Avelino Gonzalez, my advisor and friend, without whose support this long process might not have reached its culmination. To my good friend and card-bearing member, Dr. Dave Castillo, thanks for the concept discussions during our drives to Orlando and continued personal and professional friendship. To my parents, thanks for instilling a respect for education and especially to my father whose years of research at Brookhaven National Laboratory and love of knowledge has been a beacon.

Finally, my deepest thanks and love goes to my wife, Madeline, and daughters, Meredith and Claire. Without your constant love, understanding, and support this, and so many other things in my life, would be unachievable and without meaning.
# TABLE OF CONTENTS

| LIST OF TABLES | ........................................................................................................ | viii |
| LIST OF FIGURES | ........................................................................................................ | ix  |
| CHAPTER 1 INTRODUCTION | ................................................................................................. | 1  |
| Overview and Motivation | ................................................................................................. | 1  |
| Objectives | ................................................................................................. | 4  |
| Contributions | ................................................................................................. | 5  |
| Applications of Research | ................................................................................................. | 6  |
| Evaluation of Results | ................................................................................................. | 6  |
| Dissertation Organization | ................................................................................................. | 7  |
| CHAPTER 2 VIEWPOINTS OF TIME | ................................................................................................. | 8  |
| Historical Viewpoints | ................................................................................................. | 9  |
| Linear Time | ................................................................................................. | 10 |
| Cyclical Time | ................................................................................................. | 11 |
| Quantum Time | ................................................................................................. | 12 |
| Philosophical Views | ................................................................................................. | 13 |
| Approaches to Temporal Reasoning | ................................................................................................. | 15 |
| Point-Based Logics | ................................................................................................. | 16 |
| Interval-Based Logics | ................................................................................................. | 22 |
| Temporal Constraints | ................................................................................................. | 33 |
| An Example | ................................................................................................. | 37 |
| Point Representation | ................................................................................................. | 38 |
| Constraint Representation | ................................................................................................. | 43 |
| Interval Representation | ................................................................................................. | 48 |
| Summary | ................................................................................................. | 53 |
| CHAPTER 3 A BIT-MAPPED TEMPORAL ALGEBRA | ................................................................................................. | 55 |
| Defining the Sub-Algebra | ................................................................................................. | 56 |
| Ambiguity Versus Hypothesis | ................................................................................................. | 57 |
| Interval Structure and Reference Frames | ................................................................................................. | 58 |
| Events, Processes, and States | ................................................................................................. | 58 |
| Bounding Intervals | ................................................................................................. | 60 |
| Interval Semantics | ................................................................................................. | 63 |
| The Temporal Algebra | ................................................................................................. | 65 |
| Points and Intervals | ................................................................................................. | 65 |
| Representing Relationships via Directed Graphs | ................................................................................................. | 74 |
| Directed Graph Representation of Basic Relationships | ................................................................................................. | 75 |
| Representation of Transitive Relations and Ambiguity | ................................................................................................. | 77 |
LIST OF TABLES

Table 1. Allen's Temporal Intervals ........................................... 23
Table 2. The Transitivity Table for Interval-Based Relationships ..... 25
Table 3. Example Temporal Assertions ...................................... 50
Table 4. Adding Fourth Interval to Example .............................. 51
Table 5. Fifth Interval Added to Problem .................................. 52
Table 6. Additional Relation Assertion to Example .................... 53
Table 7. Bounding Relationships for Intervals ......................... 62
Table 8. Interval Relationship Signatures ................................. 63
Table 9. All Possible Paths Between Bounding Intervals .......... 64
Table 10. Bit Encoded Temporal Relationships .......................... 81
Table 11. Bit Encoded Representations of Allen's Temporal Interval Relationships ........................................... 82
Table 12. Product Table for the Temporal Algebra .................... 88
Table 13. Addition Table for Temporal Algebra ........................ 89
Table 14. Table of Bit-Encoded Relationships ............................ 101
Table 15. Table of Bit-Encoded Relationships ............................ 135
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Seven Temporal Relationships</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Simple Transitive Relationships</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Ambiguous Transitive Relationships</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Nests Showing Bounds of Interval</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Operators for Allen's Interval-based Temporal Logic</td>
<td>30</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Labeled Edges Forming Constraints Between Temporal Points in a Time Map</td>
<td>33</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Database Persistence versus Time Map Persistence</td>
<td>34</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Celsius vs. Fahrenheit Temperature Constraint Equation</td>
<td>36</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Simple Temporal Constraint Between Two Nodes</td>
<td>37</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Initial Set of Temporal Points</td>
<td>39</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Addition of Second Event Sequence Points</td>
<td>39</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Additional of Temporal Point and Equivalence Relation</td>
<td>40</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Insertion of Temporal Point Preceding T7</td>
<td>41</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Additional Temporal Point</td>
<td>42</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Inserting Point and Equivalence Relation</td>
<td>42</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Inserting an Ordering Constraint Between T9 and T8</td>
<td>43</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Initial Temporal Constraint</td>
<td>44</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Incorporating Second Event Into Constraint Network</td>
<td>44</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Merging an Event Arc with an Existing Event Node</td>
<td>45</td>
</tr>
</tbody>
</table>
Figure 64: Logarithmic Plot of Number of Intervals versus Number of Ambiguous Relations

Figure 65: Example Showing Error in Transitivity Table
CHAPTER 1
INTRODUCTION

This thing all things devours:
Birds, beasts, trees, flowers;
Gnaws iron, bites steel;
Grinds hard stones to meal;
Slays king, ruins town,
And beats high mountain down.
J. R. R. Tolkien, The Hobbit

Overview and Motivation

Time continues to be the subject of scrutiny, study, and debate. Our knowledge, concepts, truths, and beliefs that govern our interactions with the world around us are inextricably dependent on time.

The need for temporal representation and reasoning is interwoven throughout the field of engineering. A process control system must compensate for feedback delays in adjusting control variables. Industrial engineers need to incorporate time requirements for items such as construction dependencies, curing process, and task networks. Electrical and computer engineers apply temporal reasoning in the modeling and analysis of circuit delays, multi-process interactions, pipelining, and other digital circuit design issues. Software engineering must model software interactions over multiple processes, network propagation delays, and inter-process communication issues. Time is integral part of the field of artificial intelligence, particularly knowledge representation.
and natural language understanding. Lastly, time is present in the matrices of
differential equations modeling physical systems.

The goal of enabling a computer to reason over time is to provide more
expressive and powerful tools with which we can expand our understanding of
the universe around us. It is not only crucial that the temporal reasoning
mechanism be mathematically valid but also be logical, from a human point of
view. Thus, before we can construct a machine which understands and reasons
about time, we must come to an understanding of our own concepts of time, its
properties, and how we apply it in our interactions with the world.

A number of approaches to representing time have been proposed, each
with a particular focus and associated strengths and weaknesses. A problem
common to all approaches has been the development of a tractable
representation method that is concise and computationally efficient but still
captures a wide range of temporal nuances.

The traditional tradeoff is to sacrifice expressiveness for computational
efficiency. Discrete time points provide a concise method for representing and
performing computations but lack the ability to capture the interpretation of
recurring events or event relationships. For example, "He practiced batting in the
cage for three hours" or "The shortstop caught the line drive and threw the runner out at
first for the double play" provides examples of an event being repeated over some
period and relationships between two events. In a point-based view, the batter
would be in the cage after a specific point in time and out of the cage after a
specific point in time but capturing the notion of repeatedly swinging at the ball
could not be represented easily in a point based approach because we do not
know how many times the batter swung or when. In the second example, an
important temporal link is established between two events, catching the ball and
throwing it to the base before the runner returns to the base. Together they form a single event called a double play. Thus, it is the relationship between the two events that is important rather than knowing exactly when the events occurred.

The simplest representation of time within a system is the use of time points associated with states. In this viewpoint, a change in state of a variable is associated with a time point representing the moment in which the variable changed from its old value to its new value. Values are assumed to be unchanged from that point forward in time until the next time point change is identified.

Defining temporal relationships, specifying constraints between events or intervals, and ordering actions can range from specific time values to general relationships between entities. Temporal understanding ranges from explicit definition of events with well defined time boundaries to qualitative references between events describing a partial ordering of actions.

The focus of this work is to define a temporal representation formalism that provides a terse representation of relationships based on a unique signature or type. Combined with this minimalist representation is a computationally efficient algorithm for computing transitive relations directly from the representation without requiring table lookup or sets of relations. The benefit of this representation and associated algorithm is the capability to efficiently compute transitive relationships between a given set of intervals using very simple mathematical operations.

The key benefits of this research is the development of computer algorithms and data structures which provide the temporal expressiveness required but are more computationally tractable than previous implementations of an interval-based temporal algebra. These algorithms and data structures may
be applied in a number of engineering disciplines ranging from planning systems to more flexible scheduling. By providing a seamless integration between the qualitative temporal reasoning and discrete temporal reasoning substantial benefit is provided to the application.

**Objectives**

This research focuses on the development of a computationally tractable method for representing relationships between temporal intervals. It provides a computational formalism for computing transitive closure over a set of temporal relations. Specifically, the ability to capture and represent relationships in an efficiently encoded form that is both concise and expressive enough to capture all the interpretations of possible relationships between any two intervals within a system. This is in contrast to the approach developed by Allen [All83] in which it is impossible to ensure consistency beyond sets of three intervals. Allen argues that the computational price for ensuring global consistency would be prohibitive and not necessary for most applications of the approach.

Specific objectives of this research are:

1. to identify a subset of a general temporal algebra which embodies a wide range of the qualitative nuances of general interval relationships.

2. to define a machine representation for the sub-algebra which embodies the expressiveness of an interval-based representation but maintains the computational efficiency of point-based approaches.

3. to develop a transitive closure algorithm based on the representation defined which performs transitive closure computation faster than $n^3$, worst case (which is the current performance of an interval algebra).

4. to develop a computational approach to computing the transitive relation for any three relations.

5. to ensure consistency within the entire network of temporal relations.
Contributions

Contributions of this research are the development of a method for concisely representing interval relationships and computing the transitive closure over a set of temporal intervals. This capability will provide benefit to a wide range of applications such as robotics controls, planning systems, simulation systems (both qualitative and quantitative), conventional computer software applications, and any system which requires an understanding of time.

Specific contributions are described below.

1) Representation of temporal relationships in a space efficient form which captures ambiguous temporal relation interpretations between a set of intervals.

2) An encoding assignment method which yields a unique signature for all possible relationships between two intervals.

3) A relational structure and organization which allows the system to compute all possible relationships between all intervals in less than $n^3$ polynomial time.

4) The ability to identify conflicts between newly asserted relationships and prior transitive calculated values.

The above contributions form a significant extension to the current research in temporal representation and reasoning. Of particular note are the objectives of an encoded representation which captures all interpretations of possible relationships between two intervals and the computation of all possible relationships in a set of intervals in polynomial time.

Also of note is the objective to identify conflicts in consistency between relationships already asserted or computed and newly added relationships. This capability is not currently possible, for example, with Allen's [All83a] transitive computation algorithm which identifies inconsistencies in sets of any three intervals.
Applications of Research

The results of this research may be applied to a wide range of industrial and government applications. A robust and efficient mechanism for computing transitive closures over large interval sets will greatly enhance the accuracy and completeness of planning systems. Better performance in reasoning about temporal relationships could be applied to support more robust tracking and control of air traffic. Potential areas of use include autonomous robotics planning, temporal projection for collision detection and avoidance in computer-controlled robotics manufacturing plants, flight plane analysis and flight control for the aviation industry, plan generation for assembly tasks, and other similar problems.

As the application and use of industrial robots become more widespread and autonomous in operation, the ability to embed efficient-temporal reasoning becomes a critical aspect to the success of these systems. Applications for compact and efficient-temporal reasoning include embedded applications for obstacle detection and collision avoidance, the understanding of event relationships within a natural language story, and many other areas.

Evaluation of Results

The results of the research will be evaluated quantitatively via experimental trials, collection of performance data, and evaluation of the data collected. The data evaluation will provide empirical verification of the approach. Performance and space demands will be plotted and analyzed. These will form the basis of the verification of the algorithms and data structures developed. The overall goal is the development of an algebra which ensures global consistency for a temporal network in polynomial, $O(n^2)$, time.
Dissertation Organization

This dissertation is organized into 6 chapters. The content of the remaining chapters is presented below.

Chapter 2 provides an overview of human viewpoints and temporal reasoning capabilities. A historical perspective is used to illustrate how human perception of time has changed.

Chapter 3 describes the basic foundation of the computational algebra which forms the core research of this dissertation. In this chapter the essential sub-algebra is defined, comparisons to other interval-based algebras are presented, and the transitive closure algorithm is specified.

Chapter 4 presents an analysis of the representation and transitive closure algorithm. The algorithm's complexity is analyzed and compared with other interval algebra transitive closure algorithms. The second part of Chapter 4 shows the implementation of the algebra in software and discusses the various components and their interaction.

Chapter 5 presents and analyzes the empirical data gathered through testing of the computer implementation of the algorithm. Anticipated performance versus the performance observed through testing is compared and analyzed.

Chapter 6 provides conclusions of the research effort. It presents an overview of the research and assesses its overall success. It identifies areas of the algebra and its programmatic implementation which require further research or extensions, summarizes strengths and weaknesses of the algebra and its implementation, and evaluates its effectiveness in specific application areas.
CHAPTER 2
VIEWPOINTS OF TIME

Now the thing about time
Is that time isn't really real.
It's just your point of view
-- James Taylor, Secret O' Time

Despite our human ability to effortlessly reason over time, temporal reasoning remains a complex and process-intensive endeavor for machines. A number of approaches to representing and reasoning about time in a machine have been developed from various viewpoints of time.

Reasoning about time and space is a learned process. It is one of the commonsense reasoning abilities that we learn later than other essential reasoning capabilities. As children grow, their grasp of the world around them is truly an amazing achievement. As they learn to speak and communicate, they are capable of remarkably lucid conversation as early as three years old. Yet, although they have an in-depth grasp of the physical world, an understanding of simple causal relationships, and a conversational vocabulary that puts any natural language system to shame, they still lack a comprehensive understanding of time. The concepts of time of day, relationships and ordering of events, and temporal projection are still being mastered.
Whether this is due to the complexity of the domain or the need for a solid grasp of language before it can be adequately expressed or some other factor is yet to be determined. The essential fact is that humans possess a unique ability in the understanding, expression, and reasoning over time.

What makes this capability particularly intriguing is our ability to comprehend complex interactions and sequences of events coupled with spatial relationships over time. This capability includes the ability to integrate qualitative temporal relationships with numerical time points with seemingly little or no effort. It is only when the magnitude of the temporal events or their complexity greatly increases that we resort to a more "formal" method of representation.

Time is perceived in many forms with different interpretations. To build a temporal reasoning system, we must understand the different aspects of time, how it is perceived by other individuals, applied to reasoning, and conceptualized before we can develop machines that reason about time. The next section provides a brief insight into the evolution of human temporal understanding. It presents a historical perspective of human temporal understanding and identifies similarities between these perspectives and current work in temporal reasoning.

**Historical Viewpoints**

Throughout history, the human perception and understanding of time has evolved with our perception and understanding of the world around us. This section provides a brief overview of the various perspectives that human kind has held.
Linear Time

The Moving Finger writes; and, having writ,
Moves on: not all your Piety nor Wit
Shall lure it back to cancel half a Line,
Nor all your Tears wash out a Word of it.
-- E. Fitzgerald, "The Rubaiyat of Omar Khayyam"

As the above quotation eloquently states, time has long been viewed as a uni-directional entity. Life must be lived to the fullest possible for each moment, once gone, can never be relived or changed. Indeed, if asked, most people would agree with this viewpoint of time. Time is perceived to be uni-directional and unchanging in its inexorable flow towards the future. This notion is also in concert with virtually all developments in temporal reasoning. Whether the basis of the representation is a set of time points [All89b, Bel87, Vil82], intervals [All84, All88, Lad86a, Lad86b], time maps [Dea87, Dea88], constraint satisfaction [Dec88, Dec92, Vil86], linear inequalities [Val86], or directed graphs [Dea86, Fox78, Mal90], time is implicitly assumed to be uni-directional.

The fact that time is uni-directional may seem an obvious conclusion and not worth the effort to mention. However, many of the scientific disciplines, discoveries, and mathematics work equally well in reverse. Newtonian physics and Einstein’s theories of general relativity are both equally valid for time running in reverse. To an extent, Einstein’s relativity demands that the direction of time be reversed once the speed of light is exceeded.

For example, consider the equation for calculating the position of an object with constant velocity at a given point in time, \( \text{position} = \text{velocity} \times \text{time} \). Assume that the position of an object at \( t=0 \) is 0, this removes the need for any constant terms in the equation. Then, if the velocity of the object is 100 meters per second,
the position of the object at time $t=1$ is 100 meters, $t=2$ is 200 meters, and so on. However, the validity of this equation does not change if time is reversed. While not affecting any of the positions for $t>0$, positions are equally valid for $t<0$ (i.e. when $t= -1$, the position is -100 meters). Thus, time need not run in a forward direction for these laws to maintain their validity but rather are dependent upon the reference frame in which they are viewed.

**Cyclical Time**

Before the notion of linear time was developed early cultures and societies viewed time as the natural pattern of repeating cycles. The seasons, lunar phases, tides, and stars all portrayed a cyclical viewpoint of time. Time was, to early societies, a cycle of a never-ending journey around a predictable cycle of events. This view led many early civilizations to believe in historical cycles of calamity, war, and famine. This notion of temporal reasoning is still with use today. Although we no longer depend on the seasons or lunar cycle for time keeping, cyclic phenomena is very much a part of engineering disciplines. Crystal oscillation provides the clocking mechanism for a computer, wrist watches, and the digital station tuner in a television set. Cycles are intertwined in the generation, reception, and interpretation of electrical signals, radio waves, fiber optics, and music. Cyclic time is still represented in the very way we keep time, in repeating cycles of 24 hours.

Cyclic activities are typically not addressed very well in temporal reasoning systems. Point-based systems use a method of cycle time to project when the next point for a cyclic event will occur. Interval based approaches [Lad87a] have employed a technique of defining a single interval which bounds a set of repeated or convex intervals.
Quantum Time

This dichotomy between points and intervals is compounded further in a quantum view of time. In studying the physics of atomic particles, an electron orbiting about the nucleus of an atom can be mathematically expressed as a wave form or a particle. In fact, it can be viewed as either according to the Mathematics of Quantum Physics. Consequently, it must collapse into one state or the other when it is observed by some entity outside the system. This "selection" of a state is both enticing and puzzling. The concept that an entity can have the properties of both an electromagnetic wave and a physical particle but not at the same time is itself paradoxical. This quantum view of time can be found in [Lad86b] which proposes units for temporal reasoning systems. Upon inspection, we see that the units are, in fact, discreet quanta applied to a continuous representation system. These quanta are actually lists of time units in descending order. For example, the quanta, January 1, 1994 would be represented as (1994 1 1). This quanta is treated as a point when manipulating temporal relations which have extents longer than a day. However, if a finer-grained set of quanta is required, then the list is augmented with the next lower temporal unit, hours. So, 3:00 PM on January 1, 1994 would be represented as (1994 1 1 15). Thus, a quantum (i.e. point) view of time is applied to a continuous (i.e. interval) representation viewpoint.

Unlike understanding physical entities, gaining and understanding of time is a learned capability and is not as intuitive to comprehend as the physical objects in our world. The problem is that time is not a physical substance. It is, however, an abstract entity which must represented and understood to represent and reason about physical items in the world. The world in a constant state of
flux. There is no axiom such as "standing still." References to time within our everyday conversation is replete with temporal references.

"I'm going to the store after my exercise class, then I'll stop and pick up some milk."

"The exhaust manifold must be removed before unbolting the head gasket."

"The negative charge on the hollow cathode must be raised immediately prior to the introduction of Argon gas in order to ignite the plasma."

The above sentences are just three small examples of the reference and use of time in the expression and action involved in interacting with the physical world.

**Philosophical Views**

Philosophical viewpoints of time have varied. In Plato's *Timaeus* [Lee71] the distinction is made between *Being* (a destination or state) and *Becoming* (the journey or process undertaken to arrive). This viewpoint is still part of our society and its language. When a person has achieved a significant goal, we say they "have arrived." Zeno, a pupil of Plato, identified temporal paradoxes which were intended to give cause to think about time and it's perception. One of the best known of his paradoxes is that of Achilles and the tortoise which claims to show that motion is impossible if time is a substance which can be infinitesimally divided. In the paradox, Achilles is depicted chasing a tortoise. In the time it takes Achilles to reach the position where the tortoise started, the tortoise has moved forward some small but finite amount. In the time taken for Achilles to cover that distance, the tortoise has again moved forward, and so on. This paradox can only be resolved, according to Whitrow [Whi80], in one of two ways.
One of these is to reject the notion of "becoming", thus reducing the temporal system to a point-based interpretation (i.e. it is composed of non-divisible time points). In this view, time can only be represented as a series of discrete moments or points. There is no ambiguity of action and all entities in the world assume a given state at discrete points in time.

The alternative way to resolve the paradox is to reject the view that time is indivisible and, thereby, embrace an interval-based viewpoint based on temporal units which have a duration. Thus, time consists of set of actions or events which occur over some interval of time. Points are not a component of this view.

Even this philosophical view of time subscribes to the premise that time can only be viewed in one of two ways, either as intervals or points, but not in both. This philosophical viewpoint of time and the dichotomy in resolving the paradox reflects the tensions between point-based and interval-based temporal theories. In fact, Plato's notions of Being and Becoming is a mirror of the differences between point-based representations and interval-based representations of current research. In the former, the destination or end state is the only thing of importance and, hence, represented in a point-based algebra (i.e. we have rejected the process of becoming). While in the latter, the journey (i.e. interval) is the important concept and have implicitly chosen a time which is not infinitely divisible. In [All85b] and [All89b] intervals are described in terms of "nests" which are intervals of arbitrarily small duration. The duration of these nests are small enough such that the intervals take on the properties and interpretation of points.
Approaches to Temporal Reasoning

Temporal reasoning by machine is still an error-prone, computationally complex, and intractable problem. The computational complexity of temporal reasoning has not yielded a compact method for representing these concepts nor an efficient computational method.

Temporal reasoning is necessary in engineering disciplines such as autonomous robotics, assembly planning, and air traffic control. Each of these areas require extensive understanding of relative time, the temporal projection of events and their effect on the environment in which the system lives. Not only must these systems and others like them support temporal reasoning, but they must support it in such a way that allows for varied degrees of time. The above systems consist of temporal entities for which detailed information can be provided and entities which may have nothing more than a relation specification between itself and another entity.

Prior work in temporal representation and reasoning focused on a representation which provided support in a single area, from intervals [All83], to time maps [Dea86], situational calculus [McC63], and point-based methods [Vil82], [Vil86]. Interval-based temporal reasoning focused on the capture and representation of events as entities which require some duration of time but employed relationships between intervals to capture the dependencies.

Activities such as understanding natural language, expert systems, machine learning, and diagnosis and planning all encompass developing conclusions based on events, or representing real-world activities and, consequently, must provide some accounting for temporal reasoning. Consider natural language understanding. In the simple sentence, "John went to the store for milk on his way home from work." there are several temporal relations, both explicit
and inferred. The two main events referenced are the action of proceeding home from work and the action of buying some milk. The immediate assumption is that sometime during the event of proceeding home from work, John performed the action of buying some milk. Thus, within a single simple sentence, is the notion of an event or action being performed during another.

This reviews various approaches to representing and reasoning in the temporal domain. The strengths and weaknesses of the various approaches will be presented from the point of view of their suitability and expressiveness for the physical world.

In general, the representation methods for temporal reasoning range from the discrete to continuous. The approaches which use time-point algebra and numerical representations are in the discrete category. The relative approaches, such as the interval-based methods, provide more flexibility and expressiveness in the areas of abstract or relative time.

The balance of this chapter organizes the prior work into one of three basic approaches, 1) point-based, 2) interval-based, or 3) constraint-based. The last category can be viewed, through some transformation, as a representational form of either point or interval temporal algebra. However, a significant amount of work has been performed, particularly much of the recent research in temporal reasoning, which use a network representation of constraints. Thus the properties of a constraint system warrants promoting it to its own category.

**Point-Based Logics**

This section describes work which views time from the point perspective. This covers a range of effort from Situational Calculus [McC63], to the point algebra of Vilain and Kautz [Vil82]. Also included are representation of time
using linear inequalities, and finally a "quantum" view of time proposed by Ladkin [Lad86b]. Although Ladkin's units of time can be interpreted as intervals, for the purpose of this discussion, they are viewed as points, since they are bound, finite units.

**Situational Calculus**

Situational calculus can be viewed as one of the earliest approaches to temporal reasoning. It is built on a point-based view of the world, that is a fact is true until proven otherwise. In most systems, this is the approach taken to represent a datum or fact. It is simply asserted and, when it is referenced again, it is still assumed to be true. No consideration is given to conditions or events which may occur within the system that may invalidate the fact. The only action which is significant is the retraction of the fact or assertion of a new fact in its place.

At first glance, this approach seems entirely appropriate and, in fact, is how we form assumptions about facts over time. For example, if I park my car in the morning before going into my office then I assume that the car will still be there when I leave to return home. This assumption is made however, based on the lack of knowing any facts which may invalidate the assumption. For example, my car may have been stolen or towed.

Problems creep into this method almost immediately when attempting to model the physical world around us. For example, consider the following scenario:

*John sees Sam at noon. He says, "My lawn is too high."*
Later on in the day, about six o'clock, if asked about John's lawn, Sam would likely reply that it is "too high." However, consider the following conversation.

*John sees Sam at noon. He says, "My lawn is too high. John Jr. is cutting it now."*

Then, when asked at six o'clock about John's lawn, he might reply that it is "short" or freshly trimmed. This second statement provides an indication that the fact that John's lawn is high is probably no longer true. When asked at six o'clock, Sam automatically notes that six hours have elapsed since he talked to John and makes the assumption that John Jr. has completed cutting the lawn. An important point in the above example is that we automatically factor in temporal considerations when considering the validity of facts previously known to be true.

A variant of the above might be that, after seeing John around noon, he is asked about John's lawn at twelve-thirty. Instead of replying that it is short, he might reply that it is long. Again time is factored into reasoning about a fact. Since only a half hour had passed between the time that Sam talked with John and when Sam was asked about the lawn, he assumes that John Jr. has not completed mowing the lawn. Thus, the default assumption remains true.

This example shows several key elements regarding temporal reasoning and making default assumptions. While it can be generally regarded that once a fact is asserted it continues to be true, that is based on no knowledge of any other event or contradictory facts asserted since the original assertion. Thus, we do not simply assume a fact to be still true. We assume a fact is still true because of the explicit knowledge that there is no other assertion which contradicts the original
fact. Thus, human temporal reasoning makes use of our introspective capabilities. We implicitly search for facts which negate a fact previously known as true and reach the conclusion that we don't know of any contradictions. Thus, the prior established fact is still believed to be true. This is a clear distinction from the logic-based approaches in which the value false is indistinguishable from unknown.

Frame-based representations of temporal aspects of knowledge typically employ a "snapshot" in time approach. That is, the frame structure maintains a picture of the last change effected on the frame structure. This is similar in behavior to conventional databases. We can view the last change but not prior values or future projections.

McCarthy [McC69] expounded on the problems encountered when attempting to represent real-world events within the structural confines of a frame system. The "point in time" view of the frames is precisely the problem described by McCarthy.

Point Algebra

In point-based algebra, time consists of an ordered collection of time points which have no duration. In [Vil86], Vilain and Kautz state that interval-based representations can be modeled in a point-based algebra using point pairs as markers for the beginning and end of an interval. In fact, Vilain and Kautz make the assertion that this approach must be used rather than employing the full interval-based algebra of Allen [All83] because, based on an analysis of Allen’s relation propagation algorithm, they assert Allen’s algorithm\(^1\) has an

\(^1\)This algorithm is presented in Chapter 3.
average performance complexity of $O(n^3)$ rather than $O(n^2)$, as originally asserted in [All83].

Another viewpoint is that a point can be represented by the mathematical notion of limits. Thus, while all intervals have some finite duration, $d$, a point, $P$, is an interval, $I$, in which the duration approaches zero. Thus, a point can be represented as an interval with a duration of zero.

$$\lim_{d \to 0} (I_d \Rightarrow P)$$

This convention provides some convenient approaches to representing time such that both interval-based and point-based representations are consistent and allows for the incorporation of "real time" into the set of temporal relations.

This approach allows the relationship between time points and time intervals to be represented within the same set of thirteen temporal intervals originally developed by Allen [All83]. Davis [Dav90] proposes a similar relationship between points and intervals by proposing that points are, in fact, the start and end of an interval. He represents these points as function symbols to represent the start and end of an interval, called $\text{start}(I)$ and $\text{end}(I)$. Davis then defines a lower_bound of an interval $I$ as a point $X$ such that $X$ is less than or equal to every element, $y$, of $I$.

$$\text{lower_bound}(X, I) \iff \forall_{y \in I} X \leq y$$

The same argument can be applied to determine an upper bound for an interval $I$. This leads to the assertion that the lower bound, $X$, of the interval $I$, is the starting point of the interval.
Linear Inequalities

When time is quantified it is possible to represent relationships between intervals or events using linear inequalities [Mal83, Val86]. This has the advantage of enabling the computation of relationships to be a simple algebraic operation. The representation of computation of boundary conditions and tests for feasibility are well understood and implemented using well-known algorithms. This approach does assume that the time components of the problem are quantifiable and a base set is known.

For example, the statement, "John stopped on the way home to pick up some milk," could be represented using linear inequalities by defining the appropriate inequalities between the starting and ending times of the event "on the way home" and the event "pick up milk." Since the purchase of milk was performed while John was performing the "on the way home event," the implicit assumption is that the purchase of the milk occurred after John started on his way home and before he arrived home.

\[
\begin{align*}
\text{start}(\text{wayHome}) & \leq \text{start}(\text{buyMilk}) \\
\text{end}(\text{buyMilk}) & \leq \text{end}(\text{wayHome})
\end{align*}
\]

This illustrates how relationships between start and end points of an event can be used to represent qualitative relationships between the events bounded by the points. However, this mechanism is still limited to instances where the points bounding the intervals are quantified.
Quantum Temporal Units

As previously mentioned in the historical perspective. Time can take on a "quantum" view. Interval become the continuous interpretation of time while points form the individual quanta of time. This perception is analogous to Ladkin's [Lad86b] notion of time units. Ladkin proposes that units of time be organized into a list of order points. Thus the list [1952 9 21] forms a point or quantum of time whose scope is the day of September 21, 1952. Thus, the date can be viewed as a point or, when referring to the whole day, an interval.

This viewpoint has some interesting implications because it implicitly introduces the notion of a frame of reference for interpreting time. In the scope of a day's affairs, the unit (1952 9 21) represents an interval over which a number of events could and do occur. Taken in the context of geological time, however, the single date amounts to no more than a point (or less some might say) on earth's geological time line.

Consequently, the limit at which the unit effectively approaches zero and takes on the properties of a point is dependent on the scope and extent of the intervals in which it is viewed.

Interval-Based Logics

Interval-based approaches to temporal reasoning generally rely on a reified logic to represent the existence or truth of a fact to hold over a given temporal interval. Although the notion of an interval as a concept for representing events and actions can be traced back to Montague [Dow79] and his semantic grammar, most of the research in interval-based temporal reasoning over the past decade can be linked to Allen's initial paper [All83] on the subject.
Interval Algebra

Allen presents a temporal logic for reasoning over temporal intervals. This logic consists of thirteen relationships and is built upon seven basic relationships, shown in Table 1. The six relationships, their inverses, plus the identity relationship (=) cover the spectrum of relationships that may exist between temporal intervals.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Symbol</th>
<th>Inverse</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X precedes Y</td>
<td>&lt;</td>
<td>&gt;</td>
<td>The interval X is before the interval Y and they do not overlap in any way.</td>
</tr>
<tr>
<td>X equal y</td>
<td>=</td>
<td>=</td>
<td>Both X and Y are equivalent.</td>
</tr>
<tr>
<td>X meets Y</td>
<td>m</td>
<td>mi</td>
<td>Interval X is before interval Y, but there is no interval between them (i.e., X ends as Y starts).</td>
</tr>
<tr>
<td>X overlaps Y</td>
<td>o</td>
<td>oi</td>
<td>X starts before Y starts and ends before Y ends.</td>
</tr>
<tr>
<td>X during Y</td>
<td>d</td>
<td>di</td>
<td>The interval X is fully contained within the interval Y.</td>
</tr>
<tr>
<td>X starts y</td>
<td>s</td>
<td>si</td>
<td>Both X and Y have the same beginning but X ends before Y.</td>
</tr>
<tr>
<td>X finishes Y</td>
<td>f</td>
<td>fi</td>
<td>Both X and Y share the same end but X begins after Y begins.</td>
</tr>
</tbody>
</table>

These intervals are represented pictorially in Figure 1.
Figure 1. Seven Temporal Relationships

Allen’s justification for an interval-based approach is built on the argument that it more closely matches how we reason about time. Thus, restricting time to absolute points overly limits the class of problems which the system is capable of addressing. This interval-based approach has been the catalyst for many other research efforts which view temporal relationships in the same light.
Table 2. The Transitivity Table for Interval-Based Relationships

<table>
<thead>
<tr>
<th>A-B</th>
<th>B&lt;C</th>
<th>BmC</th>
<th>BoC</th>
<th>BsoC</th>
<th>BdC</th>
<th>BfC</th>
<th>B&gt;&lt;C</th>
<th>BmiC</th>
<th>BoiC</th>
<th>BsiC</th>
<th>BdiC</th>
<th>BfiC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&gt;B</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>o</td>
<td>m</td>
<td>d</td>
<td>s</td>
<td>&lt;</td>
<td>o</td>
</tr>
<tr>
<td>AmB</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>m</td>
<td>o</td>
<td>d</td>
<td>s</td>
<td>o</td>
<td>d</td>
<td>s</td>
<td>&gt;</td>
<td>o</td>
</tr>
<tr>
<td>AoB</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>m</td>
<td>o</td>
<td>o</td>
<td>d</td>
<td>s</td>
<td>o</td>
<td>i</td>
<td>d</td>
<td>i</td>
</tr>
<tr>
<td>AsB</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>m</td>
<td>o</td>
<td>s</td>
<td>d</td>
<td>d</td>
<td>&gt;</td>
<td>mi</td>
<td>o</td>
<td>f</td>
</tr>
<tr>
<td>AdB</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>m</td>
<td>o</td>
<td>s</td>
<td>d</td>
<td>d</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>AfB</td>
<td>&lt;</td>
<td>m</td>
<td>o</td>
<td>d</td>
<td>s</td>
<td>d</td>
<td>d</td>
<td>f</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>A&gt;B</td>
<td>all</td>
<td>&gt;</td>
<td>o</td>
<td>i</td>
<td>m</td>
<td>d</td>
<td>f</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>AmiB</td>
<td>&lt;</td>
<td>o</td>
<td>m</td>
<td>d</td>
<td>f</td>
<td>si</td>
<td>o</td>
<td>i</td>
<td>d</td>
<td>f</td>
<td>d</td>
<td>f</td>
</tr>
<tr>
<td>AoB</td>
<td>&lt;</td>
<td>o</td>
<td>m</td>
<td>d</td>
<td>f</td>
<td>f</td>
<td>m</td>
<td>o</td>
<td>d</td>
<td>f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AsiB</td>
<td>&lt;</td>
<td>o</td>
<td>m</td>
<td>d</td>
<td>f</td>
<td>f</td>
<td>s</td>
<td>si</td>
<td>o</td>
<td>d</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>AdB</td>
<td>&lt;</td>
<td>o</td>
<td>m</td>
<td>d</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>d</td>
<td>o</td>
<td>i</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>AfB</td>
<td>&lt;</td>
<td>m</td>
<td>o</td>
<td>o</td>
<td>d</td>
<td>s</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 presents the transitivity relationships based on Allen's interval relationships. The equal (=) relationship is omitted because does not have any transitive implications in terms of asserting new relationships. Transitive relationships are located in the table by matching the relation type between the first and second interval to the those shown in the leftmost column. This selects the entire row for that entry.
Then the relationship between the second and third interval is matched along the top row. This selects the column indicated by the matching entry. The intersection of the row and the column yields the set of transitive intervals for the two relationship types identified. For example, if $A$ *overlaps* $B$ and $B$ *is finished by* $C$ then the row selected will be the third row $A \circ B$. The column selected will be the rightmost, $B \bar{f} C$. The transitive relations consists of the set located at the intersection of the row and the column, $(<, o, m)$.

An illustration of the temporal transitivity is shown in Figure 2. Interval $A$ *overlaps* $B$, $(A \circ B)$, and interval $B$ *precedes* $C$, $(B < C)$. Thus, $A$ *precedes* $C$ as well, $(A p C)$. This can be looked up using the transitivity table previously mentioned. Thus, a straightforward transitive table can be used to define the transitive relationships between three or more intervals.

![Diagram of transitive relationships](image)

**Figure 2. Simple Transitive Relationships**

Problems begin to arise, however, when the relationship is not as clear as illustrated in Figure 3. For example, assume the assertions that interval $A$ *precedes* $B$, $(A p B)$, and $B$ *during* $C$, $(B d C)$. The resultant computed transitive relationship between $A$ and $C$ is, in fact, a set of five possible relationships, *precedes*, *meets*, *overlaps*, *starts*, and *during*. How this ambiguity can arise is illustrated in Figure 3.
The interval relationships provide logical and understandable 
descriptions of relationships between two intervals. However, the nature of the 
interval calculus is such that the exact position in time of the intervals is not 
always known. This gives rise to ambiguous relationships when attempting to 
compute the transitive closure of a set of intervals. As shown in Figure 3, there 
are five possible placements for the interval A in which the assertion that A 
precedes B is true with five different consequences in terms of the transitive 
computation of the relationship between A and C.

The presence of ambiguous relationships within a set of intervals is a 
common occurrence when attempting to describe any moderately complex set of 
interactions between entities. This gives rise to a large amount of overhead in 
computing transitive closures because the algorithm must take into account the 
fact that the transitive computation is not a closed world, i.e. in traditional 
approaches to transitive closure computation only a single relation is represented 
and the transitive computation is simply an efficient mechanism for representing 
and computing the propagation of the relation throughout all possible nodes.

The problem quickly becomes intractable for reasonably large sets of 
intervals with moderate ambiguity. The algorithm must, in essence, follow 

![Figure 3. Ambiguous Transitive Relationships](image-url)
divergent forks whenever it encounters an ambiguous condition. Thus, for the above example, if a fourth interval, D, is added to the set and a relation is asserted between C and D, the computation of the transitive closure from A to D via C must follow five different possibilities. Each of these possibilities may have multiple interpretations which must then be merged together to form the set of possible relations between A and D.

Although earlier papers [All83a and All83b] present Allen's interval-based theory of time, "Towards a General Theory of Action and Time" [All84] is his first comprehensive treatment of the theory. Here he presents the theory based on a typed, first-order predicate calculus.

Allen and Hayes [All85] provide a stepwise development of the thirteen temporal relationships based on the single relationship "MEETS." This paper also presented the concepts of "instantaneous" events which were called "Nests." These are described as the "beginning or ending" of intervals. The important aspect of nests are that they have a very small duration and, thus, take on the properties of points within the temporal calculus. This concept is illustrated in Figure 4.

![Figure 4. Nests Showing Bounds of Interval](image)

The essential quality of nests is that they have the property of being totally ordered, thus (from [All85]), for any two nests N and M, either $N<M$, $M<N$, or
These nests form an initial starting point for the development of the theory presented here.

Allen [All91] also proposes that there are multiple approaches to representing time. These alternatives are applied based on factors concerning the certainty with which one can assert temporal facts about the system being represented. Issues regarding simultaneity, time stamp ordering, event overlap, and event duration are discussed relative to their effect on the choice of representation.

When reasoning about plans, there are tradeoffs which must be considered which address the expressiveness and need for a temporally explicit representation for a given domain. Allen [All89] states that much recent work makes simplifying assumptions about time in favor of examining other issues in detail. Some detail is given to several influential assumptions and how they affect the range of problems that can be solved. Specific attention is given to the problem of reasoning about simultaneous events and how many proposed solutions with limited temporal models are not generalizable to realistic domains.

A mechanism is presented in [Dea84] which addresses planning for robotics problem solving handling queries of temporal knowledge involving uncertainty. Specific issues covered include the computational complexity inherent in reasoning about temporal actions.

The assumptions that must be made, by human or machine, regarding temporal information must often be made under uncertainty or incomplete information. Often these plans and assertions must be modified [Dea87] as axioms previously held to be true are, in fact, found to be false or have changed.
Other problems surface when attempting to reason over time. Intervals or processes can occur over extended period of time and may be intermittent in nature. Cyc [Len90] represents events in the same general manner as any other tangible object within the representation system. By providing a declarative structure for events, not only can the effect of the event be propagated throughout the knowledge base but also the relationship between the event and other entities in the knowledge base can be modeled. This concept was also part of the representation system proposed in [Kov86].

Allen [All84] defines a logic for temporal intervals using the symbols shown in Figure 5. This interval-based logic provides a flexible method for capturing the relative time references in natural language understanding and other problems where absolute time is not precisely known.

\[
\begin{array}{c}
\& \text{conjunction} \\
\lor \text{disjunction} \\
\neg \text{nagation} \\
\Rightarrow \text{implication} \\
\Leftrightarrow \text{equivalence} \\
\forall \text{universal quantifier} \\
\exists \text{existential quantifier} \\
\exists! \text{existence of a unique object}
\end{array}
\]

\textbf{Figure 5. Operators for Allen's Interval-based Temporal Logic}

The interval logic employs a small number of predicates. The essential concepts of the logic is based on the notion that some fact (i.e., property) is true
for a given interval of time. Thus, the property "holds" for a given time interval. This is stated as:

\[ \text{HOLDS}(p, t) \]

Allen then develops the thirteen basic predicates of the interval calculus, presented in Table 1. From this basic set of interval and predicate descriptions, Allen describes the concept of an occurrence. An occurrence becomes a declarative object within the temporal logic. Thus, events are part of the logic just as are properties. This can be applied to represent the occurrence of specific events as being performed over a given time interval by representing the events themselves as objects which are related over functions within the calculus. Thus,

\[ \text{OCCUR}(\text{ChangePos}(\text{Ball}_1, \text{Pos}_1, \text{Pos}_2), T_{100}) \]

provides a representation of the fact that over time interval \( T_{100} \) the event \( \text{CHANGE-POS} \) occurred on the ball object. This can be extended by defining the implications that must be true prior to the ball actually moving. These are expressed below.

\[ \text{OCCUR}(\text{ChangePos}(\text{object}, \text{source}, \text{goal}), t) \Rightarrow \exists t_1, t_2. \]
\[ \text{MEETS}(t_1, t) \& \text{MEETS}(t, t_2) \& \]
\[ \text{HOLDS}(\text{at}(\text{obj}, \text{source}), t_1) \& \]
\[ \text{HOLDS}(\text{at}(\text{object}, \text{goal}), t_2) \]

Thus for some change in position, there exists two temporal intervals, \( t_1 \) and \( t_2 \), in which some other interval, \( t, \text{MEETS} \) interval \( t_2 \) and interval \( t_1 \text{MEETS} t \). For interval \( t_1 \), the proposition that the ball is at the source holds, while for interval \( t_2 \), the proposition that the ball is at the goal holds.
Subsets of the Interval Algebra

The full temporal algebra proposed by Allen has been shown to be \( \mathcal{NP} \)-complete with the average performance complexity to be polynomial, \( O(n^3) \) and, consequently, computationally expensive to implement. This has led to a number of research efforts which seek to reduce the complexity of the algebra by addressing a subset of the full algebra. The goal has been to maintain a reasonably robust temporal representation while achieving better than \( O(n^3) \) complexity.

Several research efforts have developed in response to or parallel with the realization of the complexity of the full algebra. In particular, Vilain's work which was discussed in the previous section, was in parallel to Allen's and it was Vilain and Kautz who identified the complexity of the interval algebra. Other representations, such as [Dec92], have been developed as a means for reducing the complexity in order to achieve polynomial performance. These efforts, however, resulted in different representation forms than the interval-based concept in order to achieve less complexity.

In [Van90] a subset of the temporal algebra is proposed which maintains the interval as the basic representational unit. It does, however, use a directed graph to represent the collection of interval relationships. Thus, the nodes of the graph represent the intervals and the arcs represent the relationships. This is in contrast to other constraint-based approaches in which the nodes represent states (or points) and the arcs represent durations (i.e. intervals).
Temporal Constraints

Constraint-based temporal reasoning system view the world as a series of states which are connected via temporal constraints forming a directed graph of state changes over time.

Time Maps and Temporal Databases

Dean and McDermott [Dea87] extend the concept of situational calculus to what they call a temporal database. The database itself is called a time map with the collection of functions for interacting with the database, maintaining consistency, and forward inferencing called a time map management (TMM) system.

Essentially, a time map is a directed graph whose vertices represent points or instants of time denoting the beginning and ending of events. The directed arcs connecting the time points represent the performance of a particular action or event. Each of the edges are labeled using a pair of points representing the upper and lower bounds of the temporal distance separating the two points in time. An example is shown below in Figure 6.

![Figure 6. Labeled Edges Forming Constraints Between Temporal Points in a Time Map](image-url)
Thus for any path in the network, we can compute the temporal distance. Typically, only those paths of the least upper bound or the greatest lower bound are of interest. In this representation approach, a set of constraints, $c_1, c_2, \ldots, c_n$, is consistent if the sum of the lower bounds is less than or equal to the sum of the upper bounds for all paths through the network, $p_0 c_1 p_1 \ldots c_n p_n$. This consistency is, in effect, simply stating that all of the time points in the time map are in an ascending order for both the greatest lower bound paths and the least upper bound paths.

![Diagram](image)

**Figure 7. Database Persistence versus Time Map Persistence**

The edges in the time map represent *intervals* for which the beginning and ending points are represented by the constraint pair specified. Within the time map, the specific occurrence of an event is labeled with a *token*. Thus a token may be referred to as an instantiation of an event and may be labeled as a tuple...
of symbols representing the event (e.g. \texttt{(state-of switch-12 on)}). The instantiation of an event, such as the switch in the previous sentence, result in a state or condition which is \textit{persistent}. Persistence in a time map is different from usual database persistence. In a database, once a fact is asserted, it remains true until explicitly overwritten by another transaction. Persistence in the time map provide a scope of persistence over which the fact remains true.

Therefore, within the time map database, it is possible for a fact to be retracted without being explicitly overwritten by another. Thus, the value of an attribute of an object, as shown in Figure 7 for example, may move from a known state to an unknown state. The time map, therefore must keep track of the durations, either known or assumed, of the various actions within the temporal constraints.

This leads to the need to support the identification and resolution of conflicts, temporal projections, non-monotonicity, and an understanding of the events and their durations.

\textbf{Temporal Constraint Satisfaction}

Constraint satisfaction problems deal with the expression of relationships between object properties in a mathematical fashion such that, updates or inequalities defined between items will be automatically calculated and updated as dependent information is updated. A common example is the expression of the relationship between Fahrenheit and Celsius temperature scales as a constraint expression. This is illustrated in Figure 8 below. Essentially a data flow machine is expressed and constructed via the constraint equation which defines the dependencies between the expression of a temperature in one scale and its equivalent temperature in the other scale.
When either value is changed, the other is updated through the constraint equation such that it reflects the appropriate temperature in the other scale.

A similar approach has been applied to the expression of temporal constraints [Dec91]. A Temporal Constraint Satisfaction Problem (TCSP) is defined as a specific class of the more general Constraint Satisfaction Problem (CSP). Essentially, a constraint graph is developed for a set of temporal references. The nodes of the graph represent time points in which specific actions or events have been completed. The edges of the graph represent the existence of constraints between the nodes and are labeled with pairs of time references which denote the lower and upper bounds of the constraint. This is similar to the basic constraints developed by Dean and McDermott [Dea87]. For example, take the following description:

John left for the airport at 7:00 AM. It takes between 45 minutes to an hour to get to the airport, depending on traffic.

This is represented by a simple constraint between two nodes shown in Figure 9.
Complex sets of temporal constraints can be represented as an interconnected set of nodes and arcs. Mapping this formalism to an interval-based approach can be accomplished by letting the arcs represent the intervals and the nodes represent the end points of the intervals. Then the labeling becomes an expression of the approximate duration of the interval.

However, the interval-based relationships cannot be encoded directly as binary constraints [Dec91] but rather as 4-ary constraints where the pairs represent the beginning and end of an interval. These 4-ary constraints can be decomposed into sets of binary constraints to allow the standard Temporal Constraint Satisfaction Problem algorithms to be applied. Then the intervals can be reassembled through a union operation.

**An Example**

To illustrate the strengths and weaknesses of each of the representation methods, a single scenario will be described. Then a sketch of the representation of the scenario will be developed for each of the approaches. The intent is to provide an overview of the structures and components of the temporal representations for the scenario. This will provide some insight into the similarities and differences of each of the approaches.

Consider the following scenario:

**Actors:** John, Peter, Joan, and Lois.
Scene: John and Peter are at home. Joan is at work. Lois' initial state is unknown.

Action:

1) There was no milk so John went to the store to buy some. John went to the supermarket which usually takes 20 to 30 minutes.

2) Shortly after John left, Peter also noticed there was no milk. Not knowing that John had left to get some milk, he too went to the store to buy some. John returned home with his milk prior to Peter's return.

3) Joan, meanwhile, was on her way to John and Peter's house from work and arrived at the same time that Peter returned from purchasing milk.

4) Unfortunately, Joan had stopped to buy some milk on the way also.

5) While Peter was getting milk, Lois entered the store to pick up some ice.

6) She happened to see Joan entering the store as she did.

The next three subsections sketch an approach to representing the above example in each of the three major temporal reasoning approaches, 1) point, 2) constraint-based systems, and 3) interval. Some assumptions will be made in translating the natural language description of the event into the formalism being presented. The assumptions made correspond to how we might disambiguate a temporal reference in building a coherent cognitive model of a set of temporal relations.

Point Representation

In a point-based logic, each state of the entities in the story is associated with a time point. The ordered set of time points represents the scenario in the above example.

T0: John and Peter are at home, Lois and Joan's whereabouts are unknown although later in the story we find that Joan was probably at work.
1) There was no milk so John went to the store to buy some. John went to the supermarket which usually takes 20 to 30 minutes.

This statement results in the creation of three temporal points, T1, T2, and T3. T1 represents the state of John having left for the store, T2 represents John having purchased the milk, and T3 is John returned home. Each of these points have a constraint providing ordering information. These three points are ordered and connected to the temporal origin point T0 in the constraint graph.

Figure 10. Initial Set of Temporal Points

2) Shortly after John left, Peter also noticed there was no milk. Not knowing that John had left to get some milk, he too went to the store to buy some. John returned home with his milk prior to Peter's return.

This statement results in three new time points, T4 is Peter leaving for the store, T5 is Peter's purchase of the milk, and T6 is Peter's return home. Each of these three points are linearly ordered by arcs connecting the nodes. In addition, two constraints are represented by arcs from T1 to T4 (Peter leaving after John) and a constraint between T3 and T6 (John returns home before Peter).

Figure 11. Addition of Second Event Sequence Points
3) Joan, meanwhile, was on her way to John and Peter's house from work and arrived at the same time that Peter returned from purchasing milk.

This statement results in the instantiation of temporal point T7. Since the sentence states that both Peter and Joan arrived home at the same time, the constraint between T6 (Peter arriving home) and T7 is one of equivalence. Either point can be used in calculation and, in fact, both events would be collapsed to a single point within an implementation.

![Figure 12. Additional of Temporal Point and Equivalence Relation](image)

4) Unfortunately, Joan had stopped to buy some milk on the way also.

Now it's stated that Joan stopped on the way home and purchased some milk. This establishes a temporal point, T8, which is constrained to be prior to her arrival home, T7. This is represented by the arc from T8 to T7. However, since there is no explicit reference to anchor the start of T8. Consequently, it is anchored to T0.
5) While Peter was getting milk, Lois entered the store to pick up some ice.

This statement constructs a new temporal point T9. This sentence raises some representational problems. The temporal reference within the statement is one of containment, i.e. "While Peter was getting milk" and, consequently, limits the assertions that can be made. All we can deduce is that sometime after Peter left for the store and before he returned Lois also went to the store to buy ice. T5 represents the state that Peter has purchased the milk. This is insufficient to resolve the placement of T9 to before or after T5. This is represented by the dashed, bi-directional arc between T5 and T9.

Figure 13. Insertion of Temporal Point Preceding T7
6) *She happened to see Joan entering the store as she did.*

This last statement may be incorporated into the time map in several ways. One is to instantiate a new point $T_{10}$ to represent Joan entering the store, insert it between $T_0$ and $T_8$ and assert an equivalence relation between $T_9$ and $T_{10}$ (and eventually collapse the two points to a single node). This is illustrated below.

Figure 14. Additional Temporal Point

Figure 15. Inserting Point and Equivalence Relation
Finally, this section builds a constraint-based representation of the same set of events. The constraint model is based on the constraint graphs of Dechter and Pearl.

There was no milk so John went to the store to buy some. John went to the supermarket which usually takes 20 to 30 minutes.

This statement establishes an initial temporal constraint. Two nodes, corresponding to John leaving for the milk and returning are connected via an

---

**Figure 16. Inserting an Ordering Constraint Between T9 and T8**

As the example illustrates, even a relatively straightforward natural language discourse can result in a complex set of interrelationships and set of time points.

**Constraint Representation**

Finally, this section builds a constraint-based representation of the same set of events. The constraint model is based on the constraint graphs of Dechter and Pearl.

1) *There was no milk so John went to the store to buy some. John went to the supermarket which usually takes 20 to 30 minutes.*
arc which is label to show the minimum and maximum times associated with the event.

![Figure 17. Initial Temporal Constraint]

Next, we add the constraint for Peter getting milk.

2) Shortly after John left, Peter also noticed there was no milk. Not knowing that John had left to get some milk, he too went to the store to buy some. John returned home with his milk prior to Peter’s return.

This constraint specifies that Peter left after John and returned after John. Therefore, assuming the same maximum and minimum applies to Peter getting milk two new nodes are created, 2 and 3, and are connected via an arc labeled to show the minimum and maximum temporal extents. Also, since Peter left after John, an additional constraint is expressed via an arc between node 0 and node 2.

![Figure 18. Incorporating Second Event Into Constraint Network]
The constraint arc between node 0 and node 2 however, is initially unlabeled. This is because there is no specification of how long it was that Peter left after John. Upon some inspection, however, it can be deduced that, in order to maintain consistency, we can specify an initial maximum of 20. Thus the arc could be labeled \([0, 20]\). However, there is no way of knowing that, in fact, John took 30 minutes and Peter left 25 minutes after John did. Thus, the effectiveness of the constraint network is diminished. Furthermore, the assignment of \([0,20]\) is made based upon observation and deductive reasoning. Expecting a temporal reasoning mechanism to embody such deductive capabilities is not realistic.

3) Joan, meanwhile, was on her way to John and Peter's house from work and arrived at the same time that Peter returned from purchasing milk.

This statement creates another node, 4, which is connected to node 3. The arc represents Joan getting some milk as well and arriving home at the same time as Peter. However, as in the previous statement, there is no mention of time duration. Thus, the arc from 4 to 3 is unlabeled.

Figure 19. Merging an Event Arc with an Existing Event Node
Thus far we have a network of five nodes and four arcs, only two of which have been labeled with constraints identifying the minimum and maximum times for the action represented by the arc.

4) Unfortunately, Joan had stopped to buy some milk on the way also.

This statement presents a perturbation in the existing network. Instead of a simple additive operation, the existing network, as with the point-based representation must be altered. A new node, Node 5, must be created. The arc from node 4 to 3 must be broken and a constraint from 4 to 5 and between 5 and 3 must be asserted.

Figure 20. Inserting a Node into the Temporal Constraint Network

5) While Peter was getting milk, Lois entered the store to pick up some ice.

This statement adds a new node and two constraints to the network. Again, since no quantitative limits of durations were provided, only a qualitative
ordering can be asserted showing that node 6 occurred sometime between node 2 and node 3.

![Diagram showing node relationships and timing]

**Figure 21. Insertion of an Activity Occurring During Another**

6) *She happened to see Joan entering the store as she did.*

This sentence expresses a further constraint between the existing nodes effectively limiting the occurrence of node 5 to the same limits as node 6. Therefore, node 5 now has the same qualitative constraint placing it sometime between nodes 2 and 3.
At this point, although some limits have been defined for the network, there is relatively little information provided which would allow the assertion or inference of relationships between the action of John going to the store (Node 0 Node 1) and Nodes 5 or 6.

**Interval Representation**

Due to the complexity involved and the intractability of using a transitive lookup table when ambiguity is introduced, a small problem consisting of only five intervals will be described. I will use two columns to keep track of the assertions. One will be used to show the primitive assertions and the other will show the computed assertions based on the transitivity table.

1) *There was no milk so John went to the store to buy some. John went to the supermarket which usually takes 20 to 30 minutes.*
This action can be represented as a single temporal interval. Although this interval may be decomposed into smaller sub-intervals (i.e. to represent John arriving at the store, purchasing the milk and other more primitive actions) for the present a single interval, A, will suffice to represent the action of John going to the store for some milk.

2) Shortly after John left, Peter also noticed there was no milk. Not knowing that John had left to get some milk, he too went to the store to buy some. John returned home with his milk prior to Peter's return.

Interval B represents the action of Peter going to the store. Since the sentence states that Peter left after John and returned after John, we can assert an overlap relation between A and B.

3) Joan, meanwhile, was on her way to John and Peter's house from work and arrived at the same time that Peter returned from purchasing milk.

Now things begin to get a little more complex. Statement 3 defines a third interval, C, which represents Joan going home and ends at the same time as B. However, it does not state whether or not Interval B's duration is longer, shorter, or the same. Thus, we must account for three possible relationships between B and C, finished-by (fi), same (=), and finishes (f).
Using the transitivity table, we see that there are three possible relationships that may exist between A and C, specifically, precedes, meets, and overlaps. This arises from the ambiguity that we do not know the exact duration of either A or C. Thus far, we have two primitive assertions, one of which is ambiguous, BC, and five possible transitive assertions between A and C.

<table>
<thead>
<tr>
<th>Table 3. Example Temporal Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primitive Assertions</strong></td>
</tr>
<tr>
<td>A o B</td>
</tr>
<tr>
<td>B (f, =, f) C</td>
</tr>
</tbody>
</table>

Thus far, we have four asserted relations, of which three are a result of an ambiguity in the specification and five transitive assertions resulting from applying the transitivity table to identify potential relationships between A and C.

4) Unfortunately, Joan had stopped to buy some milk on the way also.

This sentence states that sometime during Joan's trip home, she stopped and picked up some milk. This results in a fourth interval D and asserts the fact that D is during C, C \( \cap \) D.
Figure 25. Adding a Fourth Interval During an Ambiguous Interval

Now to compute the transitive closure from A to D, we have five possible paths based on the relationship that exists between A and C shown above.

<table>
<thead>
<tr>
<th>Primitive Assertions</th>
<th>Transitive Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A o B</td>
<td>A (&lt;, m, o, s, d) C</td>
</tr>
<tr>
<td>B (fi, f, =) C</td>
<td>A (&lt;, m, di, oi, fi, &gt;, mi, o) D</td>
</tr>
<tr>
<td>D d C</td>
<td>B (si, oi, &gt;, c) D</td>
</tr>
</tbody>
</table>

At this point a large number of relationships, asserted and transitive, are ambiguous. However, it should be noted that the ambiguity is confined to unknown ordering between start and end points of the intervals.

5) While Peter was getting milk, Lois entered the store to pick up some ice.

Now add E and assert that E starts after B's start but before B's completion. Because it doesn't state when Lois leaves the store, the interval could extend towards the right beyond the current set of intervals. This could result in any of three relationships, di, fi, or o, based on the description of the relationship in the text.
Computing the transitive closure from A, C, and D to E yields the set of relation shown in Table 5.

**Table 5. Fifth Interval Added to Problem**

<table>
<thead>
<tr>
<th>Primitive Assertions</th>
<th>Transitive Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A o B</td>
<td>A (&lt;, m, o, s, d) C</td>
</tr>
<tr>
<td>B (f, f, =) C</td>
<td>A (&lt;, m, d, o, f, &gt;, m, o) D</td>
</tr>
<tr>
<td>D d C</td>
<td>B (s, o, &gt;, c) D</td>
</tr>
<tr>
<td>B (d, f, o) E</td>
<td>C (d, s, o, c, f, =) E</td>
</tr>
<tr>
<td></td>
<td>D (&lt;, m, o, s, d, s, d, =) E</td>
</tr>
</tbody>
</table>

6) *She happened to see Joan entering the store as she did.*

This statement results in the assertion that D and E start at the same time. This reduces some of the ambiguity by limiting the relations that other intervals may have with E.
Keeping the set of relations constant but adding one more primitive assertion, D starts E, results in a number of transitive assertions that are no longer valid because they are in conflict either with the primitive assertion or in conflict with the transitive computations resulting from the primitive assertion. The first relations to be retracted are all of the transitive assertions between D and E since D and E now have a primitive assertion. Secondly, since D starts E and D is during C, the transitive assertions C during E and C starts E are no longer valid either and must be retracted. This is reflected in Table 6.

<table>
<thead>
<tr>
<th>Primitive Assertions</th>
<th>Transitive Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A o B</td>
<td>A (&lt;, m, o, s, d) C</td>
</tr>
<tr>
<td>B (fi, f, =) C</td>
<td>A (&lt;, m, di, oi, fi, &gt;, mi, o) D</td>
</tr>
<tr>
<td>D d C</td>
<td>B (si, oi, &gt;, c) D</td>
</tr>
<tr>
<td>B (di, fi, o) E</td>
<td>C (s, o, c, fi) E</td>
</tr>
<tr>
<td>D (s, =, si) E</td>
<td></td>
</tr>
</tbody>
</table>

**Summary**

It has been shown [Vil86] that the interval representation approach of Allen [All83] is in the order of O(n^3) and therefore a computationally expensive problem. For this reason recent efforts have focused on restricted views [Vil86], modified approaches to intervals [Lad86], or alternative views [Dec92] using Temporal Constraint Networks. While these efforts have had some success in improving computational efficiency, they do not capture the full expressiveness of the interval algebra.

Another problem with the algorithm used by Allen is the inability to ensure consistency between more than three intervals at a time. If an
inconsistency is encountered when calculating the transitive closure then the system must be capable of backtracking to return the system to a consistent state.

In the previous sections we have covered a wide range of approaches to representing temporal information and reasoning over that information. All systems provide some representation mechanism for intervals, points, and relationships.

Situational calculus [McC63] views time as a sequence of discrete points which represent time points in which certain facts are true. The weakness of the situational calculus approach is the lack of support for representing actions which occur over time or intermittently.

Time maps [Dea87] use a point based representation with the edges connecting the points representing intervals of time over which activities may occur. These are labeled with upper and lower bounds of time information. The primary focus of Dean's work was the application of temporal reasoning to database assertions, although it can be argued that any temporal reasoning system is concerned with the truth or persistence of a fact over time.

Point-based algebra [Vil86] are essentially a hybrid of the point-based time maps of [Dea87] and the interval based approach of [All83]. Temporal events are represented by bounding points defining the start and stop of the event. These bounding points are also similar to the nests defined by Allen [All85] and Hayes which provide a point-based capability within the interval algebra.
CHAPTER 3
A BIT-MAPPED TEMPORAL ALGEBRA

The whole of science is nothing more than a refinement of everyday thinking.
-- Albert Einstein, Physics and Reality, 1936

This chapter lays the foundation for the development of a computational algebra for temporal intervals based on bit-mapped relationships for the representation and a binary algebra for computing transitive relationships. The origins of the algebra are presented and then, through a series of stepwise transformations, a computational algebra is developed. The algebra developed in this chapter is formed as a hybrid of the interval-based approach and time-point algebra. Specifically, the representational expressiveness of interval-based systems is preserved, particularly the area of relative temporal references, while providing the computational efficiency of a time-point algebra.

The prime goal throughout the formulation of the algebra is expressiveness coupled with computational efficiency. The origins of the efficiency stem from the application of partially ordered sets (posets) to the problem of organizing and representing class hierarchies in an object-oriented
programming language. Posets provide a well structured framework for organizing and labeling classes. This method provides a bit-encoding mechanism which forms the basis of the efficient calculations and performance.

The balance of the chapter extends this basic concept to develop a comprehensive approach to the representation of temporal intervals and their relationships.

**Defining the Sub-Algebra**

The first step towards developing a representation and computational method is to define and partition the problem. A number of researchers have developed representations based on subsets of Allen’s interval algebra to avoid the computational complexity of the full algebra. Current work by [Alu94] employ a timed-logic concept which provides a representation tailored more towards real-time systems and event processing. Vilain and Kautz [Vil86] collapse the interval algebra into a point-based method in which the arcs connecting the points represent the passage of time. Constraint-based systems such as [Dec92] and [Dea87] 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. However, as shown in the previous chapter, both these approaches sacrifice some of the expressiveness of the full interval algebra in order to achieve computational efficiency.

The sequence graphs of Dorn [Dor92] preserve the richness of the temporal algebra but relies on a graph reduction algorithm to minimize the

\(^2\) These are similar to the states in situational calculus.
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.

**Ambiguity Versus Hypothesis**

Upon closer inspection, a great deal of the intractability of Allen's interval algebra results from temporally implausible conditions. For example, the statement "Vince entered the room before or after I turned out the light." expresses just such an implausibility. Letting *Vince* correspond to the event of Vince entering the room and *Light* represent the actions of turning out the light, the temporal relations between Vince and Light would be *Vince* (<, >) *Light*. This is perfectly legal in the full algebra.

In fact, what is being said by this statement is not a temporal ambiguity but rather a statement of hypothesis. There is not and, in fact, can't be a single temporal interval that could have both the before (<) and after (>) relationship as a set of compatible choices. Therefore, what is really expressed by the above statement is that there are two potential events *Vince1* and *Vince2* where *Vince1* (<) *Light* and *Vince2* (>) *Light* and either *Vince1* or *Vince2* occurred but not both.
This concept is illustrated in Figure 28. There are two hypotheses, each asserts an event in the knowledge space. Each of these events represents a hypothetical occurrence of Vince entering the room with a different relationship to the event that turns on the light.

The premise for viewing these as two distinct hypothesis rather than one event with multiple possible relationships is because there is no commonality of temporal extent between the two. Put differently, if two relationships have no overlap then the events they relate are disjoint.

Interval Structure and Reference Frames

This section provides some initial background regarding the representational approach for the research to be performed for this dissertation.

Events, Processes, and States

Temporal understanding implicitly assumes that the agent (be it human or machine) has some knowledge and comprehension of the concepts of events,
processes, and states. The use of these terms within this research will be as follows:

An event is an occurrence of a temporal interval of some finite duration (possibly zero) which is used to represent some action. Events in this connotation can be thought of as being synonymous to Allen's intervals [All83] with one extension. The interpretation of an event may have any length duration. Thus, as the duration of an event becomes arbitrarily small, events take on the characteristics of points [Dea87] or "Nests" [All85b]. Thus, a process can be thought of as three intervals, a start and end with durations of 0, and an event for the process itself with a duration bounded by the start and end intervals.

A process is the action(s) performed resulting from the event occurrence. The distinction between a process and an event is that events are simply temporal markers, points or intervals, while processes convey actions and behaviors. The two are inextricably linked in that the start, end, and performance of a process are events. This distinction allows us to avoid the reified logic [Gal91] necessary to implement Allen's intervals. Processes can also be hierarchically decomposed using a time duration and offset [Lit94] to provide a process description of ordered intervals (i.e. events) within a plan.

State refers to the value of an attribute which describes an object within a temporal reasoning system. State changes occur according to the process which describes the state change and the event which performs the change. Thus, for the event of purchasing some milk. The state of the ownership of the milk changes from the store to the individual with the event which corresponds to the money exchange.
Bounding Intervals

When describing the temporal intervals defined by Allen, references are made to the start and end of the intervals in question. In fact, Allen describes his entire set of relationships using the concept of nests and a single relation, MEETS, which utilizes the notion of the start and end of an interval being a point.

Although Allen categorically states that time points in understanding temporal relationships is unnecessary [All83], the description of the interval relationships nonetheless reference the start and end of an interval as though it were a discrete point in time. This work is based on the relationships that exist between the start and end points of intervals. These relationships exist between bounding intervals which identify and bound the interval of interest. Thus, a bounding interval is defined below.

**Definition 1:** A bounding interval, $I_b$, for a temporal interval, $X$, is defined as an interval which either starts or finishes $X$ and is denoted by $X_s$ for the start interval and $X_e$ for the ending interval. Both $X_s$ and $X_e$ have durations, $d$, which are 0 in length.

For example, in Figure 29 below, the interval $X$ is bounded by the intervals $X_s$ and $X_e$. Every temporal interval can be viewed as having these two intervals.

![Figure 29. Bounding Intervals](image-url)
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.

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. These are illustrated in Table 7.
Table 7. Bounding Relationships for Intervals

<table>
<thead>
<tr>
<th>Relation</th>
<th>Xs→Ys</th>
<th>Xe→Ys</th>
<th>Xs→Ye</th>
<th>Xe→Ye</th>
</tr>
</thead>
<tbody>
<tr>
<td>X &lt; Y</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>X m Y</td>
<td>&lt;</td>
<td>=</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>X o Y</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>X s Y</td>
<td>=</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>X d Y</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>X f Y</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>=</td>
</tr>
<tr>
<td>X oi Y</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>X mi Y</td>
<td>&gt;</td>
<td>&gt;</td>
<td>=</td>
<td>&gt;</td>
</tr>
<tr>
<td>X &gt; Y</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>X s i Y</td>
<td>=</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>X d i Y</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>X f i Y</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>=</td>
</tr>
<tr>
<td>X = Y</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
</tbody>
</table>

The above table illustrates the finite differences between each of the interval relationships in terms of the relationships between each of the bounding intervals. The interesting phenomenon to note in inspecting the table above is that, by using the set of relationships between the bounding intervals to describe the temporal relation, each relation has a unique pattern or signature. The precedes relationship, for example, has the signature (<<<<) while during has the signature (>>>>).
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 which identifies a specific relation type.

**Interval Semantics**

The initial approach to developing the hierarchy of interval relations will be based on the semantics of each of the relationships. That is, in examining each relation's signature for finite differences, as mentioned in the previous section, each relation will be categorized based on the relationships between all possible

<table>
<thead>
<tr>
<th>Relation</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedes (&lt;)</td>
<td>&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>Meets (m)</td>
<td>&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>Overlaps (o)</td>
<td>&lt;&gt;&lt;&gt;</td>
</tr>
<tr>
<td>Starts (s)</td>
<td>=&gt;&lt;&lt;</td>
</tr>
<tr>
<td>During (d)</td>
<td>&gt;&gt;&lt;&lt;</td>
</tr>
<tr>
<td>Finishes (f)</td>
<td>&gt;&gt;&lt;=</td>
</tr>
<tr>
<td>Overlap Inverse (oi)</td>
<td>&gt;&gt;&lt;&gt;</td>
</tr>
<tr>
<td>Meets Inverse (mi)</td>
<td>&gt;&gt;&gt;&gt;</td>
</tr>
<tr>
<td>Precedes Inverse (&gt;)</td>
<td>&gt;&gt;&gt;&gt;&gt;</td>
</tr>
<tr>
<td>Starts Inverse (si)</td>
<td>=&lt;&lt;&lt;</td>
</tr>
<tr>
<td>During Inverse (di)</td>
<td>&lt;&gt;&gt;&gt;</td>
</tr>
<tr>
<td>Finishes Inverse (fi)</td>
<td>&lt;&gt;&lt;=</td>
</tr>
<tr>
<td>Equal (=)</td>
<td>====</td>
</tr>
</tbody>
</table>
paths between the bounding intervals of the two relationships as they pass through the common relation. This is illustrated in Figure 31.

![Figure 31. All Possible Paths Between Bounding Intervals](image)

In expressing the transitive relationship between the bounding interval any two intervals (A and C), passing through a common third interval (B), there are eight possible paths. These are enumerated in Table 9.

**Table 9. All Possible Paths Between Bounding Intervals**

<table>
<thead>
<tr>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_s \rightarrow B_s \rightarrow C_s$</td>
</tr>
<tr>
<td>$A_s \rightarrow B_s \rightarrow C_e$</td>
</tr>
<tr>
<td>$A_s \rightarrow B_e \rightarrow C_s$</td>
</tr>
<tr>
<td>$A_s \rightarrow B_e \rightarrow C_e$</td>
</tr>
<tr>
<td>$A_e \rightarrow B_s \rightarrow C_s$</td>
</tr>
<tr>
<td>$A_e \rightarrow B_s \rightarrow C_e$</td>
</tr>
<tr>
<td>$A_e \rightarrow B_e \rightarrow C_s$</td>
</tr>
<tr>
<td>$A_e \rightarrow B_e \rightarrow C_e$</td>
</tr>
</tbody>
</table>
The Temporal Algebra

This section defines the temporal algebra, based on the unique signatures identified earlier in this chapter. 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 the last part of the chapter.

Points and Intervals

The elements of the algebra are built around points. The algebra is defined for a set $P$ of time points, $P = \{p_1, p_2, p_3, ..., p_n\}$. Each temporal point, $p_i$, is a non-dimensional point in time that is analogous to points on a number line. Intervals are described by point-pairs, $(p_i, p_j)$, which define the interval. Thus, intervals in the algebra are described as the set of point-pairs asserted into the system, $I = \{P_1(p_a, p_b), P_2(p_c, p_d), ..., P_n(p_x, p_y)\}$.

Thus far, the algebra defined is a hybrid of the point algebra of Vilain and Kautz and Allen's interval algebra where the terms, $p_1, p_2, ..., p_n$, map to the point-based representation while the terms, $P_1, P_2, ..., P_n$, correspond to the intervals. Points form the basis of the representation and intervals are represented as point pairs. The distinction, as will be shown later, is that Vilain and Kautz’s algebra perform transitive calculations using the primitive data provided by the points of the intervals. The approach described herein, however, allows the representation of intervals as primitive entities, thus allowing a more efficient propagation algorithm.

Terms

There are four basic terms in the algebra. The set of terms, $T$, consists of four elements which indicate a discrete relationship between two end points.
\[ T = \{<,>,=,\leq,\rangle\} \]

The terms refer to a relationship between two endpoints, either start or finish, of two intervals. Thus, if \( a \) and \( b \) are endpoints of intervals \( X \) and \( Y \) then \( X_a < Y_b \) is interpreted as "endpoint \( a \) of \( X \) is less than endpoint \( b \) of \( Y \)." It should be noted that the endpoints need not be the same type (i.e. start or finish) for the relationship to be valid. The interpretations of \( <, >, \) and \( = \) are self-evident. However, the term, \( \leq,\rangle \), requires a little more explanation. It is important because it forms a key element of the approach to the algebra. In Allen's [All85] definition of his temporal algebra in terms of nests, the three relationships, \( (c, >, =) \) were used to define the algebra. The definition provided, however, did not take into account the potential for ambiguous relationships based on unknown end point relationships between the intervals. This unknown, when represented and incorporated into the algebra provides a powerful mechanism for representing a wide range of qualitative relationships while maintaining computational efficiency.

The term \( \leq,\rangle \) means that there is no definitive relationship between the two endpoints for which it is asserted. Thus it is not known whether a less than \( (<) \), greater than \( (>) \), or equal \( (=) \) relationship exists between the two endpoints. This term in the algebra is critical because it forms the basis for propagating the "memory" of an ambiguous transitive assertion to subsequent interval relations.

Based on the premise that the endpoints are dimensionless points, there can only be the four terms defined above representing the relationship between two endpoints. This is shown to be true in the following simple proof.

1. \( \text{Let } Pa \text{ and } Pb \text{ be two points on a number line.} \)
2. \( \text{Both } Pa \text{ and } Pb \text{ are dimensionless (i.e. they have zero length)} \)
3. \( \text{Assign } Pa \text{ the value } 2 \text{ on the number line.} \)
4. Assign Pb the value 1 on the number line.
5. Pb is less than (<) Pa.
6. Gradually increase the value of Pb by some discrete value.
7. Pb moves to the right (i.e. increasing value) on the number line until it has reached Pa.
8. Since both Pa and Pb are dimensionless, Pb changes from less than Pa to equal (=) Pa in as a result of a discrete change in value.
9. As Pb is incrementated once more, it changes to greater than (>) Pa.

Thus, Pb is less than Pa until a discrete change places it at the same location as Pa on the number line. This causes the relationship to change to equality. An additional discrete change moves Pb beyond Pa on the number line and changes the relationship to greater than (>). This illustrates that, for two points whose value is known, there can only be three possible relationships, less than (<), equal (=), and greater than (>).

There is one additional relationship that must be represented and included in the algebra, however. It is the unknown relationship. When either or both points in a system have an unknown value then the relationship between the two points is unknown. This is represented in the algebra by the symbol (<=>). Thus, there are four possible relationships that can be expressed between any two points. Three relationships which define a distinct relationship, less than, equal, and greater than, and a fourth, unknown, which represents ambiguous relationships between points.

Intervals

Intervals, then are represented as a set of end points and their relationships. The projection of the transitive computation over a set of intervals
can also be represented using the end-point relationships to compute the transitive closure. Let's examine these in more detail.

Below is an example of an overlaps relationship between intervals A and B. Based on the projection of the bound intervals to a time line, it can be shown that there is a unique ordering of the end points on the time line. Thus, while the exact magnitude or time point of the bounding intervals are known, an unambiguous assignment of relationships can be performed for the interval pair.

![Diagram](Image)

**Figure 32. Mapping end points to timeline for A overlaps B relationship**

As shown in Figure 32, the mapping of the endpoints of the two intervals, A and B, to the timeline result in discrete, unambiguous relationships between each of the points. These relationships correspond to the overlaps interval signature, $(<><<)$, presented earlier. This could also be represented in a matrix form as shown below in Figure 33.
The matrix representation does not present any new information but does provide a form which provides a natural lead into the interval algebra calculation of transitive closure. This form was also applied by [She92] to represent end-point relations and perform basic transitive computations. Essentially, the matrix provides a clear tabular form which presents any interval relationship in terms the relationships between each of the endpoints of the participating intervals.

The next few pages takes this concept of mapping the intervals to a timeline and the corresponding representation of the endpoint relationships within a matrix and walks through the calculation of the transitive closure calculation.

Figure 34 presents an overlaps relationship between the intervals B and C. This is mapped to timeline points, as performed earlier with A overlaps B, resulting in the same set of unambiguous endpoint relationships between the start and end points of B and C.
Figure 34. Mapping end points to timeline for B overlaps C relationship

Representing the endpoint relationships between the intervals B and C in matrix form is shown below in Figure 35. Again, the same signature as shown for A overlaps B, (<><>), is asserted for B overlaps C.

```
<table>
<thead>
<tr>
<th></th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 35. Matrix Representation of B overlaps C Relationship Signature

The next step is to compute the transitive closure between these two interval relationships. This is first performed using the graphical technique of mapping endpoints to time lines. A problem arises, however, in that an ambiguity is raised for the endpoints $A_i$ and $C_i$. 
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 $B_e$, 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, $A_e < C_s$, $A_e = C_s$, and $A_e > C_s$. 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 $A$ overlaps $B$ and $B$ overlaps $C$. These ambiguities are illustrated in Figures 36, 37, and 38.

![Diagram](image)

**Figure 36. Transitive closure of two overlaps relationships where $A_e < C_s$**

This alternative is shown in matrix form in Figure 37 below. The resultant relationship between $A$ and $C$ in this case is $A$ precedes $C$. 
In the case of $A < C$, the resultant interval relationship computed between $A$ and $C$ is precedes, $A < C$. This asserted that, since $A$ is less than $C$, there exists some discrete duration of time, $D$, where $D > 0$.

Figure 37. Matrix Representation of A precedes C Relationship

Figure 38. Transitive closure of two overlaps relationships where Ae=Cs
In Figure 38, the case where $A_e = C_e$ is illustrated. Here both endpoints $A_e$ and $C_e$ are equal or identical. This results in the ending of interval $A$ being equal to the start of interval $C$. This results in a meets relationship between $A$ and $C$, $A m C$.

Figure 39. Matrix Representation of A meets C Relationship

Figure 40. Transitive closure over overlaps relationship where $C_s < A_e$
Finally, it is possible that $C_\nu$ could be less than $A_\nu$. This possibility results in the point ordering shown above in Figure 40. By stating that the endpoint $A_\nu$ is after $C_\nu$, then an overlaps relationship exists between $A$ and $C$.

<table>
<thead>
<tr>
<th></th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 41. Matrix Representation of A overlaps C Relationship**

In summary, this section has taken the transitive closure table defined and proven by Allen and presented an alternative representation of the interval relationships using unique signatures based upon endpoint relationships. The next section takes the endpoint relationships which form the interval relation signature and presents a form representing the transitive relation computation using a directed graph notation. This presentation and representation form will be used as the basis to define the bit-mapped interval algebra developed in the subsequent sections.

**Representing Relationships via Directed Graphs**

Taking the matrix of relationships presented in the previous section, a directed graph can be constructed which depicts the relationships between the end points of intervals. This graphical representation is equivalent to the matrix form shown above and, consequently, adds no additional information. The directed graph does, however, provide a representation which illustrates the
distinction between an unambiguous and ambiguous relation. It also presents a visual equivalent of the transitive relation computation.

The next two sections will present the directed graph representation of several basic temporal relations and the representation of several transitive relation calculations.

**Directed Graph Representation of Basic Relationships**

The interval endpoints and their corresponding relationships may be represented using a directed graph where the nodes of the graph represent the individual endpoints and the directed arcs of the graph represent the temporal direction. The temporal direction is represented as a vector from earlier time to later time. So, for example, in the figure below, the relationship *A overlaps B* is represented graphically. The start of *A* is less than the start of *B* and the end of *B*. The start of *B* is less than the end of *A*. Of course, the start of an interval is always less than the end of the interval, so the start of *A* and *B* is less than the end of *A* and *B*, respectively.

![Diagram of A overlaps B relationships](image)

**Figure 42. Directed Graph Representation of A overlaps B Relationship**

Figure 43 below presents the directed graph representation of the relationship *B precedes C*. This representation is similar to the overlaps representation shown above. The singular difference is the direction of the arc
between the start of the second relation, C in this case, and the end of the first relation, B.

Figure 43. Directed Graph Representation of B Precedes C Relationship

Figure 44 shows the directed graph representation of the meets relation. The meets relation is, essentially, a union of the arcs of the precedes and overlaps relations. Thus, it has two arcs connecting the end of the first interval and the start of the second interval.

Figure 44. Directed Graph Representation of B Meets C Relationship

This set of directed graph representation of relationships also illustrates the mathematical foundation of the sub-algebra. Between any two points in the graph there are only three possible assignments of directed arcs. These three assignments plus the unknown assignment provides the basis of the four possible values that can be asserted between endpoints in a relation.
The next section takes this concept to the next logical step and uses the directed graph representation together with the union of two subgraphs to represent transitive relations.

**Representation of Transitive Relations and Ambiguity**

Transitive relations may be represented as the graphical union of the respective basic relations. This is a straightforward operation which is performed by combining the two basic graphs over common endpoints or nodes in the graphs. For example, the graphical union of A overlaps B and B overlaps C is shown below in Figure 45.

![Graphical Union of A o B and B o C Relationships](image)

**Figure 45. Graphical Union of A o B and B o C Relationships**

As can be seen from the above figure, the resultant directed graph retains the properties of the two initial directed graphs representing the base relations. This form of representing the transitive relationship provides a graphical means for identifying those transitive relationships which result in ambiguities.
The A overlaps B and B overlaps C example is just such a case where ambiguity results when the transitive relation is calculated. Since the resultant transitive relation is represented as a directed acyclic graph where the nodes in the graph represent the end points of the intervals over which the relationship is asserted and the edges of the graph correspond to a temporal direction of increasing time, then a transitive relationship is defined to be unambiguous if a path can be identified which is a spanning tree for the transitive relation graph. This is the graphical equivalent to being able to place each of the interval endpoints on a number line and maintain the temporal integrity of the relationships between the endpoints.

![Diagram](image.png)

**Figure 46. Spanning Tree Path of Transitive Relation Excluding Cs**

Figure 46 above shows the directed acyclic graph of the transitive relation graph. As can be seen, however, not all nodes are traversed by this directed acyclic graph. Thus, since a spanning tree of the graph cannot be constructed via a directed acyclic subgraph, this transitive relationship the transitive relationship represented is ambiguous.
Figure 47. Spanning Tree Path of Transitive Relation Excluding Ae

Figure 47 illustrates an alternative directed acyclic subgraph of the transitive relation which omits the end node of interval A. The alternative subgraph differences are centered around the very nodes A, and C, which form the core of the ambiguity in a transitive computation of two overlaps relations.

Figure 48. Graphical Union of A o B and B p C Relationships

Figure 48 shows the graphical union representing the transitive computation of the overlaps and precedes relationships. This transitive relation possesses the property of containing a directed acyclic subgraph which forms a
spanning tree. Thus, this transitive computation yields an unambiguous transitive relation.

![Figure 49. Spanning of A o B and B p C Relationships](image)

The only question that remains is to identify what relation is represented in the spanning tree of the transitive relation graph. This can be accomplished by inspecting the possible paths between each of the endpoints of A to each of the endpoints of C via the directed graph. Thus, $A_s$ to $C_s$ is a precedes relationship ($<$) or $A_s \rightarrow C_s$. This is illustrated by the two possible paths $A_s \rightarrow B_s \rightarrow C_s$ and $A_s \rightarrow B_s \rightarrow C_s$. The balance of the pairs can be identified using the same approach. Thus, $A_s$ to $C_s$ is a precedes relationship ($<$), $A_e$ to $C_e$ and $A_s$ to $C_e$ is the same.

Thus, the graphical representation of the sets of paths for the interval endpoints between A and C is isomorphic with the representation of the precedes relationship shown in Figure 43.
Figure 50 shows the spanning tree identified for the transitive relation graph of $A \text{ overlaps } B$ and $B \text{ meets } C$. The spanning tree identified is the same as the tree identified for the transitive relation graph of $A \text{ overlaps } B$ and $B \text{ precedes } C$ shown earlier in Figure 47. This identical subgraph denotes a resultant transitive relation of $A \text{ precedes } C$. This is, in fact, the appropriate relation resulting from the transitive closure computation.

**Encoding Relations Based on Point References**

Based on the assertion that their 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. These bit-encoded relations are summarized in Table 10.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Bit Encoded Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal (=)</td>
<td>00</td>
</tr>
<tr>
<td>Less Than (&lt;)</td>
<td>01</td>
</tr>
<tr>
<td>Greater Than (&gt;)</td>
<td>10</td>
</tr>
<tr>
<td>Unknown (&lt;=&gt;)</td>
<td>11</td>
</tr>
</tbody>
</table>
Mapping Relation Signatures to Bit Representations

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 11.

Table 11. Bit Encoded Representations of Allen's Temporal Interval Relationships

<table>
<thead>
<tr>
<th>Relation</th>
<th>Signature</th>
<th>Bit Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedes (&lt;)</td>
<td>&lt;&lt;&lt;&lt;</td>
<td>01010101</td>
</tr>
<tr>
<td>Meets (m)</td>
<td>&lt;&lt;&lt;&lt;</td>
<td>01000101</td>
</tr>
<tr>
<td>Overlaps (o)</td>
<td>&gt;&gt;&lt;&lt;</td>
<td>01100101</td>
</tr>
<tr>
<td>Starts (s)</td>
<td>=&gt;&lt;&lt;</td>
<td>00100101</td>
</tr>
<tr>
<td>During (d)</td>
<td>&gt;&gt;&gt;&lt;</td>
<td>00100101</td>
</tr>
<tr>
<td>Finishes (f)</td>
<td>&gt;&gt;&gt;=</td>
<td>01000100</td>
</tr>
<tr>
<td>Overlap Inverse (oi)</td>
<td>&gt;&gt;&lt;&gt;</td>
<td>01001101</td>
</tr>
<tr>
<td>Meets Inverse (mi)</td>
<td>&gt;&gt;&gt;&gt;</td>
<td>01010010</td>
</tr>
<tr>
<td>Precedes Inverse (&gt;)</td>
<td>&gt;&gt;&gt;&gt;&gt;</td>
<td>01010101</td>
</tr>
<tr>
<td>Starts Inverse (si)</td>
<td>&lt;=&gt;</td>
<td>00101110</td>
</tr>
<tr>
<td>During Inverse (di)</td>
<td>&lt;&lt;&lt;&lt;&gt;</td>
<td>00101110</td>
</tr>
<tr>
<td>Finishes Inverse (fi)</td>
<td>&lt;&lt;=</td>
<td>01100100</td>
</tr>
<tr>
<td>Equal (=)</td>
<td>====</td>
<td>00000000</td>
</tr>
</tbody>
</table>

These relation signatures can be presented using the matrix form used previously. Figure 51 illustrates the bit code representation used to represent an overlaps relationship between two intervals, A and C, in matrix form.
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 overlaps B and B overlaps C, the computation of the transitive closure (A,C) is as follows.

Figure 52 shows the representation of the overlaps relationship between intervals A and B. Figure 53 below shows the relationships between intervals B and C. To calculate the transitive closure, the set of paths from the start and end points of interval A to C via B must be enumerated and combined logically.
Figure 53. Matrix Representation of B overlaps C Using Bit Codes

This combination can be accomplished by following the standard ordering for matrix multiplication between two matrices of equal size. This is illustrated in Figure 54. 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 54. Matrix operation ordering for transitive closure computation
Thus, the transitive computation is performed following the standard procedure for matrix multiplication. This matrix product operation is shown below.

\[
\begin{bmatrix}
(R_{11}R_{21} + R_{12}R_{22}) & (R_{11}R_{212} + R_{12}R_{22}) \\
(R_{21}R_{21} + R_{12}R_{22}) & (R_{212} + R_{12}R_{22})
\end{bmatrix}
\]

Figure 55 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 an unknown or ambiguous relationship between the endpoints.

The next section presents the relationship matrix used to contain the set of interval relations within a given instance of a temporal system. The matrix forms the basic structure over which the transitive closure operation will be computed.
Relation Matrix

The collection of interval relationships can be thought of as a lattice of the relationships in a partially ordered set (poset) form. This is illustrated in Figure 56.

![Figure 56. Simple lattice of three intervals and their relationships](image)

The intervals are represented by the three rectangles, A, B, and C. The relationships between the intervals are represented by the three ovals with the interval pairs shown in the parentheses. For example, the interval relationship, $R$, for the interval pair $(A,B)$ is represented by following the path from A to the node $(A,B)$.

The relationship from B to A may be found by taking the inverse, $R^{-1}$, of $R(A,B)$. This holds for all the relationships represented in the lattice. The inverse of a relationship is isomorphic with the definition of the inverse relationships identified by Allen.

This lattice can be stored within an $n^2$ matrix, where $n = $ the number of intervals in the system. An example of a relation matrix is shown below. in Figure 57.
The interval matrix, as can be seen from the above example, is divided along the diagonal by a set of equal relations corresponding to the identity operation (i.e. $A = A$). The two diagonal sub-matrices are symmetric with respect to the lower diagonal matrix being an inverse of the values in the upper matrix (i.e. $R(i,j) = R'(j,i)$) where $i =$ the row and $j =$ the column of the interval relation.

In the next section, the binary operations of addition and product are defined. These operations form the basic calculation routines of the temporal algebra.

**Operations**

There are two basic operations that can be performed in the interval algebra. These are the addition ($\oplus$) and product ($\otimes$) operations. Addition in the algebra is defined as a binary inclusive or of the two bit codes. The product is defined to be the binary and of the two bit codes representing the relations. The
addition and multiplication operators shown are also consistent with symbolic versions of relation operators defined in [She92].

**Multiplication**

The relation multiplication 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>
<thead>
<tr>
<th>Term A</th>
<th>Term B</th>
<th>Product (⊗) Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 (&lt;)</td>
<td>01 (&lt;)</td>
<td>01 (&lt;)</td>
</tr>
<tr>
<td>01 (&lt;)</td>
<td>00 (=)</td>
<td>01 (&lt;)</td>
</tr>
<tr>
<td>01 (&lt;)</td>
<td>10 (&gt;)</td>
<td>11 (&lt;=&gt;)</td>
</tr>
<tr>
<td>01 (&lt;)</td>
<td>11 (&lt;=&gt;)</td>
<td>11 (&lt;=&gt;)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>10 (&gt;)</td>
<td>10 (&gt;)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>00 (=)</td>
<td>10 (&gt;)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>01 (&lt;)</td>
<td>11 (&lt;=&gt;)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>11 (&lt;=&gt;)</td>
<td>11 (&lt;=&gt;)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>01 (&lt;)</td>
<td>01 (&lt;)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>10 (&gt;)</td>
<td>10 (&gt;)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>00 (=)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>11 (&lt;=&gt;)</td>
<td>11 (&lt;=&gt;)</td>
</tr>
</tbody>
</table>

**Sum**

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...
which are consistent between the two terms. Thus the and operation is used to perform this calculation.

<table>
<thead>
<tr>
<th>Term A</th>
<th>Term B</th>
<th>Addition (⊕) Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 (&lt;)</td>
<td>01 (&lt;)</td>
<td>01 (&lt;)</td>
</tr>
<tr>
<td>01 (&lt;)</td>
<td>00 (=)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>01 (&lt;)</td>
<td>10 (&gt;)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>01 (&lt;)</td>
<td>11 (&lt;=&gt;)</td>
<td>01 (&lt;)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>10 (&gt;)</td>
<td>10 (&gt;)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>00 (=)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>01 (&lt;)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>10 (&gt;)</td>
<td>11 (&lt;=&gt;)</td>
<td>10 (&gt;)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>01 (&lt;)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>10 (&gt;)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>00 (=)</td>
<td>00 (=)</td>
</tr>
<tr>
<td>00 (=)</td>
<td>11 (&lt;=&gt;)</td>
<td>00 (=)</td>
</tr>
</tbody>
</table>

Table 13. Addition Table for Temporal Algebra

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

Transitive Closure Computation

The key, limiting factor in all temporal reasoning systems is the ability to efficiently compute full transitive closure over the set of intervals defined. This process ensures consistency across all temporal relations within the system and asserts transitively computed relations for those which have no primitive relation asserted. 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.

**Allen's Constraint Propagation Algorithm**

Allen's constraint propagation algorithm for computing transitive closure is presented below. As previously noted, this algorithm has been shown in [Vil86] to perform on average $O(n^3)$. This is the same order of complexity as Warshall's algorithm for transitive closure.

**Definitions:**
- **Table** - a two-dimensional array indexed by intervals
- **Queue** - a FIFO stack containing changed relations
- **Intervals** - the set of all intervals for which assertions have been made

**ADD:** This routine asserts (adds) a relationship between two intervals, $(i,j)$. It saves the previous value of the cell in the relation matrix for the indexes $i$ and $j$ and then adds the new relationship to the cell location as a union of the new relation and the cell's existing contents. The routine then checks to see if the new value with the asserted relationship is equal to the old value. If it is then nothing further needs to be done and the algorithm terminates. If a change is detected, however, the intervals for which the new assertion was made are added to the set of intervals for which a change has been made.

```
Add(R<i,j>)
begin
Old ← Table[i,j];
Table[i,j] ← Table[i,j] + R<i,j>;
If Table[i,j] ≠ Old
   then Store <i,j> on Queue;
   Intervals ← Intervals ∪ {i,j};
end;

CLOSE:

Close
while Queue ≠ null do
   Get next <i,j> from Queue;
   Propagate(i,j);
end;
```
PROPOGATE: This is the primary function of the algorithm which performs the transitive Closure

**Propagate**(I,J)

For each interval K in Intervals do 

begin 

Temp ← Table[I,K] + (Table[I,J] ⊗ Table[J,K]);
If Temp = 0 

then raise(contradiction_exception);
If Table[I,K] ≠ Temp

then Store <I,K> on Queue;
Table[I,K] ← Temp;
Temp ← Table[K,J] + (Table[K,I] ⊗ Table[I,J]);
If Temp = 0

then raise(contradiction_exception);
If Table[K,J] ≠ Temp

then Store <K,J> on Queue;
Table[K,J] ← Temp;

end;

The following section presents a representation of temporal relations based on a bit-encoded representation of interval end points or “nests” referred to by [All85]. This representation presents a terse coding of the temporal relations and, with the addition of the ambiguous relation, (<=>,), not defined in Allen’s description, allows the representation of 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.
Bit-mapped Closure Algorithm

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. This computation is based on the graphical foundation presented in the previous sections. This operation is illustrated below.

\[
\text{ComputeTransitive}(R1[], R2[])
\begin{align*}
\text{begin} \\
\text{Result[]} & = 0; \\
\text{Result}[1,1] & \leftarrow (R1[1,1] \lor R2[1,1]) \land (R1[1,2] \lor R2[2,1]); \\
\text{Result}[1,2] & \leftarrow (R1[1,1] \lor R2[1,2]) \land (R1[1,2] \lor R2[2,2]); \\
\text{Result}[2,1] & \leftarrow (R1[2,1] \lor R2[1,1]) \land (R1[2,2] \lor R2[2,1]); \\
\text{Result}[2,2] & \leftarrow (R1[2,1] \lor R2[1,2]) \land (R1[1,2] \lor R2[2,2]); \\
\text{If isAmbiguous(Result[])} \text{ then} \\
\text{AmbiguousRelations} & \leftarrow \text{Result[]}; \\
\text{return}(\text{Result[]});
\end{align*}
\text{end;}
\]

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 \( B \ R2[] C \). 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.

Transitive products may result in an ambiguity. These ambiguous relations are key to the 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 \( \text{AmbiguousRelations} \). This set is referred to during the transitive closure computation to identify other relationships that may be recomputed during the process.
The algorithm maintains the relationships in a square matrix. However, the matrix is symmetric about the diagonal with the lower triangular matrix entries being an 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.

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. $>$ $\Rightarrow$ $<$ and $<$ $\Rightarrow$ $>$) 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.

\[
\text{ComputeInverse}(R[]) \begin{align*}
\text{Result}[] &= 0; \\
\text{If } R[1,1] &= 01 \text{ or } 10 \text{ then} \\
\quad &\text{Result}[1,1] \leftarrow 11 \lor R[1,1] \\
\quad &\text{else} \\
\quad &\text{Result}[1,1] \leftarrow R[1,1]; \\
\text{If } R[2,1] &= 01 \text{ or } 10 \text{ then} \\
\quad &\text{Result}[1,2] \leftarrow 11 \lor R[2,1] \\
\quad &\text{else} \\
\quad &\text{Result}[1,2] \leftarrow R[2,1]; \\
\text{If } R[1,2] &= 01 \text{ or } 10 \text{ then} \\
\quad &\text{Result}[2,1] \leftarrow 11 \lor R[1,2] \\
\quad &\text{else} \\
\quad &\text{Result}[2,1] \leftarrow R[1,2]; \\
\text{If } R[2,2] &= 01 \text{ or } 10 \text{ then} \\
\quad &\text{Result}[2,2] \leftarrow 11 \lor R[2,2] \\
\quad &\text{else} \\
\end{align*}
\]
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 \( \oplus \)) of the end point relationship if the relationship is either < or > (i.e. 01 or 10).

The ComputeTransitive operation refers to an IsAmbiguous operation which test if a relationship is ambiguous and returns true if it is or false if not. This operation is shown below.

\[
\text{IsAmbiguous}(R[]) \\
\text{begin} \\
\quad \text{If} \\
\quad \quad R[1,1] = 11 \text{ or } R[1,2] = 11 \text{ or } R[2,1] = 11 \text{ or } R[2,2] = 11 \text{ then} \\
\quad \quad \text{return(true)} \\
\quad \text{else} \\
\quad \quad \text{return(false);} \\
\text{end;} \\
\]

Simply stated, a relation is ambiguous is 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.

\[
\text{isCompatible}(RAssert[],RExist[]) \\
\text{begin} \\
\quad \text{If} \\
\quad \quad (RAssert[1,1] = RExist[1,1] \text{ or } RExist[1,1] = 11 \text{ or } Rassert[1,1] = 11)
\]
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 \( k \leftarrow (i - 1) \) to 0 step -1 do begin
begin
Table[k,j] \leftarrow \text{ComputeTransitive}(Table[k,i],Table[i,j]);
Table[j,k] \leftarrow \text{ComputeInverse}(Table[k,j]);
end;
end;
For \( k \leftarrow (i + 1) \) to \( j \) step 1 do begin
begin
Table[k,j] \leftarrow \text{ComputeTransitive}(Table[k,i],Table[i,j]);
Table[j,k] \leftarrow \text{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 matrix through the Add operation. This operation asserts a relation $R[i]$ for an interval pair $<i,j>$ within the matrix. The Add operation is shown below.

```plaintext
Add(R[i]<i,j>)
begin
  If Table[i,j] is null then
    begin
      Table[i,j] ← R[i]<i,j>;  
      Table[j,i] ← ComputeInverse(R[i]);
      ComputeColumn(i,j);
      return();
    end;
  If isAmbiguous(Table[i,j]) then
    If isCompatible(R[i],Table[i,j]) then
      begin
        Table[i,j] ← R[i]<i,j>;
        Table[j,i] ← ComputeInverse(R[i]);
        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 which 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.

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.

\[
\text{ComputeClosure()}
\]

begin

while ChangedRelations is \text{null} do

begin

\(\langle i,j \rangle \leftarrow \text{first element of ChangedRelations};\)

\text{For each pair} \(\langle k,l \rangle\) \text{in row} \(i\), column \(i\), row \(j\), or column \(j\)

\text{where} \(\langle k,l \rangle\) \text{is in the upper triangular matrix and} \(\langle k,l \rangle\) \text{is ambiguous} do

begin

If \(i=k\) then

begin

\(\text{Table}[k,l] \leftarrow \text{ComputeTransitive}(i,j,l);\)

\(\text{Table}[l,k] \leftarrow \text{ComputeInverse}(R[k,l]);\)

\(\text{ChangedRelations} \leftarrow \langle k,l \rangle;\)

\(\text{Remove} \langle k,l \rangle \text{from} \text{AmbiguousRelations};\)

end;

If \(j=l\) then

begin

\(\text{Table}[k,l] \leftarrow \text{ComputeTransitive}(k,i,l);\)

\(\text{Table}[l,k] \leftarrow \text{ComputeInverse}(R[k,l]);\)

\(\text{ChangedRelations} \leftarrow \langle k,l \rangle;\)

\(\text{Remove} \langle k,l \rangle \text{from} \text{AmbiguousRelations};\)

end;

If \(i=l\) then

begin

\(\text{Table}[k,l] \leftarrow \text{ComputeTransitive}(k,j,i);\)

\(\text{Table}[l,k] \leftarrow \text{ComputeInverse}(R[k,l]);\)

\(\text{ChangedRelations} \leftarrow \langle k,l \rangle;\)

\(\text{Remove} \langle k,l \rangle \text{from} \text{AmbiguousRelations};\)

end;

If \(j=k\) then

begin

\(\text{Table}[k,l] \leftarrow \text{ComputeTransitive}(j,i,l);\)

\(\text{Table}[l,k] \leftarrow \text{ComputeInverse}(R[k,l]);\)

\(\text{ChangedRelations} \leftarrow \langle k,l \rangle;\)

\(\text{Remove} \langle k,l \rangle \text{from} \text{AmbiguousRelations};\)

end;

end;

end while;
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 chapter performs an analysis 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.
CHAPTER 4
IMPLEMENTATION AND ANALYSIS OF THE TEMPORAL ALGEBRA

This chapter presents an analysis of the performance of the algorithm in terms of its complexity. An initial implementation of the temporal algebra and algorithms developed in the previous chapter is then presented. This implementation is then used as the vehicle for several tests to gather empirical data regarding the actual complexity and performance of the algorithm.

The Temporal System Operations

The calculation of the transitive closure depends on the representation of the interval network in some space efficient form. The implementation developed as part of this research uses a four element list to represent the bit-encoded relationship. This is illustrated in Figure 58 below.

<table>
<thead>
<tr>
<th>R</th>
<th>Bs</th>
<th>Be</th>
</tr>
</thead>
<tbody>
<tr>
<td>As</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>Ae</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>00 =&gt; 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 =&gt; 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 =&gt; 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 =&gt; 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R(1,1) R(1,2) R(2,1) R(2,2)

(01 01 10 11)

(1 1 2 3)

Figure 58. Matrix of Interval Relationships
Relation $R(A,B)$ consists of the relation matrix shown in Figure 58. The matrix can be represented in a list form by the list of elements $(R_{11}, R_{12}, R_{21}, R_{22})$. The binary values can be mapped to their corresponding integer representations. Thus, for the relation matrix shown above, the 4-tuple list representation is $(1 1 2 3)$. This representation shall be used throughout the balance of the dissertation when referring to implementation details and calculations.

**Table 14. Table of Bit-Encoded Relationships**

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Bit-Encoded List Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedes (p)</td>
<td>(1 1 1 1)</td>
</tr>
<tr>
<td>Meets (m)</td>
<td>(1 1 0 1)</td>
</tr>
<tr>
<td>Overlaps (o)</td>
<td>(1 1 2 1)</td>
</tr>
<tr>
<td>Starts (s)</td>
<td>(0 1 2 1)</td>
</tr>
<tr>
<td>Finishes (f)</td>
<td>(2 1 2 0)</td>
</tr>
<tr>
<td>During (d)</td>
<td>(2 1 2 1)</td>
</tr>
<tr>
<td>Equal (e)</td>
<td>(0 1 2 0)</td>
</tr>
<tr>
<td>During Inverse (di)</td>
<td>(1 1 2 2)</td>
</tr>
<tr>
<td>Finishes Inverse (fi)</td>
<td>(1 1 2 0)</td>
</tr>
<tr>
<td>Starts Inverse (si)</td>
<td>(0 1 2 2)</td>
</tr>
<tr>
<td>Overlaps Inverse (oi)</td>
<td>(2 1 2 2)</td>
</tr>
<tr>
<td>Meets Inverse (mi)</td>
<td>(2 0 2 2)</td>
</tr>
<tr>
<td>Precedes Inverse (pi)</td>
<td>(2 2 2 2)</td>
</tr>
</tbody>
</table>
Insertion of New Intervals

New intervals inserted into the network form a new row/column pair in the matrix. If five intervals have been defined, a five-by-five square matrix is used to represent the set of all relations between the intervals. Adding a new intervals results in a new row and column being added to represent the new intervals relationships creating an \( n \times n \) relation matrix of size six.

When a new interval is created and added to the matrix, at least one relationship must be asserted between the new interval and one of the previously defined intervals. Based on a single relationship, the ComputeColumn function is called using the row and column indices of the new relationship to compute the remainder of the relationships between the new interval and the others. Thus, adding a new interval and relationship and then calculating the remainder of the transitive relationships for the new interval requires only a single pass through each interval relationship in the column. The row entries are computed within the ComputeColumn routine as the inverse of each column entry computed. Therefore, the ComputeColumn routine is linearly dependent upon the number of intervals (\( n \)) in the matrix.

Removal of an Interval

Removing an interval is straightforward and requires no transitive calculation. When an interval is removed, the remainder of the intervals and their relationships remain valid. Thus, no additional computation is required. Furthermore, the algorithm could be enhanced to allow for interval removal by simply tagging the row and column of a removed interval as being invalid. All other interval relationships remain valid and do not need to be modified.
Asserting a Relation Over a Computed Value

When a relation is asserted over a computed closure, the relationship to be asserted is compared against the existing entry to ensure that it is compatible. The computed value within the matrix entry contains within its encoding all possible relationships that could be asserted for the given table entry. However, when asserting a relation over a previously computed value, it must be checked to ensure that it is compatible with the existing ambiguous relation. For example, the overlaps relation code (1 1 2 1) would not be compatible with an ambiguous relation of (1 1 3 2) while it would be compatible with the ambiguous relation (1 1 3 1). Thus, when a relation is asserted over an existing relation and it is found not to be compatible then an exception is raised.

Implementation

This section presents the implementation of the bit-mapped temporal system presented in the previous chapter. The system was implemented using LISP on a Windows computer. After the routines were tested and verified for correct operation, the system was invoked on a SUN SPARC workstation where the benchmarks and performance testing were performed.

The system has three essential groups of components. These are:

- **Core routines** - These are the basic functions which comprise the bit-mapped algebra.

- **Support routines** - These are functions which provide the infrastructure around which the Core Routines implement the temporal algebra.

- **Test Routines** - These are a set of functions and routines which automate the generation and testing of the algorithms and collect empirical data for performance analysis.
These functional groups are presented in the remainder of the chapter.

Core Routines

The core routines provide the basic implementation of the bit-mapped temporal algebra. These routines follow the algorithm description in the previous chapter with the addition of basic interval and matrix structure manipulation routines.

Intervals are created using the CreateInterval function. This routine instantiates an interval structure and inserts it into the interval array. The interval array is a one-dimensional array that can contain up to MaxIntervals number of relations. The control variable MaxIntervals is set as an optional parameter to the initialize function. After the interval structure has been created and entered into the next slot in the array, the array index of the interval is returned. All other operations on intervals use the index to refer to the interval structure. This index is also used as the row and column index into the relation matrix for relation assertions and transitive computations.

(defun createInterval (&key (start 0) (end 0) (duration 0))
  (if (< IntervalIndex MaxIntervals)
      (prog2
       (setf (aref temporalIntervals IntervalIndex)
             (make-interval :start start :end end :duration duration))
       IntervalIndex
       (setf IntervalIndex (1+ IntervalIndex))
       (if *printTrace*
          (format *fo* "Interval <-A> created.-% IntervalIndex))
          (error "Press Return to Continue" "Maximum number of intervals (-A) already created." maxintervals)))

The inverse of a relation is calculated by computing the inverse of the greater-than (10) and less-than (01) relations and interchanging cells (1,2) and (2,1). Cells (1,1) and (2,2) remain in place. Ambiguous entries (11) and equal (00) are not inverted. This is performed through the relation-inverse function show
below. The function takes a relation structure, inverts the entries, and returns the result.

(defun relation-inverse (rel)
  (list (if (or (= (nth 0 rel) 1)
                 (= (nth 0 rel) 2))
           (logxor 3 (nth 0 rel))
           (nth 0 rel))
           (if (or (= (nth 2 rel) 1)
                   (= (nth 2 rel) 2))
               (logxor 3 (nth 2 rel))
               (nth 2 rel))
           (if (or (= (nth 1 rel) 1)
                   (= (nth 1 rel) 2))
               (logxor 3 (nth 1 rel))
               (nth 1 rel))
           (if (or (= (nth 3 rel) 1)
                   (= (nth 3 rel) 2))
               (logxor 3 (nth 3 rel))
               (nth 3 rel))))

The assert-relation function provides the basic entry function for asserting a relationship between two intervals within the relationship matrix. The function takes three arguments, the index of the first interval, the relation symbol of the relation to be asserted, and the index of the second interval. The function retrieves the current value in the relation matrix for the row (index of the first interval) and column (index of the second relation). If the value is null then this is a new interval being inserted into the network and the ComputeColumn routine is called after the asserted relation is inserted in the matrix.

If the row and column already has an entry, it is checked to see that it is ambiguous and compatible with the new relation to be asserted. For a relation to be compatible, it means that it can be inserted into the matrix at the specified point, without conflicting with the relationship already there. For example, the relation (2 1 2 1) would be compatible with the ambiguous relation (3 3 2 1) because the two entries that are ambiguous can be resolved to the values proposed by the new relation to be asserted. However, attempting to assert the
relation (2 1 2 1) over an ambiguous relation of (3 2 2 1) would result in a temporal inconsistency because the second end point relationship of the relation to be asserted (1) conflicts with the second end point relation of the previously computed interval (2). Thus, the existing relationship states that the start point of the first interval is greater than the end point of the second interval. The relationship being asserted, however, states exactly the opposite.

(defun assert-relation (il relation i2)
  (let* ((rel-code (get-rel-code relation))
         (old-rel (get-relation il i2)))
    (if (and rel-code
           (is-ambiguous-p old-rel)
           (is-compatible-p rel-code old-rel))
        (progn
          (setf (aref temporalRelations (get-relation-index il i2))
                rel-code)
          (setf (aref temporalRelations (get-relation-index i2 il))
                (relation-inverse rel-code))
          (if *printTrace*
              (format *fo* "Relation asserted \(-A => (~A, ~A)\)\-%*
                      rel-code il i2))
          (if (null old-rel)
              (if (< il i2)
                  (compute-column il i2)
                  (compute-column i2 il))
              (progn
                (push (if (< il i2) (list il i2) (list i2 il))
                      changedRelations)
                (remove-ambiguous-relation
                 (if (< il i2) (list il i2) (list i2 il)))
                (compute-closure)))
          rel-code)
        (error "Invalid relation provided."))))))

If the prior value in the matrix was null then the routine automatically invokes the ComputeColumn function which then computes the transitive product for the remainder of the entries in the column and the inverses along the row for the new interval.

If the relation is being asserted over a previous ambiguous relation then the relation pair is pushed onto the ChangedRelations queue and the
ComputeClosure function is called to recompute the transitive relationships for those ambiguous relationships which are related to the asserted entry.

The relation-product function is one of the simplest functions within the temporal algebra implementation and yet it forms the core of the bit mapped algebra. It is this relation-product function which actually computes the transitive relationship using the bit algebra. This function takes two relations and returns the transitive relation computation.

(defun relation-product (rel1 rel2)
  (list (logand (logior (nth 0 rel1) (nth 0 rel2))
           (logior (nth 1 rel1) (nth 2 rel2)))
        (logand (logior (nth 0 rel1) (nth 1 rel2))
           (logior (nth 1 rel1) (nth 3 rel2)))
        (logand (logior (nth 2 rel1) (nth 0 rel2))
           (logior (nth 3 rel1) (nth 2 rel2)))
        (logand (logior (nth 2 rel1) (nth 1 rel2))
           (logior (nth 3 rel1) (nth 3 rel2))))

The compute-transitive function provides the higher-level logic of the transitive relation calculation. It checks to see if the target relation exists or is ambiguous. It also checks that the computed product is compatible with the existing target relation. If there is no relationship already at the target then it simply inserts the new relationship into the matrix. If a previously computed relationship exists, then it must perform a bit-logical and over each of the relation bit codes for the computed value and the target relation. This reduces ambiguity by refining each bit code in the relation to the most specific representation possible. For example, performing a relation-and on the two relations (3 2 2 1) and (1 3 3 1) would yield the relation (1 2 2 1). This is relation is the unambiguous relation which is compatible with both of the ambiguous relations shown.

After the relation is calculated and inserted into the matrix, the function then computes the inverse and inserts it into the appropriate location in the row.
The computed relation is then checked to see if it is ambiguous. If so, the interval pair defining the ambiguous relation is added to the set of interval pairs in the `AmbiguousRelations` list. Finally, if the newly computed relation is not ambiguous it is pushed onto the `ChangedRelations` queue and it is removed from the `AmbiguousRelations` list.

```lisp
(defun compute-transitive (il i2 i3)
  (if *printTrace*
      (format *fo* "Computing transitive relation over ~A -> ~A -> ~A-%" il i2 i3))
  (let* ((targetIndex (get-relation-index il i3))
         (startIndex (get-relation-index il i2))
         (endIndex (get-relation-index i2 i3))
         (targetRelation (aref temporalRelations targetIndex))
         (startRelation (aref temporalRelations startIndex))
         (endRelation (aref temporalRelations endIndex)))
    (if (or (null startRelation) (null endRelation))
        nil
      (if (or (null targetRelation) (is-ambiguous-p targetRelation))
          (let ((rprod (relation-product startRelation endRelation))
                  (targetAmbiguous (and (not (null targetRelation))
                                        (is-ambiguous-p targetRelation))))
            (if (is-compatible-p rprod targetRelation)
                (progn
                  (setf (aref temporalRelations targetIndex) (if (null targetRelation) rprod (relation-and rprod targetRelation)))
                  (setf (aref temporalRelations (get-relation-index i3 il)) (relation-inverse (aref temporalRelations targetIndex)))
                  (if (is-ambiguous-p (get-relation il i3))
                      (add-ambiguous-relation (list il i3))
                      (if targetAmbiguous
                          (progn
                            (push (list il i3) changedRelations)
                            (remove-ambiguous-relation (list il i3))))))
            (if *printTrace*
                (format *fo* "Transitive computation ~A =\(\) (~A, ~A)-%" rprod il i3)))))
      (if *printTrace*
          (format *fo* "Target (~A,-A) is already non-ambiguous.-%" il i3)))
  )
)
```

The ComputeColumn routine takes a row and column value of a new relation in the matrix and computes the rest of the column entries for the new interval. The inverse values in the corresponding row are simply computed
from the transitive relation and inserted into the corresponding location in the row.

(defun compute-column (i1 i3)
  (if *printTrace*
      (format *fo* "Computing column for (-A, -A)-%" i1 i3))
  (if (> i1 0)
      (dotimes (i2 i1)
        (if (is-ambiguous-p (get-relation i2 i3))
            (compute-transitive i2 i1 i3)
            (setf (aref temporalRelations (get-relation-index i3 i2))
                  (relation-inverse (get-relation i2 i3))))
      (if (< i1 (1- i3))
          (dotimes (i2 (1- (- i3 i1)))
            (if (is-ambiguous-p (get-relation (1+ (+ i1 i2)) i3))
                (compute-transitive (1+ (+ i1 i2)) i1 i3)
                (setf (aref temporalRelations (get-relation-index i3 (1+ (+ i1 i2))))
                      (relation-inverse (get-relation (1+ (+ i1 i2)) i3))))))

The ComputeClosure function is called from within the assert-relation function when a relation is asserted over a previously ambiguous relation. The compute-closure function pops entries off the ChangedRelations queue and compares the interval indexes against the relation index pairs stored in the AmbiguousRelations list. The function remains active while there are changed relations available to be popped off the ChangedRelations queue. For each of the interval pairs popped off the ChangedRelation queue, it iterates over the AmbiguousRelations list using the check-transitive function which is described below to see if the change can be propagated further within the matrix. The function shown also has several instrumentation points used to capture performance information, timing, and cycles through the while loop within the function.

(defun compute-closure ()
  (let (changeRelation
          (start-time (current-time))
          (outer-cycles 0)
          (inner-cycles 0)
          end-time)
    (while changedRelations
      (setf changeRelation (pop changedRelations)))
The check-transitive function takes the changed relation popped off of the ChangedRelations queue and an ambiguous relation and checks to see if there is any common intervals between the end points of the changed relation and the ambiguous relation. If so, then a transitive computation can be performed using the common interval as the transitive center.

There are four possible combinations of end points. This function simply checks each of the four combinations and, if it finds one to be true, calculates the transitive relation using the compute-transitive function over the three intervals.
The next section presents the support routines which provide supporting roles to the core functions of the algebra described above.

**Support Routines**

The following routines are not part of the core algebra implementation but, together with the algebra functions, form a complete, functioning system. The get-relation function returns a relation code from the matrix, given a pair of interval indexes.

```lisp
(defun get-relation (il i2)
  (aref temporalRelations (get-relation-index il i2)))
```

The interval itself is a simple structure containing three data items, start, end, and duration. These data items are time point values which, for the interval algebra computations, do not need to be provided. These items are used when the algebra is working in a hybrid domain of partial knowledge regarding time points and interval relationships. The interval end point basis of the algebra allows the system to seamlessly integrate the quantitative information of time points with the qualitative information of the interval algebra.

```lisp
(defstruct interval
  start
  end
  duration)
```

The relation codes are stored in a table as a set of 5-tuple lists. The first element in the list is the symbol of the interval relationship (e.g. oi) and the remaining four items is the bit code for the relationship. The nthrelcode function is used as a support function to the test routines for obtaining the relation mnemonic given a random index into the relation table.

```lisp
(defun nthrelcode (index)
  (car (nth index *relationCodes*)))
```
When a relation is asserted, the symbol for the relation is used as an argument to get-rel-code which uses it as a lookup key for the relation code table. It then returns the bit codes for the relationship requested.

(defun get-rel-code (mnemonic)
  (cdr (assoc mnemonic 'relationCodes*)))

Part of the algorithm checks to see if the relation location in the matrix already contains an ambiguous relation. This is performed with the is-ambiguous-p function shown below. An ambiguous relation is defined as a relation which has an unknown value (11) in any of the four entries of the relation bit codes. This function also returns true if the relation provided as the function argument is null. This is used when adding a new interval to the matrix and relationships had not yet been asserted for the interval.

(defun is-ambiguous-p (relation)
  (if (null relation)
      t
      (or (= 3 (nth 0 relation))
          (= 3 (nth 1 relation))
          (= 3 (nth 2 relation))
          (= 3 (nth 3 relation)))))

The is-ambiguous-relation function is provided a pair list of interval indexes. It checks to see if the interval pair specified is currently defined as an ambiguous relation. It checks the pair list against the entries in the AmbiguousRelations list returning true if it finds the pair or null if it reaches the end of the list.

(defun is-ambiguous-relation (pair-list)
  (do ((relations ambiguousrelations (cdr relations))
       (answer (if relations
                 (equal pair-list (car relations)))
               (if relations (equal pair-list (car relations))))
       (if (or (null relations) answer) answer))))
The remove-ambiguous-relation function takes an interval pair and removes it from the AmbiguousRelations list. The add-ambiguous-relation function is provided an interval pair to add to the AmbiguousRelations list. If the pair is not already a member of the list, it pushes the interval pair onto the list.

```
(defun remove-ambiguous-relation (rel)
  (do ((oldAmbiguous ambiguousRelations (cdr oldAmbiguous))
       newAmbiguous)
      ((null oldAmbiguous)
       (setf ambiguousRelations newAmbiguous))
    (if (not (equal rel (car oldAmbiguous)))
      (setf newAmbiguous (cons (car oldAmbiguous) newAmbiguous))))

(defun add-ambiguous-relation (rel)
  (if (not (is-ambiguous-relation rel))
      (push rel ambiguousRelations)))
```

Before a transitive computation can be used to update the relation matrix, it must checked to ensure that the computation is compatible with any value that may already be present at that location in the matrix. The function is-compatible-p performs this check. It ensures that each bit code entry in the computed transitive relation and the existing relation are compatible.

It uses the function compatible-points to check each of the four combinations. Two points are defined to be compatible if 1) they are equal, 2) either of the points are ambiguous (i.e. equal to 3), or 3) the logical and of the two points are not zero.

```
(defun is-compatible-p (new-rel old-rel)
  (cond ((null old-rel) t)
        ((and (compatible-points (nth 0 new-rel) (nth 0 old-rel))
              (compatible-points (nth 1 new-rel) (nth 1 old-rel))
              (compatible-points (nth 2 new-rel) (nth 2 old-rel))
              (compatible-points (nth 3 new-rel) (nth 3 old-rel)))
     t)
        (t nil)))

(defun compatible-points (p1 p2)
  (or
   (= p1 p2)
   (= (bitcode p1) 3)
   (= (bitcode p2) 3)
   (= (bitcode (and p1 p2)) 0))
```
\begin{verbatim}
(or (= p1 3) (= p2 3))
(if (and (> p1 0) (> p2 0))
   (not (zerop (logand p1 p2)))))))

The maximum number of relation entries that must be handled is computed by max-rels, based on the number of intervals provided. This number is used to allocate a relation matrix of the proper size.

(defun max-rels (number-of-intervals)
  (truncate (/ (* number-of-intervals (1- number-of-intervals)) 2)))

The relation-and function performs a logical and of two relations. The logical and is performed for each entry in the relation matrix. The resulting interval relation is returned.

(defun relation-and (rell re12)
  (list (logand (nth 0 rell) (nth 0 re12))
        (logand (nth 1 rell) (nth 1 re12))
        (logand (nth 2 rell) (nth 2 re12))
        (logand (nth 3 rell) (nth 3 re12))))

(defun get-relation-index (il i2)
  (+ (* i2 MaxIntervals) il))

The print-table function prints the relation table to the console or file specified by the stream variable *fo*. It also prints out the number and list of ambiguous relations remaining in the system.

(defun print-table ()
  (format *fo* "-A ambiguous relations: ~%-A~%-"
          (length ambiguousRelations) ambiguousRelations)
  (format *fo* "~%-"
  (dotimes (k intervalIndex)
    (if (< k 10)
      (format *fo* " 0-A " k)
      (format *fo* " -A " k)))
  (format *fo* "~%-")
  (dotimes (i intervalIndex)
    (if (< i 10)
      (format *fo* " 0-A " i)
      (format *fo* " -A " i))
    (dotimes (j intervalIndex)
      (let ((relation (get-relation i j)))
        (if (null relation)
          

\end{verbatim}
The following section presents the control functions of the temporal algebra implementation.

**Control Functions**

The system is set up for use through the initialize function. This function sets some control variables, identifies the maximum number of intervals to be tracked, and creates empty arrays to store the set of intervals as they are defined and the relation matrix.

The control variable *fo* is a stream variable which, by default is set to true. Any output directed to the output stream would print on the console in the default case. To capture the trace of the operations, the start-log function is called with a file name to which the output is directed. The function opens the file name specified for output and sets the stream variable *fo* to the stream handle returned. Any output is then directed into the output file.

The output stream is reset using the stop-log function. This function closes the output stream and reset the stream variable to true, directing output back to the console.

```
(defun initialize (&optional (maxInt 20))
  (setf *printTrace* t)
  (setf MaxIntervals maxInt) ; Maximum number of intervals allowed.
  (setf IntervalIndex 0) ; Current interval index
  (setf AmbiguousRelations nil) ; List of ambiguous relations
  (setf changedRelations nil) ; List of newly asserted relations
  (setf MaxRelations (* MaxIntervals MaxIntervals))
  (setf temporalIntervals (make-array MaxIntervals))
  (setf temporalRelations (make-array MaxRelations)))
```
(setq *fo* t)

(defun start-log (filename)
  (setq *fo* (open filename :direction :output)))

(defun stop-log ()
  (close *fo*)
  (setq *fo* nil))

The `printTrace` function sets the `*printTrace*` flag which is used to enable or inhibit the printing of detail messages during execution. The `createIntervals` function creates a number of intervals specified by the input argument.

(defun printTrace (flag)
  (setq *printTrace* flag))

(defun createIntervals (num)
  (do ((i num)) ((> i num))
    (createInterval))
  num)

The relations are stored in the `*relationCodes*` table. These are stored as 5-tuple with the first entry in the tuple being the mnemonic symbol for the relation code. The last fours elements of the list are the binary code for the relation in integer list form.

(setq *relationCodes* '(
  (p 1 1 1 1)
  (m 1 1 0 1)
  (o 1 1 2 1)
  (s 0 1 2 1)
  (f 2 1 2 0)
  (d 2 1 2 1)
  (e 0 1 2 0)
  (di 1 1 2 2)
  (fi 1 1 2 0)
  (si 0 1 2 2)
  (oi 2 1 2 2)
  (mi 2 0 2 2)
  (pi 2 2 2 2)
))
**Analysis**

This section presents an analysis of the transitive closure algorithm. The closure algorithm forms the computationally intensive portion of the system and, consequently, is the driving factor in the overall efficiency. The analysis is developed as a study of the complexity of the algorithm. Specifically, the overall structure of the algorithm is analyzed, and iterative loops are identified, as are any other embedded routines which themselves contain iterative loops.

The complexity of the algorithm is then developed as a function of the input data provided to the algorithm. This analysis approach is a straightforward method widely employed in evaluating algorithm complexity.

**Storage Requirements**

As expected, the storage requirements for the transitive relation matrix requires $n^2$ storage. However, because of the separation of basic temporal relationship information from other types of data and the terse representation of temporal relationships within bit codes, there are some storage tradeoffs and savings that are worthy of mention. These are presented in the next two sections.

**Basic Relation Matrix**

As mentioned, the relation matrix requires $n^2$ storage space. However, because of the bit-encoded representation, each entry requires only a single byte to represent the different relationships possible for a given entry. Thus, even for a matrix containing 1,000 intervals, only one megabyte would be required to represent all possible relationships between the intervals. While this may have been an excessive demand in the past, the availability and cost of memory make a megabyte requirement seem almost minor.
In fact, the required storage could be squeezed a bit further. Since the matrix represents two triangular matrices, one being the inverse of the other, the application algorithms could be modified to simply use a triangular matrix and, when an inverse was required, simply calculate it using the ComputeInverse function for the base relationship entry. Also, the diagonal will always be the equals relationship. This could then be removed as well.

This would reduce the storage requirement to \( n(n-1)/2 \). This reduction is still \( O(n^2) \) in the limit. However, in practical applications, the reduction would have a significant impact. For example, in the case of a 1,000 interval temporal system, the storage requirements would be reduced from roughly 1,000,000 bytes to 500,000 bytes. This would represent a significant space savings in an operational system.

**Additional Temporal Information**

In addition to representing temporal relationships, one of the capabilities of the algorithm and representation was the integration of discrete time points as part of the interval representation. This is contained within the interval structure. The interval structure contains three data items, 1) the start point, 2) end point, and 3) duration. The relationship between these three items is self-evident. The difference between the start and the end points is the duration. Given any two of the items, the third may be calculated.

An important concept, however, is the fact that, through the bit-mapped encoding of the relationships in the matrix, none of the time points is required to completely represent a set of relationships between the intervals in the system. In fact, the relationships asserted imply certain relationships exist between the end points of an interval pair. Thus, the bit-encoded representation can also be
used to verify discrete time points as they are asserted for intervals within the system.

For example, if the relationship A overlaps B is asserted, then the relationships between the end points of the intervals dictate that the start point of B lie between the start and end point of A. If the start and end point of A is asserted to be 34535 and 56453, respectively, then attempting to assert that the time point associated with the start point of interval B is 65765 will raise an exception.

Performance

The transitive closure portion of the algorithm is presented below. The iterative loops are highlighted by the boxed areas. There are two iterative loops as shown by the double and single line borders. The inner loop iterates over the row and column for the intervals of the changed relation. The outer loop iterates over a queue of changed relations while the queue is not empty.

```
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>;
      Remove <k,l> from AmbiguousRelations;

    end

  end

end

```


The first distinction of the algorithm is the iteration over the row and column of each interval index in the changed relation rather than the entire set of interval pairs as in Allen’s original algorithm. The worst case is a totally ambiguous matrix where the number of ambiguous relations is equal to the entries in the matrix\(^3\) or \(n^2\) where \(n\) = the number of intervals in the system.

The `changedRelations` queue is initialized to the value of the interval pair for which a new, unambiguous relation is asserted. This value is popped off the

---

\(^3\) 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.
queue 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 then the relation is removed from the AmbiguousRelations set and inserted into the ChangedRelations queue. Thus, the number of maximum comparisons performed by the closure algorithm is $4n^2n(n-1)/2$ or $O(n^3)$. The maximum possible value of $n$ is the number of intervals in the system. Consequently, the worst case performance of the algorithm is $O(n^3)$.

**Scalability**

An important consideration of any computer algorithm is the scalability or the basic implementation. The algorithm presented maintains the same relative complexity as the number of intervals grows. This still can result in a lengthy calculation as the number of intervals grows very large.

An alternative implementation approach using parallel processing may provide a solution to the processing. As a new interval is asserted and the closure computation is initiated, each of the ambiguous intervals inspected can be done in parallel. So, if there are 25 ambiguous relations within the system and a refinement of one of the intervals is asserted, the remaining 24 ambiguous relations could be inspected in parallel.
Summary

In this chapter an implementation of the algorithm presented in Chapter 3 was presented in detail. The implementation was analyzed with regards to complexity, performance, and storage requirements. As was shown in the chapter, the performance of the algorithm is as predicted at the outset of the dissertation.

In the next chapter data collected during various tests and execution of the algorithm implementation is presented and analyzed. Several stress tests were developed and executed to validate the performance of the algorithm as obtained through the analysis of this chapter.
CHAPTER 5
DATA COLLECTION AND PERFORMANCE ANALYSIS

This chapter presents test cases, empirical data gathered through execution of the test cases, and an analysis of the data collected. The test cases developed may be categorized into one of several basic types. The first is a basic validation of the transitive relation computation algorithm. The second is a set of test executions to validate the complexity of the matrix in terms of the number of ambiguous relationships. The third is a trace of an interactive session with the system in which intervals were defined, relations asserted, and unambiguous relations asserted over ambiguous relations in the matrix which triggered the closure computation algorithm. Finally, the last type of test addresses the scalability of the algorithm for larger interval sets.

Definition of Test Cases

Computation of a transitive relation is the resolution of relationships between the end-points of two intervals via a common interval.
Figure 59. Example of Temporal Intervals and Relationships

Figure 59 identifies a set of six intervals and five assertions. This set of intervals and assertions result in the matrix shown below in Figure 60.

![Matrix showing primitive assertions](image)
As Figure 60 shows, the primitive relation assertions and their inverses are shown as shaded entries in the relation matrix. Computing a transitive relation in the matrix is illustrated in Figure 61. The darker shaded cells show the two relations over which the transitive relation is being calculated. The lighter shaded cell shows the target or result cell.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00 01</td>
<td>01 01</td>
<td>01 01</td>
<td>01 01</td>
<td>01 01</td>
<td>01 01</td>
</tr>
<tr>
<td></td>
<td>10 00</td>
<td>10 01</td>
<td>11 01</td>
<td>11 01</td>
<td>01 01</td>
<td>10 00</td>
</tr>
<tr>
<td>B</td>
<td>10 01</td>
<td>00 01</td>
<td>01 01</td>
<td>01 01</td>
<td>01 01</td>
<td>11 01</td>
</tr>
<tr>
<td></td>
<td>10 10</td>
<td>10 00</td>
<td>10 01</td>
<td>10 00</td>
<td>00 01</td>
<td>10 10</td>
</tr>
<tr>
<td>C</td>
<td>10 11</td>
<td>10 01</td>
<td>00 01</td>
<td>11 01</td>
<td>01 01</td>
<td>11 11</td>
</tr>
<tr>
<td></td>
<td>10 10</td>
<td>10 10</td>
<td>10 00</td>
<td>10 10</td>
<td>10 11</td>
<td>10 10</td>
</tr>
<tr>
<td>D</td>
<td>10 11</td>
<td>10 01</td>
<td>11 01</td>
<td>00 01</td>
<td>01 01</td>
<td>11 11</td>
</tr>
<tr>
<td></td>
<td>10 10</td>
<td>10 00</td>
<td>10 01</td>
<td>10 00</td>
<td>00 01</td>
<td>10 10</td>
</tr>
<tr>
<td>E</td>
<td>10 10</td>
<td>10 00</td>
<td>10 01</td>
<td>10 00</td>
<td>00 01</td>
<td>10 10</td>
</tr>
<tr>
<td></td>
<td>10 10</td>
<td>10 10</td>
<td>10 11</td>
<td>10 10</td>
<td>10 00</td>
<td>10 10</td>
</tr>
<tr>
<td>F</td>
<td>10 01</td>
<td>11 01</td>
<td>11 01</td>
<td>11 01</td>
<td>01 01</td>
<td>00 01</td>
</tr>
<tr>
<td></td>
<td>10 00</td>
<td>10 01</td>
<td>11 01</td>
<td>11 01</td>
<td>01 01</td>
<td>10 00</td>
</tr>
</tbody>
</table>

**Figure 61. Computing a Transitive Relation in the Upper Half of the Matrix**

As the example in Figure 61 shows, the transitive relation calculation of \( A \rightarrow B \rightarrow C \) results in an ambiguous relation being asserted for \( A \rightarrow C \). Figure 62 illustrates a transitive computation being performed which utilizes a inverse relation from the lower portion of the matrix.
Figure 62. Computing a Transitive Relation Using an Inverse Relationship

The relation $D \rightarrow B \rightarrow E$ is used to compute $D \rightarrow E$. As shown in the figure, the resultant computation is unambiguous.

**Test Cases**

The following routines were implemented to generate test cases for analyzing the performance of the algorithm. The test generator developed a set of temporal system consisting of a random number of intervals between 2 and 25. One temporal relation for each column in the matrix was selected at random and the resultant closure was calculated.

The number of ambiguous intervals was then identified along 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.

(defun testnumambiguous (&optional fileName)
(initialize)
(if fileName (start-log fileName))
;(format *fo* "Number of ambiguous relations:-%")
(dotimes (runNum runcount)
  (format t "Run # ~A-%" runNum)
  (let ((intNum (+ 2 (mod (tick-count) numOfInts))))
    (format *fo* "~A," intNum)
    (initialize (1+ intNum))
    (printTrace nil)
    (createIntervals intNum)
    (dotimes (interval (- intNum 2))
      (let ((rel (nthRelCode (mod (tick-count) 12))))
        ;(format *fo* "Asserting (~A ~A ~A-%" interval rel (1+ interval))
        (assert-relation interval rel (1+ interval))))
    (format *fo* "~A-%" (length ambiguousRelations)))
  (if fileName (stop-log)))

In addition to the above test generator, a set of temporal systems was generated containing 20, 40, 60, 80, and 100 intervals respectively. These resulted in temporal matrices of 400, 1,600, 3,600, 6,400, and 10,000 relations, respectively. A relation was selected at random for each of the columns in these tests and the closure computed. The number of ambiguous intervals in each of these system was identified and stored.

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, took the next interval pair off of the ambiguous relations set and asserted into the matrix the first primitive relation which was compatible with the ambiguous relation.

This initiated the transitive closure computation 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.

**Data Collected**

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.

**Distribution of Ambiguous Relations**

Since the performance of the algorithm is critically 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.

Thus, a completely populated 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 63.

![Ambiguous Relations vs Number of Intervals](image)

**Figure 63: Plot of Number of Intervals versus Number of 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 64. 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.

![Logarithmic Plot of Ambiguous Relations vs Number of Intervals](image)

**Figure 64: 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 64 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.

**Timing Data**

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 were to validate the performance and complexity analysis of the algorithm presented in Chapter 4.

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 test data is presented in the appendix section which lists the interaction with the system to construct a 12 interval system and assert several new relations over the ambiguous relations in the system.

Clock time was also collected for the 12 interval system but the large scale tests 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.
Summary of Analysis Results

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 ran a bit less than $n^2$. 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 following sections provide a summary of the specific test cases run. The output from these test cases are furnished as appendices to this dissertation.

Computed Transitivity Table

The goal of this test was to validate the basic transitive relation computation. The program was a simple loop which iterated of the set of thirteen relations and computed the transitive relation of all possible combinations. The results of these computations were then compared to the entries in Allen's transitivity table to validate the computation.

In comparing the results of the computed transitive relations with those in Allen's transitivity table, three results did not match. These were the transitive products resulting from, overlaps and overlaps inverse ($o\ oi$), during and during inverse ($d i\ \ d$), and overlaps inverse and overlaps ($o i\ o$). In all cases, the computed value from the algorithm yielded the set of relations ($o\ sf\ d = d i\ f i\ si\ o i$) while the entry in Allen's transitivity table for each was the set of relations ($o\ oi\ d\ di\ =$).
The difference between the two sets of relations is the set consisting of \((s \ si \ f \ fi)\). The question raised was whether the transitive computation was incorrect or, in fact, there were errors in the transitive table. To answer this question, an example of three intervals which reflected the base relationships was constructed and then analyzed to identify if there were arrangements of the intervals which preserved the original relationships but yielded the relations not stated in Allen’s transitive matrix.

![Interval A](image1)

<table>
<thead>
<tr>
<th>A&lt;br&gt;&lt;o&gt;</th>
<th>B</th>
<th>Interval A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B&lt;br&gt;&lt;oi&gt;</td>
<td>C</td>
<td>Interval B</td>
</tr>
<tr>
<td>A&lt;br&gt;&lt;s&gt;</td>
<td>C</td>
<td>Interval C</td>
</tr>
</tbody>
</table>

![Interval A](image2)

<table>
<thead>
<tr>
<th>A&lt;br&gt;&lt;o&gt;</th>
<th>B</th>
<th>Interval A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B&lt;br&gt;&lt;oi&gt;</td>
<td>C</td>
<td>Interval B</td>
</tr>
<tr>
<td>A&lt;br&gt;&lt;f&gt;</td>
<td>C</td>
<td>Interval C</td>
</tr>
</tbody>
</table>

**Figure 65: Example Showing Error in Transitivity Table**

Figure 65 shows examples for two of the four relations in question. In the first example, the original relationships are maintained, but the transitive relation \(A<s>C\) is a valid assertion. In the second example, the original relations are maintained once again and the relation \(A<f>C\) is valid. The inverses of these relations (si and fi) could also be shown by reversing the size of intervals A and C. This example can be repeated for the other two interval pairs, (d di) and (oi o). These discrepancies were also noted by [She92].

Thus, the transitive relation computation algorithm is correct and can directly compute the transitive relation from the input interval relations. This
eliminates any need for table lookup and reduces the storage requirement by being able to represent sets of relations composing an ambiguous relation within the bit-encoding.

**Random Assertion of Relations**

The next test addressed the number of ambiguous intervals that may be asserted into a system as a result of transitive relation computation. The purpose was to identify the number of ambiguous that were entered into the system and show the general distribution of those intervals.

The test generated 500 temporal matrices. The number of intervals in each individual matrix ranged between 2 and 25 and was selected randomly by the test routine. Then, a relation for each of the intervals and the next interval in the system was selected at random and asserted into the system. Finally, as each new interval relation was asserted, the closure of that interval relation was computed.

After the matrix was totally populated, the number of intervals and the number of ambiguous relations for that test was logged to a recording file. Each entry in the data written to the file consisted of a pair of numbered, the number of intervals and the number of ambiguous relations. These numbers were then sorted and graphed. Both a raw count graph was developed and a logarithmic graph. The graphs were presented in Figures 63 and 64 earlier in this chapter.

The location and shape of the curves plotted indicate that the number of ambiguous relations averages somewhat less than \( n^2 \), where \( n \) is the number of intervals in the system.
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 15.

<table>
<thead>
<tr>
<th>Interval Pair and Relation</th>
<th>Bit Code Asserted</th>
<th>Number of Ambiguous Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 overlaps 1</td>
<td>(1 1 2 1)</td>
<td>0</td>
</tr>
<tr>
<td>0 overlaps 2</td>
<td>(1 1 2 1)</td>
<td>1</td>
</tr>
<tr>
<td>2 meets 3</td>
<td>(1 1 0 1)</td>
<td>2</td>
</tr>
<tr>
<td>2 precedes 4</td>
<td>(1 1 1 1)</td>
<td>4</td>
</tr>
<tr>
<td>3 during 5</td>
<td>(2 1 2 1)</td>
<td>8</td>
</tr>
<tr>
<td>5 overlaps inverse 6</td>
<td>(2 1 2 2)</td>
<td>13</td>
</tr>
<tr>
<td>4 meets 7</td>
<td>(1 1 0 1)</td>
<td>17</td>
</tr>
<tr>
<td>7 finishes inverse 8</td>
<td>(1 1 2 0)</td>
<td>21</td>
</tr>
<tr>
<td>0 finishes 9</td>
<td>(2 1 2 0)</td>
<td>23</td>
</tr>
<tr>
<td>7 during 10</td>
<td>(2 1 2 1)</td>
<td>31</td>
</tr>
<tr>
<td>10 starts 11</td>
<td>(0 1 2 1)</td>
<td>39</td>
</tr>
</tbody>
</table>
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 below.

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>(----)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(3131)</td>
<td>(3333)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2120)</td>
<td>(3131)</td>
<td>(3131)</td>
</tr>
<tr>
<td>01</td>
<td>(2122)</td>
<td>(----)</td>
<td>(3123)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(3133)</td>
<td>(3333)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(2122)</td>
<td>(3133)</td>
<td>(3133)</td>
</tr>
<tr>
<td>02</td>
<td>(2122)</td>
<td>(3123)</td>
<td>(----)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(3121)</td>
<td>(3323)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2122)</td>
<td>(3131)</td>
<td>(3131)</td>
</tr>
<tr>
<td>03</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2022)</td>
<td>(----)</td>
<td>(1133)</td>
<td>(2121)</td>
<td>(2323)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(2222)</td>
<td>(3133)</td>
<td>(3133)</td>
</tr>
<tr>
<td>04</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(----)</td>
<td>(2323)</td>
<td>(2323)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2222)</td>
<td>(3121)</td>
<td>(3121)</td>
</tr>
<tr>
<td>05</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3122)</td>
<td>(1122)</td>
<td>(1133)</td>
<td>(----)</td>
<td>(2122)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(3322)</td>
<td>(3133)</td>
<td>(3133)</td>
</tr>
<tr>
<td>06</td>
<td>(3333)</td>
<td>(3333)</td>
<td>(3133)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(1121)</td>
<td>(----)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(3333)</td>
<td>(3133)</td>
<td>(3133)</td>
</tr>
<tr>
<td>07</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2022)</td>
<td>(2323)</td>
<td>(2323)</td>
<td>(----)</td>
<td>(1120)</td>
<td>(2222)</td>
<td>(2121)</td>
<td>(2121)</td>
</tr>
<tr>
<td>08</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2323)</td>
<td>(2120)</td>
<td>(----)</td>
<td>(2222)</td>
<td>(2212)</td>
<td>(2121)</td>
</tr>
<tr>
<td>09</td>
<td>(1120)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(3131)</td>
<td>(3333)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(----)</td>
<td>(3131)</td>
<td>(3131)</td>
</tr>
<tr>
<td>10</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3122)</td>
<td>(3323)</td>
<td>(3323)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(3322)</td>
<td>(----)</td>
<td>(0121)</td>
</tr>
<tr>
<td>11</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3122)</td>
<td>(3323)</td>
<td>(3323)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(3322)</td>
<td>(3322)</td>
<td>(0122)</td>
</tr>
</tbody>
</table>

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

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

---

\(^4\)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.
algorithm performed 37 iterations for the initial 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 $4 \times 33 + 37$ or 159. The resultant matrix is shown below.

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>(----)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(3121)</td>
<td>(3121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2120)</td>
<td>(3131)</td>
<td>(3131)</td>
</tr>
<tr>
<td>01</td>
<td>(2122)</td>
<td>(----)</td>
<td>(3123)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(3123)</td>
<td>(3123)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(2122)</td>
<td>(3133)</td>
<td>(3133)</td>
</tr>
<tr>
<td>02</td>
<td>(2122)</td>
<td>(3123)</td>
<td>(----)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2121)</td>
<td>(2121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2122)</td>
<td>(3131)</td>
<td>(3131)</td>
</tr>
<tr>
<td>03</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2022)</td>
<td>(----)</td>
<td>(1133)</td>
<td>(2121)</td>
<td>(2123)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(2222)</td>
<td>(3133)</td>
<td>(3133)</td>
</tr>
<tr>
<td>04</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(----)</td>
<td>(2121)</td>
<td>(2120)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2222)</td>
<td>(3121)</td>
<td>(3121)</td>
</tr>
<tr>
<td>05</td>
<td>(3122)</td>
<td>(3123)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(----)</td>
<td>(2122)</td>
<td>(1123)</td>
<td>(1133)</td>
<td>(3122)</td>
<td>(3123)</td>
<td>(3123)</td>
</tr>
<tr>
<td>06</td>
<td>(3122)</td>
<td>(3123)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(----)</td>
<td>(1123)</td>
<td>(1133)</td>
<td>(3122)</td>
<td>(3123)</td>
<td>(3123)</td>
<td>(3123)</td>
</tr>
<tr>
<td>07</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2022)</td>
<td>(2123)</td>
<td>(2022)</td>
<td>(----)</td>
<td>(1120)</td>
<td>(2222)</td>
<td>(2121)</td>
<td>(2121)</td>
</tr>
<tr>
<td>08</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2120)</td>
<td>(----)</td>
<td>(2222)</td>
<td>(2121)</td>
<td>(2121)</td>
</tr>
<tr>
<td>09</td>
<td>(1120)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(3121)</td>
<td>(3121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(----)</td>
<td>(3131)</td>
<td>(3131)</td>
</tr>
<tr>
<td>10</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3122)</td>
<td>(3123)</td>
<td>(3123)</td>
<td>(3122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(3322)</td>
<td>(----)</td>
</tr>
<tr>
<td>11</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3122)</td>
<td>(3123)</td>
<td>(3123)</td>
<td>(3122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(3322)</td>
<td>(0122)</td>
</tr>
</tbody>
</table>

In the first example, the 38 iterations took 36.75 seconds to complete. In the second example which refined an addition 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 the appendices.

**Large Intervals Sets**

The last set of test 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 are 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 were 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.

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.

<table>
<thead>
<tr>
<th>Number of Intervals</th>
<th>Number of Relations</th>
<th>Ambiguous Relations</th>
<th>Ambiguous Relations Remaining</th>
<th>Ambiguous Relations Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>400</td>
<td>107</td>
<td>2</td>
<td>105</td>
</tr>
<tr>
<td>40</td>
<td>1,600</td>
<td>651</td>
<td>305</td>
<td>346</td>
</tr>
<tr>
<td>60</td>
<td>3,600</td>
<td>1,629</td>
<td>1,291</td>
<td>338</td>
</tr>
<tr>
<td>80</td>
<td>6,400</td>
<td>2,953</td>
<td>2,566</td>
<td>387</td>
</tr>
<tr>
<td>100</td>
<td>10,000</td>
<td>4,521</td>
<td>3,926</td>
<td>595</td>
</tr>
</tbody>
</table>
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 \( xR \) 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 than \( x-1 \) iterations were performed.

**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 chapter, the conclusions are summarized for this research, the results are compared with the original goals stated at the beginning of the dissertation, potential areas for further improvement are noted, and future research areas are presented.
CHAPTER 6
CONCLUSIONS AND FUTURE RESEARCH

Conclusions

This chapter presents the conclusions reached as a result of the research presented in the dissertation. The goals outlined at the beginning of the dissertation are revisited and then an overall assessment of the success of the research is stated.

Goals

The goals of this research were to develop a more 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 relation matrix in $O(n^3)$ 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.
Success of the Research

The performance of the algebra met and slightly exceeded my goals for the research. The closure algorithm consistently performed better than the worst case analysis of $O(n^3)$ and, on average, performed slightly better than $O(n^2)$. Combining the representation of interval relationships as bit codes and an interval structure capable of representing discrete time points provided the best of both temporal representation approaches. It allowed the set of actions represented by the intervals to be completely represented by the interval relationships or represented via time points set on the interval structures.

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 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.

Limitations

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 of the 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.

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 \((p \text{ m})\), of other sets in the algebra, such as \((p \text{ m o})\), which are valid. Currently the bit-encoding mechanism cannot capture these subsets uniquely.

**Future Research**

In performing the research leading to the development of the bit-encoded temporal intervals and the resulting implementation in software, there were a number of areas which warrant further exploration and research. These additional research areas fall into one of two general categories. The first is concerned with specific improvements to the representation and performance of the algebra presented in this dissertation. The second identifies opportunities and areas in which the concepts and engineering implementation presented herein could be expanded to other domains or problems. I have attempted to capture these areas and present them in the next few sections.

**Extensions to Spatial Domains**

The representational approach used for this research is based on the notion of mapping intervals which have some temporal dimension along a timeline and mapping relationships between the intervals based on relations
between the interval endpoints. These endpoint relationships are based on the concept of mapping the interval endpoints to a timeline.

The concept of mapping the endpoints to a vector denoting a singular dimension could be expanded in scope such that the general relationships and computations could be applied in a spatial domain. Thus, the endpoints of temporal intervals could be adapted in the spatial domain to represent general spatial boundary relationships between physical objects. This concept could be applied to model solid objects providing a qualitative spatial reasoning capability.

**Extending the Bit Algebra**

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 which 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 $C$. This type of assertion retains the basis of the bit-encoded representation in that it does not describe two events which are disjoint but rather it defines a specific ambiguity concerning the end point relationship of the end of $A$ and the start of $C$. 
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 endpoint 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. This might be accomplished with the following mapping.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>0100</td>
</tr>
<tr>
<td>&gt;</td>
<td>0001</td>
</tr>
<tr>
<td>=</td>
<td>0010</td>
</tr>
<tr>
<td>&lt;=&gt;</td>
<td>0111</td>
</tr>
</tbody>
</table>

This would result in an increase to 16 bits per relationship rather than the 8 bits currently needed. However, it would support the representation of subsets such as (p m). This could be represented as the bit code 0110.

More research is needed to ensure that the transitive relation computation can be extended to use this representation. The transitive closure computation, however, should be the same complexity.

**Refinement of Algorithm to Hardware Implementation**

The bit-encoded relationship lends itself to migration of the algorithm into silicon. The basic transitive computation algorithm could be implemented very easily in hardware. A specialized circuit could be developed to provide the basic computational engine in computing transitive relationships.
The key to an implementation of the algorithm is the simplicity of the method and the fact that all transitive relationships can be computed directly from the two relationships provided to the transitive computation function. No table lookup is required. Thus, a temporal adder could be implemented as a special purpose processor with an associated temporal memory.

**Parallelization of Closure Computations**

The closure computation function is a prime candidate for parallelization. Since the core of the closure algorithm iterates over the set of ambiguous relationships, this iteration is a candidate for parallel implementation. The key driver to implementing the iteration in parallel is the independence of each of the ambiguous relations during the inspection and transitive closure computation process.

This parallelization could be implemented as a set of processes, each computing the resultant transitive relation for one of the ambiguous relations. This is feasible because the transitive computation for an interval relation and each of the entries in the ambiguous relation set is mutually independent. Any dependencies surface during each cycle of the transitive closure algorithm over the changed relation queue.
APPENDICES
APPENDIX A

COMPUTED TRANSITIVITY TABLE
The following data was generated by iterating over all possible relation combinations and computing the transitive relation. The computed answer was then matched against the relation code table to identify all relationships which were represented by the computed value. Thus, an answer of (1 1 1 1 1) corresponds to the precedes relationship. However, the value (3 1 3 1) corresponds to the relation set, (during, starts, overlaps, meets, and precedes). This set is a result of matching the pattern (3 1 3 1) against all entries in the relation code table using the entries containing a three as a universal match character because of the ambiguity.

It should be noted that in the process of analyzing the resulting computed transitivity table entries, three instances were found where the set of computed transitive relations disagreed with the entries shown in Allen’s original transitivity table. These items, upon closer inspection, were found to be correct and the entries in Allen’s transitivity table were in error. The three entries were:

<table>
<thead>
<tr>
<th>Relation Pair</th>
<th>Computed Value</th>
<th>Allen’s Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0 oi)</td>
<td>(o s f d = di fi si oi)</td>
<td>(o oi d di =)</td>
</tr>
<tr>
<td>(di d)</td>
<td>(o s f d = di fi si oi)</td>
<td>(o oi d di =)</td>
</tr>
<tr>
<td>(oi o)</td>
<td>(o s f d = di fi si oi)</td>
<td>(o oi d di =)</td>
</tr>
</tbody>
</table>

The entries, as shown above, were consistent in the difference between the computed value and the transitive table value. The analysis showing that the computed values are correct is presented in the Computed Transitivity Table section of the dissertation on page 132.

\[
p p \Rightarrow (1 1 1 1) \quad \text{Relations} = (p) \\
p m \Rightarrow (1 1 1 1) \quad \text{Relations} = (p) \\
p o \Rightarrow (1 1 1 1) \quad \text{Relations} = (p) \\
p s \Rightarrow (1 1 1 1) \quad \text{Relations} = (p) \\
p f \Rightarrow (3 1 3 1) \quad \text{Relations} = (d \ s \ o \ m \ p)\]
\[
\begin{align*}
p \ d & => (3 \ 1 \ 3 \ 1) \quad \text{Relations} = (d \ s \ o \ m \ p) \\
p \ e & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
p \ di & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
p \ fi & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
p \ si & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
p \ oi & => (3 \ 1 \ 3 \ 1) \quad \text{Relations} = (d \ s \ o \ m \ p) \\
p \ mi & => (3 \ 1 \ 3 \ 1) \quad \text{Relations} = (d \ s \ o \ m \ p) \\
p \ pi & => (3 \ 3 \ 3 \ 3) \quad \text{Relations} = (p \ i \ m \ i \ o \ i \ s \ i \ f \ i \ d \ i \ e \ d \ f \ s \ o \ m \ p) \\
m \ p & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
m \ m & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
m \ o & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
m \ s & => (1 \ 1 \ 0 \ 1) \quad \text{Relations} = (m) \\
m \ f & => (3 \ 1 \ 2 \ 1) \quad \text{Relations} = (d \ s \ o) \\
m \ d & => (3 \ 1 \ 2 \ 1) \quad \text{Relations} = (d \ s \ o) \\
m \ e & => (1 \ 1 \ 0 \ 1) \quad \text{Relations} = (m) \\
m \ di & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
m \ fi & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
m \ si & => (1 \ 1 \ 0 \ 1) \quad \text{Relations} = (m) \\
m \ oi & => (3 \ 1 \ 2 \ 1) \quad \text{Relations} = (d \ s \ o) \\
m \ mi & => (3 \ 1 \ 2 \ 0) \quad \text{Relations} = (f \ i \ e \ f) \\
m \ pi & => (3 \ 3 \ 2 \ 2) \quad \text{Relations} = (p \ i \ m \ i \ o \ i \ s \ i \ f \ i \ d \ i \ e \ d \ f \ s \ o \ m \ p) \\
o \ p & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
o \ m & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
o \ o & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
o \ s & => (1 \ 1 \ 2 \ 1) \quad \text{Relations} = (o) \\
o \ f & => (3 \ 1 \ 2 \ 1) \quad \text{Relations} = (d \ s \ o) \\
o \ d & => (3 \ 1 \ 2 \ 1) \quad \text{Relations} = (d \ s \ o) \\
o \ e & => (1 \ 1 \ 2 \ 1) \quad \text{Relations} = (o) \\
o \ di & => (1 \ 1 \ 3 \ 3) \quad \text{Relations} = (f \ i \ d \ i \ o \ m \ p) \\
o \ fi & => (1 \ 1 \ 3 \ 1) \quad \text{Relations} = (o \ m \ p) \\
o \ si & => (1 \ 1 \ 2 \ 3) \quad \text{Relations} = (f \ i \ d \ i \ o) \\
o \ oi & => (3 \ 1 \ 2 \ 3) \quad \text{Relations} = (o \ i \ s \ i \ f \ i \ d \ i \ e \ d \ f \ s \ o) \\
o \ mi & => (3 \ 1 \ 2 \ 2) \quad \text{Relations} = (o \ i \ s \ i \ d) \\
o \ pi & => (3 \ 3 \ 2 \ 2) \quad \text{Relations} = (p \ i \ m \ i \ o \ i \ s \ i \ d) \\
s \ p & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
s \ m & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
s \ o & => (1 \ 1 \ 3 \ 1) \quad \text{Relations} = (o \ m \ p) \\
s \ s & => (0 \ 1 \ 2 \ 1) \quad \text{Relations} = (s) \\
s \ f & => (2 \ 1 \ 2 \ 1) \quad \text{Relations} = (d) \\
s \ d & => (2 \ 1 \ 2 \ 1) \quad \text{Relations} = (d) \\
s \ e & => (0 \ 1 \ 2 \ 1) \quad \text{Relations} = (s) \\
s \ di & => (1 \ 1 \ 3 \ 3) \quad \text{Relations} = (f \ i \ d \ i \ o \ m \ p) \\
s \ fi & => (1 \ 1 \ 3 \ 1) \quad \text{Relations} = (o \ m \ p) \\
s \ si & => (0 \ 1 \ 2 \ 3) \quad \text{Relations} = (s \ i \ e \ s) \\
s \ oi & => (2 \ 1 \ 2 \ 3) \quad \text{Relations} = (o \ i \ d \ f) \\
s \ mi & => (2 \ 0 \ 2 \ 2) \quad \text{Relations} = (m \ i) \\
s \ pi & => (2 \ 2 \ 2 \ 2) \quad \text{Relations} = (p \ i) \\
f \ p & => (1 \ 1 \ 1 \ 1) \quad \text{Relations} = (p) \\
f \ m & => (1 \ 1 \ 0 \ 1) \quad \text{Relations} = (m) \\
f \ o & => (3 \ 1 \ 2 \ 1) \quad \text{Relations} = (d \ s \ o) \\
f \ s & => (2 \ 1 \ 2 \ 1) \quad \text{Relations} = (d) \\
f \ f & => (2 \ 1 \ 2 \ 0) \quad \text{Relations} = (f) \\
f \ d & => (2 \ 1 \ 2 \ 1) \quad \text{Relations} = (d) \\
f \ e & => (2 \ 1 \ 2 \ 0) \quad \text{Relations} = (f) \\
f \ di & => (3 \ 3 \ 2 \ 2) \quad \text{Relations} = (p \ i \ m \ i \ o \ i \ s \ i \ d) \\
f \ fi & => (3 \ 1 \ 2 \ 0) \quad \text{Relations} = (f \ i \ e \ f) \\
f \ si & => (2 \ 3 \ 2 \ 2) \quad \text{Relations} = (p \ i \ m \ o) \\
f \ oi & => (2 \ 3 \ 2 \ 2) \quad \text{Relations} = (p \ i \ m \ o) \\
f \ mi & => (2 \ 2 \ 2 \ 2) \quad \text{Relations} = (p) 
\end{align*}
\]
\[
\begin{align*}
f \pi & => (2 \ 2 \ 2 \ 2) & \text{Relations} & => (\pi) \\
d \ p & => (1 \ 1 \ 1 \ 1) & \text{Relations} & => (p) \\
d \ m & => (1 \ 1 \ 1 \ 1) & \text{Relations} & => (p) \\
d \ o & => (3 \ 1 \ 3 \ 1) & \text{Relations} & => (d \ s \ o \ m \ p) \\
d \ s & => (2 \ 1 \ 2 \ 1) & \text{Relations} & => (d) \\
d \ f & => (2 \ 1 \ 2 \ 1) & \text{Relations} & => (d) \\
d \ d & => (2 \ 1 \ 2 \ 1) & \text{Relations} & => (d) \\
d \ e & => (2 \ 1 \ 2 \ 1) & \text{Relations} & => (d) \\
d \ di & => (3 \ 3 \ 3 \ 3) & \text{Relations} & => (p \ i \ m \ o \ i \ s \ i \ f \ i \ d \ i \ e \ d \ f \ s \ o \ m \ p) \\
d \ fi & => (3 \ 1 \ 3 \ 1) & \text{Relations} & => (d \ s \ o \ m \ p) \\
d \ si & => (2 \ 3 \ 2 \ 3) & \text{Relations} & => (p \ i \ m \ o \ i \ d \ f) \\
d \ oi & => (2 \ 3 \ 2 \ 3) & \text{Relations} & => (p \ i \ m \ o \ i \ d \ f) \\
d \ mi & => (2 \ 2 \ 2 \ 2) & \text{Relations} & => (p) \\
d \ pi & => (2 \ 2 \ 2 \ 2) & \text{Relations} & => (p) \\
e \ p & => (1 \ 1 \ 1 \ 1) & \text{Relations} & => (p) \\
e \ m & => (1 \ 1 \ 0 \ 1) & \text{Relations} & => (m) \\
e \ o & => (1 \ 1 \ 2 \ 1) & \text{Relations} & => (o) \\
e \ s & => (0 \ 1 \ 2 \ 1) & \text{Relations} & => (s) \\
e \ f & => (2 \ 1 \ 2 \ 0) & \text{Relations} & => (f) \\
e \ d & => (2 \ 1 \ 2 \ 1) & \text{Relations} & => (d) \\
e \ e & => (0 \ 1 \ 2 \ 0) & \text{Relations} & => (e) \\
e \ di & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (d) \\
e \ fi & => (1 \ 1 \ 2 \ 0) & \text{Relations} & => (f) \\
e \ si & => (0 \ 1 \ 2 \ 2) & \text{Relations} & => (s) \\
e \ oi & => (2 \ 1 \ 2 \ 2) & \text{Relations} & => (o) \\
e \ mi & => (2 \ 0 \ 2 \ 2) & \text{Relations} & => (m) \\
e \ pi & => (2 \ 2 \ 2 \ 2) & \text{Relations} & => (p) \\
di \ p & => (1 \ 1 \ 3 \ 3) & \text{Relations} & => (f) \\
di \ m & => (1 \ 1 \ 2 \ 3) & \text{Relations} & => (f) \\
di \ o & => (1 \ 1 \ 2 \ 3) & \text{Relations} & => (f) \\
di \ s & => (1 \ 1 \ 2 \ 3) & \text{Relations} & => (f) \\
di \ f & => (3 \ 1 \ 2 \ 2) & \text{Relations} & => (o) \\
di \ d & => (3 \ 1 \ 2 \ 3) & \text{Relations} & => (o) \\
di \ e & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (e) \\
di \ di & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (d) \\
di \ fi & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (d) \\
di \ si & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (d) \\
di \ oi & => (3 \ 1 \ 2 \ 2) & \text{Relations} & => (o) \\
di \ mi & => (3 \ 1 \ 2 \ 2) & \text{Relations} & => (o) \\
di \ pi & => (3 \ 3 \ 2 \ 2) & \text{Relations} & => (p) \\
fi \ p & => (1 \ 1 \ 1 \ 1) & \text{Relations} & => (p) \\
fi \ m & => (1 \ 1 \ 0 \ 1) & \text{Relations} & => (m) \\
fi \ o & => (1 \ 1 \ 2 \ 1) & \text{Relations} & => (o) \\
fi \ s & => (1 \ 1 \ 2 \ 1) & \text{Relations} & => (o) \\
fi \ f & => (3 \ 1 \ 2 \ 0) & \text{Relations} & => (f) \\
fi \ d & => (3 \ 1 \ 2 \ 1) & \text{Relations} & => (d) \\
fi \ e & => (1 \ 1 \ 2 \ 0) & \text{Relations} & => (f) \\
fi \ di & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (d) \\
fi \ fi & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (f) \\
fi \ si & => (1 \ 1 \ 2 \ 2) & \text{Relations} & => (d) \\
fi \ oi & => (3 \ 1 \ 2 \ 2) & \text{Relations} & => (o) \\
fi \ mi & => (3 \ 1 \ 2 \ 2) & \text{Relations} & => (o) \\
fi \ pi & => (3 \ 3 \ 2 \ 2) & \text{Relations} & => (p) \\
si \ p & => (1 \ 1 \ 3 \ 3) & \text{Relations} & => (f) \\
si \ m & => (1 \ 1 \ 2 \ 3) & \text{Relations} & => (f) \\
si \ o & => (1 \ 1 \ 2 \ 3) & \text{Relations} & => (f) \\
si \ s & => (0 \ 1 \ 2 \ 3) & \text{Relations} & => (s) \\
si \ f & => (2 \ 1 \ 2 \ 2) & \text{Relations} & => (o) \\
si \ d & => (2 \ 1 \ 2 \ 3) & \text{Relations} & => (o) 
\end{align*}
\]
si e => (0 1 2 2)  Relations = (si)
si di => (1 1 2 2)  Relations = (di)
si fi => (1 1 2 2)  Relations = (di)
si si => (0 1 2 2)  Relations = (si)
si oi => (2 1 2 2)  Relations = (oi)
si mi => (2 0 2 2)  Relations = (mi)
si pi => (2 2 2 2)  Relations = (pi)
oi p => (1 1 3 3)  Relations = (fi di o m p)
oi m => (1 1 2 3)  Relations = (fi di o)
oi o => (3 1 2 3)  Relations = (oi si fi di e d f s o)
oi s => (2 1 2 3)  Relations = (oi d f)
oi f => (2 1 2 2)  Relations = (oi)
oi d => (2 1 2 3)  Relations = (oi d f)
oi e => (2 1 2 2)  Relations = (oi)
oi di => (3 3 2 2)  Relations = (pi mi oi si di)
oi fi => (3 1 2 2)  Relations = (oi si di)
oi si => (2 3 2 2)  Relations = (pi mi oi)
oi oi => (2 3 2 2)  Relations = (pi mi oi)
oi mi => (2 2 2 2)  Relations = (pi)
oi pi => (2 2 2 2)  Relations = (pi)
m i p => (1 1 3 3)  Relations = (fi di o m p)
m i m => (0 1 2 3)  Relations = (si e s)
m i o => (2 1 2 3)  Relations = (oi d f)
m i s => (2 1 2 3)  Relations = (oi d f)
m i f => (2 0 2 2)  Relations = (mi)
m i d => (2 1 2 3)  Relations = (oi d f)
m i e => (2 0 2 2)  Relations = (mi)
m i di => (2 2 2 2)  Relations = (pi)
m i fi => (2 0 2 2)  Relations = (mi)
m i si => (2 2 2 2)  Relations = (pi)
m i oi => (2 2 2 2)  Relations = (pi)
m i mi => (2 2 2 2)  Relations = (pi)
m i pi => (2 2 2 2)  Relations = (pi)
pi p => (3 3 3 3)  Relations = (pi mi oi si fi di e d f s o m p)
pi m => (2 3 2 3)  Relations = (pi mi oi d f)
pi o => (2 3 2 3)  Relations = (pi mi oi d f)
pi s => (2 3 2 3)  Relations = (pi mi oi d f)
pi f => (2 2 2 2)  Relations = (pi)
pi d => (2 3 2 3)  Relations = (pi mi oi d f)
pi e => (2 2 2 2)  Relations = (pi)
pi di => (2 2 2 2)  Relations = (pi)
pi fi => (2 2 2 2)  Relations = (pi)
pi si => (2 2 2 2)  Relations = (pi)
pi oi => (2 2 2 2)  Relations = (pi)
pi mi => (2 2 2 2)  Relations = (pi)
pi pi => (2 2 2 2)  Relations = (pi)
APPENDIX B

NUMBER OF AMBIGUOUS RELATIONS
The data generated for the five hundred test cases of temporal systems consisting of 2 to 25 intervals with random relations asserted is shown below. The first number of each entry is the number of intervals in the system and the second number is the count of the ambiguous relations. This data was presented in graphical form in Chapter 5.

<table>
<thead>
<tr>
<th>Test #</th>
<th>No. of Intervals</th>
<th>No. of Ambiguous Relations</th>
<th>Test #</th>
<th>No. of Intervals</th>
<th>No. of Ambiguous Relations</th>
<th>Test #</th>
<th>No. of Intervals</th>
<th>No. of Ambiguous Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>51</td>
<td>3</td>
<td>0</td>
<td>101</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>52</td>
<td>3</td>
<td>0</td>
<td>102</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>53</td>
<td>3</td>
<td>0</td>
<td>103</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>54</td>
<td>3</td>
<td>0</td>
<td>104</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>55</td>
<td>3</td>
<td>0</td>
<td>105</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>56</td>
<td>3</td>
<td>0</td>
<td>106</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>57</td>
<td>3</td>
<td>0</td>
<td>107</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>58</td>
<td>4</td>
<td>1</td>
<td>108</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>59</td>
<td>4</td>
<td>0</td>
<td>109</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
<td>60</td>
<td>4</td>
<td>0</td>
<td>110</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0</td>
<td>61</td>
<td>4</td>
<td>0</td>
<td>111</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0</td>
<td>62</td>
<td>4</td>
<td>0</td>
<td>112</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>0</td>
<td>63</td>
<td>4</td>
<td>1</td>
<td>113</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0</td>
<td>64</td>
<td>4</td>
<td>0</td>
<td>114</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>0</td>
<td>65</td>
<td>4</td>
<td>0</td>
<td>115</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>0</td>
<td>66</td>
<td>4</td>
<td>1</td>
<td>116</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>0</td>
<td>67</td>
<td>4</td>
<td>0</td>
<td>117</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0</td>
<td>68</td>
<td>4</td>
<td>1</td>
<td>118</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>0</td>
<td>69</td>
<td>4</td>
<td>0</td>
<td>119</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>0</td>
<td>70</td>
<td>4</td>
<td>0</td>
<td>120</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>0</td>
<td>71</td>
<td>4</td>
<td>0</td>
<td>121</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>0</td>
<td>72</td>
<td>5</td>
<td>3</td>
<td>122</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>0</td>
<td>73</td>
<td>5</td>
<td>0</td>
<td>123</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>0</td>
<td>74</td>
<td>5</td>
<td>2</td>
<td>124</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>0</td>
<td>75</td>
<td>5</td>
<td>1</td>
<td>125</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>3</td>
<td>0</td>
<td>76</td>
<td>5</td>
<td>2</td>
<td>126</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td>3</td>
<td>0</td>
<td>77</td>
<td>5</td>
<td>3</td>
<td>127</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>3</td>
<td>0</td>
<td>78</td>
<td>5</td>
<td>3</td>
<td>128</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>3</td>
<td>0</td>
<td>79</td>
<td>5</td>
<td>2</td>
<td>129</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>0</td>
<td>80</td>
<td>5</td>
<td>2</td>
<td>130</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>0</td>
<td>81</td>
<td>5</td>
<td>3</td>
<td>131</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
<td>0</td>
<td>82</td>
<td>5</td>
<td>3</td>
<td>132</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>0</td>
<td>83</td>
<td>5</td>
<td>0</td>
<td>133</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>3</td>
<td>0</td>
<td>84</td>
<td>5</td>
<td>3</td>
<td>134</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>0</td>
<td>85</td>
<td>5</td>
<td>3</td>
<td>135</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>36</td>
<td>3</td>
<td>0</td>
<td>86</td>
<td>5</td>
<td>3</td>
<td>136</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>37</td>
<td>3</td>
<td>0</td>
<td>87</td>
<td>5</td>
<td>0</td>
<td>137</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>38</td>
<td>3</td>
<td>0</td>
<td>88</td>
<td>6</td>
<td>1</td>
<td>138</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>39</td>
<td>3</td>
<td>0</td>
<td>89</td>
<td>6</td>
<td>1</td>
<td>139</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>0</td>
<td>90</td>
<td>6</td>
<td>0</td>
<td>140</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>41</td>
<td>3</td>
<td>0</td>
<td>91</td>
<td>6</td>
<td>0</td>
<td>141</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>3</td>
<td>0</td>
<td>92</td>
<td>6</td>
<td>4</td>
<td>142</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>43</td>
<td>3</td>
<td>0</td>
<td>93</td>
<td>6</td>
<td>5</td>
<td>143</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>44</td>
<td>3</td>
<td>0</td>
<td>94</td>
<td>6</td>
<td>2</td>
<td>144</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>45</td>
<td>3</td>
<td>0</td>
<td>95</td>
<td>6</td>
<td>5</td>
<td>145</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>46</td>
<td>3</td>
<td>0</td>
<td>96</td>
<td>6</td>
<td>0</td>
<td>146</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>47</td>
<td>3</td>
<td>0</td>
<td>97</td>
<td>6</td>
<td>1</td>
<td>147</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>48</td>
<td>3</td>
<td>0</td>
<td>98</td>
<td>6</td>
<td>3</td>
<td>148</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>49</td>
<td>3</td>
<td>0</td>
<td>99</td>
<td>6</td>
<td>5</td>
<td>149</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>0</td>
<td>100</td>
<td>6</td>
<td>0</td>
<td>150</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Test #</td>
<td>No. of Intervals</td>
<td>No. of Ambiguous Relations</td>
<td>Test #</td>
<td>No. of Intervals</td>
<td>No. of Ambiguous Relations</td>
<td>Test #</td>
<td>No. of Intervals</td>
<td>No. of Ambiguous Relations</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------</td>
<td>-----------------------------</td>
<td>-------</td>
<td>-----------------</td>
<td>-----------------------------</td>
<td>-------</td>
<td>-----------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>151</td>
<td>7</td>
<td>5</td>
<td>201</td>
<td>9</td>
<td>9</td>
<td>251</td>
<td>12</td>
<td>29</td>
</tr>
<tr>
<td>152</td>
<td>7</td>
<td>2</td>
<td>202</td>
<td>9</td>
<td>12</td>
<td>252</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>153</td>
<td>7</td>
<td>7</td>
<td>203</td>
<td>9</td>
<td>19</td>
<td>253</td>
<td>12</td>
<td>34</td>
</tr>
<tr>
<td>154</td>
<td>7</td>
<td>6</td>
<td>204</td>
<td>9</td>
<td>20</td>
<td>254</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>155</td>
<td>7</td>
<td>5</td>
<td>205</td>
<td>9</td>
<td>21</td>
<td>255</td>
<td>12</td>
<td>35</td>
</tr>
<tr>
<td>156</td>
<td>7</td>
<td>4</td>
<td>206</td>
<td>9</td>
<td>14</td>
<td>256</td>
<td>12</td>
<td>28</td>
</tr>
<tr>
<td>157</td>
<td>7</td>
<td>3</td>
<td>207</td>
<td>9</td>
<td>10</td>
<td>257</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>158</td>
<td>7</td>
<td>4</td>
<td>208</td>
<td>9</td>
<td>14</td>
<td>258</td>
<td>12</td>
<td>31</td>
</tr>
<tr>
<td>159</td>
<td>7</td>
<td>0</td>
<td>209</td>
<td>9</td>
<td>20</td>
<td>259</td>
<td>12</td>
<td>31</td>
</tr>
<tr>
<td>160</td>
<td>7</td>
<td>3</td>
<td>210</td>
<td>10</td>
<td>25</td>
<td>260</td>
<td>12</td>
<td>38</td>
</tr>
<tr>
<td>161</td>
<td>7</td>
<td>4</td>
<td>211</td>
<td>10</td>
<td>13</td>
<td>261</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>162</td>
<td>7</td>
<td>8</td>
<td>212</td>
<td>10</td>
<td>21</td>
<td>262</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>163</td>
<td>7</td>
<td>7</td>
<td>213</td>
<td>10</td>
<td>10</td>
<td>263</td>
<td>12</td>
<td>34</td>
</tr>
<tr>
<td>164</td>
<td>7</td>
<td>2</td>
<td>214</td>
<td>10</td>
<td>17</td>
<td>264</td>
<td>12</td>
<td>22</td>
</tr>
<tr>
<td>165</td>
<td>7</td>
<td>4</td>
<td>215</td>
<td>10</td>
<td>20</td>
<td>265</td>
<td>12</td>
<td>35</td>
</tr>
<tr>
<td>166</td>
<td>7</td>
<td>10</td>
<td>216</td>
<td>10</td>
<td>17</td>
<td>266</td>
<td>12</td>
<td>38</td>
</tr>
<tr>
<td>167</td>
<td>7</td>
<td>2</td>
<td>217</td>
<td>10</td>
<td>22</td>
<td>267</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>168</td>
<td>7</td>
<td>7</td>
<td>218</td>
<td>10</td>
<td>18</td>
<td>268</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>169</td>
<td>7</td>
<td>0</td>
<td>219</td>
<td>10</td>
<td>17</td>
<td>269</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>170</td>
<td>7</td>
<td>3</td>
<td>220</td>
<td>10</td>
<td>25</td>
<td>270</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>171</td>
<td>8</td>
<td>3</td>
<td>221</td>
<td>10</td>
<td>14</td>
<td>271</td>
<td>12</td>
<td>29</td>
</tr>
<tr>
<td>172</td>
<td>8</td>
<td>6</td>
<td>222</td>
<td>10</td>
<td>23</td>
<td>272</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>173</td>
<td>8</td>
<td>5</td>
<td>223</td>
<td>10</td>
<td>20</td>
<td>273</td>
<td>12</td>
<td>34</td>
</tr>
<tr>
<td>174</td>
<td>8</td>
<td>4</td>
<td>224</td>
<td>10</td>
<td>19</td>
<td>274</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>175</td>
<td>8</td>
<td>10</td>
<td>225</td>
<td>10</td>
<td>13</td>
<td>275</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>176</td>
<td>8</td>
<td>9</td>
<td>226</td>
<td>10</td>
<td>21</td>
<td>276</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>177</td>
<td>8</td>
<td>5</td>
<td>227</td>
<td>10</td>
<td>10</td>
<td>277</td>
<td>12</td>
<td>35</td>
</tr>
<tr>
<td>178</td>
<td>8</td>
<td>12</td>
<td>228</td>
<td>11</td>
<td>31</td>
<td>278</td>
<td>12</td>
<td>38</td>
</tr>
<tr>
<td>179</td>
<td>8</td>
<td>14</td>
<td>229</td>
<td>11</td>
<td>31</td>
<td>279</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>180</td>
<td>8</td>
<td>5</td>
<td>230</td>
<td>11</td>
<td>28</td>
<td>280</td>
<td>12</td>
<td>33</td>
</tr>
<tr>
<td>181</td>
<td>8</td>
<td>4</td>
<td>231</td>
<td>11</td>
<td>29</td>
<td>281</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>182</td>
<td>8</td>
<td>6</td>
<td>232</td>
<td>11</td>
<td>32</td>
<td>282</td>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>183</td>
<td>8</td>
<td>14</td>
<td>233</td>
<td>11</td>
<td>27</td>
<td>283</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>184</td>
<td>8</td>
<td>3</td>
<td>234</td>
<td>11</td>
<td>23</td>
<td>284</td>
<td>12</td>
<td>39</td>
</tr>
<tr>
<td>185</td>
<td>8</td>
<td>14</td>
<td>235</td>
<td>11</td>
<td>30</td>
<td>285</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>186</td>
<td>8</td>
<td>9</td>
<td>236</td>
<td>11</td>
<td>25</td>
<td>286</td>
<td>13</td>
<td>51</td>
</tr>
<tr>
<td>187</td>
<td>8</td>
<td>5</td>
<td>237</td>
<td>11</td>
<td>32</td>
<td>287</td>
<td>13</td>
<td>48</td>
</tr>
<tr>
<td>188</td>
<td>8</td>
<td>4</td>
<td>238</td>
<td>11</td>
<td>30</td>
<td>288</td>
<td>13</td>
<td>42</td>
</tr>
<tr>
<td>189</td>
<td>8</td>
<td>6</td>
<td>239</td>
<td>11</td>
<td>30</td>
<td>289</td>
<td>13</td>
<td>45</td>
</tr>
<tr>
<td>190</td>
<td>8</td>
<td>9</td>
<td>240</td>
<td>11</td>
<td>10</td>
<td>290</td>
<td>13</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>191</td>
<td>8</td>
<td>6</td>
<td>241</td>
<td>11</td>
<td>20</td>
<td>291</td>
<td>13</td>
<td>48</td>
</tr>
<tr>
<td>192</td>
<td>9</td>
<td>19</td>
<td>242</td>
<td>11</td>
<td>33</td>
<td>292</td>
<td>13</td>
<td>44</td>
</tr>
<tr>
<td>193</td>
<td>9</td>
<td>15</td>
<td>243</td>
<td>11</td>
<td>25</td>
<td>293</td>
<td>13</td>
<td>48</td>
</tr>
<tr>
<td>194</td>
<td>9</td>
<td>20</td>
<td>244</td>
<td>11</td>
<td>23</td>
<td>294</td>
<td>13</td>
<td>41</td>
</tr>
<tr>
<td>195</td>
<td>9</td>
<td>19</td>
<td>245</td>
<td>11</td>
<td>26</td>
<td>295</td>
<td>13</td>
<td>37</td>
</tr>
<tr>
<td>196</td>
<td>9</td>
<td>20</td>
<td>246</td>
<td>11</td>
<td>32</td>
<td>296</td>
<td>13</td>
<td>50</td>
</tr>
<tr>
<td>197</td>
<td>9</td>
<td>17</td>
<td>247</td>
<td>11</td>
<td>23</td>
<td>297</td>
<td>13</td>
<td>54</td>
</tr>
<tr>
<td>198</td>
<td>9</td>
<td>20</td>
<td>248</td>
<td>11</td>
<td>22</td>
<td>298</td>
<td>13</td>
<td>42</td>
</tr>
<tr>
<td>199</td>
<td>9</td>
<td>19</td>
<td>249</td>
<td>11</td>
<td>31</td>
<td>299</td>
<td>13</td>
<td>50</td>
</tr>
<tr>
<td>200</td>
<td>9</td>
<td>9</td>
<td>250</td>
<td>11</td>
<td>31</td>
<td>300</td>
<td>14</td>
<td>60</td>
</tr>
<tr>
<td>Test #</td>
<td>No. of Intervals</td>
<td>No. of Ambiguous Relations</td>
<td>Test #</td>
<td>No. of Intervals</td>
<td>No. of Ambiguous Relations</td>
<td>Test #</td>
<td>No. of Intervals</td>
<td>No. of Ambiguous Relations</td>
</tr>
<tr>
<td>--------</td>
<td>------------------</td>
<td>----------------------------</td>
<td>--------</td>
<td>------------------</td>
<td>----------------------------</td>
<td>--------</td>
<td>------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>301</td>
<td>14</td>
<td>34</td>
<td>351</td>
<td>16</td>
<td>73</td>
<td>401</td>
<td>17</td>
<td>79</td>
</tr>
<tr>
<td>302</td>
<td>14</td>
<td>43</td>
<td>352</td>
<td>16</td>
<td>61</td>
<td>402</td>
<td>17</td>
<td>66</td>
</tr>
<tr>
<td>303</td>
<td>14</td>
<td>60</td>
<td>353</td>
<td>16</td>
<td>75</td>
<td>403</td>
<td>17</td>
<td>87</td>
</tr>
<tr>
<td>304</td>
<td>14</td>
<td>62</td>
<td>354</td>
<td>16</td>
<td>65</td>
<td>404</td>
<td>17</td>
<td>73</td>
</tr>
<tr>
<td>305</td>
<td>14</td>
<td>53</td>
<td>355</td>
<td>16</td>
<td>77</td>
<td>405</td>
<td>17</td>
<td>93</td>
</tr>
<tr>
<td>306</td>
<td>14</td>
<td>37</td>
<td>356</td>
<td>16</td>
<td>50</td>
<td>406</td>
<td>17</td>
<td>69</td>
</tr>
<tr>
<td>307</td>
<td>14</td>
<td>62</td>
<td>357</td>
<td>16</td>
<td>80</td>
<td>407</td>
<td>17</td>
<td>87</td>
</tr>
<tr>
<td>308</td>
<td>14</td>
<td>52</td>
<td>358</td>
<td>16</td>
<td>69</td>
<td>408</td>
<td>17</td>
<td>93</td>
</tr>
<tr>
<td>309</td>
<td>14</td>
<td>48</td>
<td>359</td>
<td>16</td>
<td>84</td>
<td>409</td>
<td>17</td>
<td>69</td>
</tr>
<tr>
<td>310</td>
<td>14</td>
<td>51</td>
<td>360</td>
<td>16</td>
<td>76</td>
<td>410</td>
<td>17</td>
<td>70</td>
</tr>
<tr>
<td>311</td>
<td>14</td>
<td>42</td>
<td>361</td>
<td>16</td>
<td>63</td>
<td>411</td>
<td>17</td>
<td>89</td>
</tr>
<tr>
<td>312</td>
<td>14</td>
<td>58</td>
<td>362</td>
<td>16</td>
<td>82</td>
<td>412</td>
<td>17</td>
<td>99</td>
</tr>
<tr>
<td>313</td>
<td>14</td>
<td>56</td>
<td>363</td>
<td>16</td>
<td>81</td>
<td>413</td>
<td>17</td>
<td>85</td>
</tr>
<tr>
<td>314</td>
<td>14</td>
<td>36</td>
<td>364</td>
<td>16</td>
<td>80</td>
<td>414</td>
<td>17</td>
<td>85</td>
</tr>
<tr>
<td>315</td>
<td>14</td>
<td>35</td>
<td>365</td>
<td>16</td>
<td>76</td>
<td>415</td>
<td>17</td>
<td>83</td>
</tr>
<tr>
<td>316</td>
<td>14</td>
<td>52</td>
<td>366</td>
<td>16</td>
<td>80</td>
<td>416</td>
<td>18</td>
<td>95</td>
</tr>
<tr>
<td>317</td>
<td>14</td>
<td>43</td>
<td>367</td>
<td>16</td>
<td>62</td>
<td>417</td>
<td>18</td>
<td>100</td>
</tr>
<tr>
<td>318</td>
<td>14</td>
<td>57</td>
<td>368</td>
<td>16</td>
<td>73</td>
<td>418</td>
<td>18</td>
<td>97</td>
</tr>
<tr>
<td>319</td>
<td>14</td>
<td>39</td>
<td>369</td>
<td>16</td>
<td>78</td>
<td>419</td>
<td>18</td>
<td>107</td>
</tr>
<tr>
<td>320</td>
<td>15</td>
<td>62</td>
<td>370</td>
<td>16</td>
<td>84</td>
<td>420</td>
<td>18</td>
<td>75</td>
</tr>
<tr>
<td>321</td>
<td>15</td>
<td>63</td>
<td>371</td>
<td>16</td>
<td>70</td>
<td>421</td>
<td>18</td>
<td>103</td>
</tr>
<tr>
<td>322</td>
<td>15</td>
<td>71</td>
<td>372</td>
<td>16</td>
<td>80</td>
<td>422</td>
<td>18</td>
<td>105</td>
</tr>
<tr>
<td>323</td>
<td>15</td>
<td>62</td>
<td>373</td>
<td>16</td>
<td>74</td>
<td>423</td>
<td>18</td>
<td>102</td>
</tr>
<tr>
<td>324</td>
<td>15</td>
<td>70</td>
<td>374</td>
<td>16</td>
<td>54</td>
<td>424</td>
<td>18</td>
<td>95</td>
</tr>
<tr>
<td>325</td>
<td>15</td>
<td>72</td>
<td>375</td>
<td>16</td>
<td>76</td>
<td>425</td>
<td>18</td>
<td>105</td>
</tr>
<tr>
<td>326</td>
<td>15</td>
<td>60</td>
<td>376</td>
<td>16</td>
<td>54</td>
<td>426</td>
<td>18</td>
<td>94</td>
</tr>
<tr>
<td>327</td>
<td>15</td>
<td>47</td>
<td>377</td>
<td>16</td>
<td>77</td>
<td>427</td>
<td>18</td>
<td>98</td>
</tr>
<tr>
<td>328</td>
<td>15</td>
<td>67</td>
<td>378</td>
<td>16</td>
<td>70</td>
<td>428</td>
<td>18</td>
<td>114</td>
</tr>
<tr>
<td>329</td>
<td>15</td>
<td>62</td>
<td>379</td>
<td>16</td>
<td>82</td>
<td>429</td>
<td>18</td>
<td>105</td>
</tr>
<tr>
<td>330</td>
<td>15</td>
<td>72</td>
<td>380</td>
<td>16</td>
<td>73</td>
<td>430</td>
<td>18</td>
<td>94</td>
</tr>
<tr>
<td>331</td>
<td>15</td>
<td>66</td>
<td>381</td>
<td>16</td>
<td>77</td>
<td>431</td>
<td>18</td>
<td>106</td>
</tr>
<tr>
<td>332</td>
<td>15</td>
<td>60</td>
<td>382</td>
<td>16</td>
<td>72</td>
<td>432</td>
<td>18</td>
<td>104</td>
</tr>
<tr>
<td>333</td>
<td>15</td>
<td>65</td>
<td>383</td>
<td>16</td>
<td>58</td>
<td>433</td>
<td>18</td>
<td>100</td>
</tr>
<tr>
<td>334</td>
<td>15</td>
<td>44</td>
<td>384</td>
<td>17</td>
<td>77</td>
<td>434</td>
<td>18</td>
<td>78</td>
</tr>
<tr>
<td>335</td>
<td>15</td>
<td>63</td>
<td>385</td>
<td>17</td>
<td>86</td>
<td>435</td>
<td>19</td>
<td>112</td>
</tr>
<tr>
<td>336</td>
<td>15</td>
<td>42</td>
<td>386</td>
<td>17</td>
<td>81</td>
<td>436</td>
<td>19</td>
<td>106</td>
</tr>
<tr>
<td>337</td>
<td>15</td>
<td>56</td>
<td>387</td>
<td>17</td>
<td>78</td>
<td>437</td>
<td>19</td>
<td>119</td>
</tr>
<tr>
<td>338</td>
<td>15</td>
<td>70</td>
<td>388</td>
<td>17</td>
<td>84</td>
<td>438</td>
<td>19</td>
<td>105</td>
</tr>
<tr>
<td>339</td>
<td>15</td>
<td>47</td>
<td>389</td>
<td>17</td>
<td>93</td>
<td>439</td>
<td>19</td>
<td>109</td>
</tr>
<tr>
<td>340</td>
<td>15</td>
<td>71</td>
<td>390</td>
<td>17</td>
<td>90</td>
<td>440</td>
<td>19</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>341</td>
<td>15</td>
<td>60</td>
<td>391</td>
<td>17</td>
<td>76</td>
<td>441</td>
<td>19</td>
<td>119</td>
</tr>
<tr>
<td>342</td>
<td>15</td>
<td>74</td>
<td>392</td>
<td>17</td>
<td>86</td>
<td>442</td>
<td>19</td>
<td>127</td>
</tr>
<tr>
<td>343</td>
<td>15</td>
<td>61</td>
<td>393</td>
<td>17</td>
<td>65</td>
<td>443</td>
<td>19</td>
<td>75</td>
</tr>
<tr>
<td>344</td>
<td>15</td>
<td>72</td>
<td>394</td>
<td>17</td>
<td>86</td>
<td>444</td>
<td>19</td>
<td>120</td>
</tr>
<tr>
<td>345</td>
<td>15</td>
<td>71</td>
<td>395</td>
<td>17</td>
<td>84</td>
<td>445</td>
<td>19</td>
<td>125</td>
</tr>
<tr>
<td>346</td>
<td>16</td>
<td>77</td>
<td>396</td>
<td>17</td>
<td>93</td>
<td>446</td>
<td>19</td>
<td>122</td>
</tr>
<tr>
<td>347</td>
<td>16</td>
<td>77</td>
<td>397</td>
<td>17</td>
<td>83</td>
<td>447</td>
<td>19</td>
<td>113</td>
</tr>
<tr>
<td>348</td>
<td>16</td>
<td>64</td>
<td>398</td>
<td>17</td>
<td>94</td>
<td>448</td>
<td>19</td>
<td>126</td>
</tr>
<tr>
<td>349</td>
<td>16</td>
<td>70</td>
<td>399</td>
<td>17</td>
<td>96</td>
<td>449</td>
<td>19</td>
<td>112</td>
</tr>
<tr>
<td>350</td>
<td>16</td>
<td>51</td>
<td>400</td>
<td>17</td>
<td>85</td>
<td>450</td>
<td>19</td>
<td>115</td>
</tr>
<tr>
<td>Test #</td>
<td>No. of Intervals</td>
<td>No. of Ambiguous Relations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>-----------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>451</td>
<td>19</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452</td>
<td>19</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>453</td>
<td>19</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>454</td>
<td>19</td>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>455</td>
<td>20</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>456</td>
<td>20</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>457</td>
<td>20</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>458</td>
<td>20</td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>459</td>
<td>20</td>
<td>129</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>460</td>
<td>20</td>
<td>112</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>461</td>
<td>20</td>
<td>107</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>462</td>
<td>20</td>
<td>131</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>463</td>
<td>20</td>
<td>127</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>464</td>
<td>20</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>465</td>
<td>20</td>
<td>133</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>466</td>
<td>20</td>
<td>114</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>467</td>
<td>20</td>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>468</td>
<td>20</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>469</td>
<td>20</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>470</td>
<td>21</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>471</td>
<td>21</td>
<td>133</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>472</td>
<td>21</td>
<td>157</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>473</td>
<td>21</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>474</td>
<td>21</td>
<td>147</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>475</td>
<td>21</td>
<td>144</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>476</td>
<td>21</td>
<td>152</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>477</td>
<td>21</td>
<td>114</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>478</td>
<td>21</td>
<td>146</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>479</td>
<td>21</td>
<td>149</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>480</td>
<td>21</td>
<td>137</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>481</td>
<td>21</td>
<td>139</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>482</td>
<td>21</td>
<td>142</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>483</td>
<td>21</td>
<td>148</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>484</td>
<td>21</td>
<td>142</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>485</td>
<td>21</td>
<td>146</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>486</td>
<td>21</td>
<td>158</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>487</td>
<td>21</td>
<td>159</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>488</td>
<td>21</td>
<td>143</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>489</td>
<td>21</td>
<td>132</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>490</td>
<td>21</td>
<td>161</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>491</td>
<td>21</td>
<td>130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>492</td>
<td>21</td>
<td>164</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>493</td>
<td>21</td>
<td>152</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>494</td>
<td>21</td>
<td>145</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>495</td>
<td>21</td>
<td>148</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>496</td>
<td>21</td>
<td>141</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>497</td>
<td>21</td>
<td>155</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>498</td>
<td>21</td>
<td>151</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>499</td>
<td>21</td>
<td>151</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>21</td>
<td>145</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C

CONSTRUCTION OF A TWELVE INTERVAL TEMPORAL SYSTEM
This appendix presents the construction of a twelve interval temporal system. The listing that follows was captured as a dribble file from the LISP interpreter. The example shown illustrates the stepwise construction of the matrix of relations by interactively adding a single interval relation at a time to the system.

It also provides an illustrative view of the algorithm processing via descriptive comments printed during different sections of the program. Specific items of interest within the listing are discussed in Chapter Five.

System initialized.
T
> Relation asserted (1 1 2 1) => (0, 1)
Computing column for (0, 1)
(1 1 2 1)
> Relation asserted (1 1 2 1) => (0, 2)
Computing column for (0, 2)
Computing transitive relation over 1 -> 0 -> 2
(1 1 2 1)
> Relation asserted (1 1 0 1) => (2, 3)
Computing column for (2, 3)
Computing transitive relation over 0 -> 2 -> 3
Computing transitive relation over 1 -> 2 -> 3
(1 1 0 1)
> 2 ambiguous relations:
((1 3) (1 2))

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>(----)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>01</td>
<td>(2122)</td>
<td>(----)</td>
<td>(3123)</td>
<td>(1133)</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>02</td>
<td>(2122)</td>
<td>(3123)</td>
<td>(----)</td>
<td>(1101)</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>03</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2022)</td>
<td>(----)</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>04</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>05</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>06</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>07</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>08</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>09</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>10</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>11</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
</tbody>
</table>
NIL
> Relation asserted (1 1 1 1) => (2, 4)
Computing column for (2, 4)
Computing transitive relation over 0 -> 2 -> 4
Computing transitive relation over 1 -> 2 -> 4
Computing transitive relation over 3 -> 2 -> 4
(1 1 1 1)
> 4 ambiguous relations:
NIL

> Relation asserted (2 1 2 1) => (3, 5)
Computing column for (3, 5)
Computing transitive relation over 0 -> 3 -> 5
Computing transitive relation over 1 -> 3 -> 5
Computing transitive relation over 2 -> 3 -> 5
Computing transitive relation over 4 -> 3 -> 5
(2 1 2 1)
> 8 ambiguous relations:
((4 5) (2 5) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))
NIL
> Relation asserted (1 1 0 1) => (4, 7)
Computing column for (4, 7)
Computing transitive relation over 0 -> 4 -> 7
Computing transitive relation over 1 -> 4 -> 7
Computing transitive relation over 2 -> 4 -> 7
Computing transitive relation over 3 -> 4 -> 7
Computing transitive relation over 5 -> 4 -> 7
Computing transitive relation over 6 -> 4 -> 7
(1 1 0 1)

> 17 ambiguous relations:
((6 7) (5 7) (3 7) (1 7) (4 6) (3 6) (2 6) (1 6) (0 6) (4 5) (2 5) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))

NIL
> Relation asserted (1 1 2 0) => (7, 8)
Computing column for (7, 8)
Computing transitive relation over 0 -> 7 -> 8
Computing transitive relation over 1 -> 7 -> 8
Computing transitive relation over 2 -> 7 -> 8
Computing transitive relation over 3 -> 7 -> 8
Computing transitive relation over 4 -> 7 -> 8
Computing transitive relation over 5 -> 7 -> 8
Computing transitive relation over 6 -> 7 -> 8
(1 1 2 0)

> 21 ambiguous relations:
((6 8) (5 8) (3 8) (1 8) (6 7) (5 7) (3 7) (1 7) (4 6) (3 6) (2 6) (1 6) (0 6) (4 5) (2 5) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))
NIL
> Relation asserted (2 1 2 0) ⇒ (0, 9)
Computing column for (0, 9)
Computing transitive relation over 1 → 0 → 9
Computing transitive relation over 2 → 0 → 9
Computing transitive relation over 3 → 0 → 9
Computing transitive relation over 4 → 0 → 9
Computing transitive relation over 5 → 0 → 9
Computing transitive relation over 6 → 0 → 9
Computing transitive relation over 7 → 0 → 9
Computing transitive relation over 8 → 0 → 9
(2 1 2 0)
> 23 ambiguous relations:
((6 9) (5 9) (6 8) (5 8) (3 8) (1 8) (6 7) (5 7) (3 7) (1 7) (4 6) (3 6) (2 6) (1 6) (0 6) (4 5) (2 5) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))
00 01 02 03 04 05 06 07 08 09 10 11
00 (----) (1121) (1121) (1111) (1111) (3131) (3333) (1111) (1111) (2120) ( )
01 (2122) (----) (3123) (1133) (1133) (3133) (3333) (1133) (1133) (2122) ( )
02 (2122) (3123) (----) (1101) (1111) (3121) (3322) (1111) (1111) (2122) ( )
03 (2222) (2323) (2022) (----) (1133) (2121) (2322) (1133) (1133) (2222) ( )
04 (2222) (2323) (2222) (2323) (----) (2323) (2323) (1101) (1111) (2222) ( )
05 (3322) (3323) (3122) (1122) (1133) (----) (2122) (1133) (1133) (3322) ( )
06 (3333) (3333) (3133) (1133) (1133) (1121) (----) (1133) (1133) (3333) ( )
07 (2222) (2323) (2222) (2323) (2022) (2323) (2323) (----) (1120) (2222) ( )
08 (2222) (2323) (2222) (2323) (2222) (2323) (2323) (2120) (----) (2222) ( )
09 (1120) (1121) (1121) (1111) (1111) (1313) (3333) (1111) (1111) (----) ( )
10 ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
11 ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
NIL
> Relation asserted (2 1 2 1) ⇒ (7, 10)
Computing column for (7, 10)
Computing transitive relation over 0 → 7 → 10
Computing transitive relation over 1 → 7 → 10
Computing transitive relation over 2 → 7 → 10
Computing transitive relation over 3 → 7 → 10
Computing transitive relation over 4 → 7 → 10
Computing transitive relation over 5 → 7 → 10
Computing transitive relation over 6 → 7 → 10
Computing transitive relation over 7 → 7 → 10
Computing transitive relation over 8 → 7 → 10
Computing transitive relation over 9 → 7 → 10
(2 1 2 1)
> 31 ambiguous relations:
((9 10) (6 10) (5 10) (4 10) (3 10) (2 10) (1 10) (0 10) (6 9) (5 9) (6 8) (5 8) (3 8) (1 8) (6 7) (5 7) (3 7) (1 7) (4 6) (3 6) (2 6) (1 6) (0 6) (4 5) (2 5) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))
00 01 02 03 04 05 06 07 08 09 10 11
00 (----) (1121) (1121) (1111) (1111) (3131) (3333) (1111) (1111) (2120) (3131) ( )
01 (2122) (----) (3123) (1133) (1133) (3133) (3333) (1133) (1133) (2122) (3133) ( )
02 (2122) (3123) (----) (1101) (1111) (3121) (3322) (1111) (1111) (2122) (3131) ( )
03 (2222) (2323) (2022) (----) (1133) (2121) (2322) (1133) (1133) (2222) (3133) ( )
04 (2222) (2323) (2222) (2323) (----) (2323) (2323) (1101) (1111) (2222) (3121) ( )
05 (3322) (3323) (3122) (1122) (1133) (----) (2122) (1133) (1133) (3322) (3133) ( )
06 (3333) (3333) (3133) (1133) (1133) (1121) (----) (1133) (1133) (3333) ( )
07 (2222) (2323) (2222) (2323) (2022) (2323) (2323) (----) (1120) (2222) (3121) ( )
08 (2222) (2323) (2222) (2323) (2222) (2323) (2323) (2120) (----) (2222) (3121) ( )
09 (3322) (3323) (3122) (1122) (1133) (----) (2122) (1133) (1133) (3322) (3133) ( )
10 ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
11 ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
NIL
NIL
> Relation asserted (0 1 2 1) => (10, 11)
Computing column for (10, 11)
Computing transitive relation over 0 -> 10 -> 11
Computing transitive relation over 1 -> 10 -> 11
Computing transitive relation over 2 -> 10 -> 11
Computing transitive relation over 3 -> 10 -> 11
Computing transitive relation over 4 -> 10 -> 11
Computing transitive relation over 5 -> 10 -> 11
Computing transitive relation over 6 -> 10 -> 11
Computing transitive relation over 7 -> 10 -> 11
Computing transitive relation over 8 -> 10 -> 11
Computing transitive relation over 9 -> 10 -> 11
(0 1 2 1)
> 39 ambiguous relations:
((9 11) (6 11) (5 11) (4 11) (3 11) (2 11) (1 11) (0 11) (9 10) (6 10) (5 10) (4 10) (3 10) (2 10) (1 10) (0 10) (6 9) (5 9) (6 8) (5 8) (3 8) (1 8)
(6 7) (5 7) (3 7) (1 7) (4 6) (3 6) (2 6) (1 6) (0 6) (4 5) (2 5) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))

NIL
>
Relation asserted (2 1 2 1) => (2, 5)
Checking ambiguous relations for (2 5)
Checking ambiguous relation (1 2)
Computing transitive relation over 1 -> 5 -> 2
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 4)
Checking ambiguous relation (3 4)
Checking ambiguous relation (0 5)
Computing transitive relation over 0 -> 2 -> 5
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 2 -> 5
Checking ambiguous relation (4 5)
Computing transitive relation over 4 -> 2 -> 5
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 6)
Checking ambiguous relation (2 6)
Computing transitive relation over 2 -> 5 -> 6
Checking ambiguous relation (3 6)
Checking ambiguous relation (4 6)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 7)
Checking ambiguous relation (5 7)
Computing transitive relation over 5 -> 2 -> 7
Checking ambiguous relation (6 7)
Checking ambiguous relation (1 8)
Checking ambiguous relation (3 8)
Checking ambiguous relation (5 8)
Computing transitive relation over 5 -> 2 -> 8
Checking ambiguous relation (6 8)
Checking ambiguous relation (5 9)
Computing transitive relation over 5 -> 2 -> 9
Checking ambiguous relation (6 9)
Checking ambiguous relation (0 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (2 10)
Computing transitive relation over 2 -> 5 -> 10
Checking ambiguous relation (3 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (5 10)
Computing transitive relation over 5 -> 2 -> 10
Checking ambiguous relation (6 10)
Checking ambiguous relation (9 10)
Checking ambiguous relation (0 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (2 11)
Computing transitive relation over 2 -> 5 -> 11
Checking ambiguous relation (3 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (5 11)
Computing transitive relation over 5 -> 2 -> 11
Checking ambiguous relation (6 11)
Checking ambiguous relation (9 11)
The evaluation took 36.75 seconds.
(2 1 2 1)
> 38 ambiguous relations:
((1 2) (1 3) (1 4) (3 4) (0 5) (1 5) (4 5) (0 6) (1 6) (2 6) (3 6) (4 6) (1 7) (3 7) (5 7) (6 7) (1 8) (3 8) (5 8) (6 8) (5 9) (6 9) (0 10) (1 10) (2 10) (3 10) (4 10) (5 10) (6 10) (9 10) (0 11) (1 11) (2 11) (3 11) (4 11) (5 11) (6 11) (9 11))

NIL
> Relation asserted (2 1 2 0) => (4, 6)
Checking ambiguous relations for (4 6)
Checking ambiguous relation (9 11)
Checking ambiguous relation (6 11)
Computing transitive relation over 6 -> 4 -> 11
Checking ambiguous relation (5 11)
Checking ambiguous relation (4 11)
Computing transitive relation over 4 -> 6 -> 11
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (6 10)
Computing transitive relation over 6 -> 4 -> 10
Checking ambiguous relation (5 10)
Checking ambiguous relation (4 10)
Computing transitive relation over 4 -> 6 -> 10
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Computing transitive relation over 6 -> 4 -> 9
Checking ambiguous relation (5 9)
Checking ambiguous relation (6 8)
Computing transitive relation over 6 -> 4 -> 8
Checking ambiguous relation (5 8)
Checking ambiguous relation (3 8)
Checking ambiguous relation (1 8)
Checking ambiguous relation (6 7)
Computing transitive relation over 6 -> 4 -> 7
Checking ambiguous relation (5 7)
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 4 -> 6
Checking ambiguous relation (2 6)
Computing transitive relation over 2 -> 4 -> 6
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 4 -> 6
Checking ambiguous relation (0 6)
Computing transitive relation over 0 -> 4 -> 6
Checking ambiguous relation (4 5)
Computing transitive relation over 4 -> 6 -> 5
Checking ambiguous relation (1 5)
Checking ambiguous relation (0 5)
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 6 -> 4
Checking ambiguous relation (1 4)
Computing transitive relation over 1 -> 6 -> 4
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
Checking ambiguous relations for (4 5)
Checking ambiguous relation (9 11)
Checking ambiguous relation (6 11)
Checking ambiguous relation (5 11)
Computing transitive relation over 5 -> 4 -> 11
Checking ambiguous relation (4 11)
Computing transitive relation over 4 -> 5 -> 11
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (6 10)
Checking ambiguous relation (5 10)
Computing transitive relation over 5 -> 4 -> 10
Checking ambiguous relation (4 10)
Computing transitive relation over 4 -> 5 -> 10
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Checking ambiguous relation (5 9)
Computing transitive relation over 5 -> 4 -> 9
Checking ambiguous relation (5 8)
Computing transitive relation over 5 -> 4 -> 8
Checking ambiguous relation (3 8)
Checking ambiguous relation (1 8)
Checking ambiguous relation (5 7)
Computing transitive relation over 5 -> 4 -> 7
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (1 6)
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 4 -> 5
Checking ambiguous relation (0 5)
Computing transitive relation over 0 -> 4 -> 5
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 5 -> 4
Checking ambiguous relation (1 4)
Computing transitive relation over 1 -> 5 -> 4
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
Checking ambiguous relations for (2 6)
Checking ambiguous relation (9 11)
Checking ambiguous relation (6 11)
Computing transitive relation over 6 -> 2 -> 11
Checking ambiguous relation (5 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Computing transitive relation over 2 -> 6 -> 11
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (6 10)
Computing transitive relation over 6 -> 2 -> 10
Checking ambiguous relation (5 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Computing transitive relation over 2 -> 6 -> 10
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Computing transitive relation over 6 -> 2 -> 9
Checking ambiguous relation (5 9)
Checking ambiguous relation (5 8)
Checking ambiguous relation (3 8)
Checking ambiguous relation (1 8)
Checking ambiguous relation (5 7)
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 6)
Computing transitive relation over 3 --> 2 --> 6
Checking ambiguous relation (1 6)
Computing transitive relation over 1 --> 2 --> 6
Checking ambiguous relation (0 6)
Computing transitive relation over 0 --> 2 --> 6
Checking ambiguous relation (1 5)
Checking ambiguous relation (0 5)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 4)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
Computing transitive relation over 1 --> 6 --> 2
Checking ambiguous relations for (6 7)
Checking ambiguous relation (9 11)
Checking ambiguous relation (6 11)
Computing transitive relation over 6 --> 7 --> 11
Checking ambiguous relation (5 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (6 10)
Computing transitive relation over 6 --> 7 --> 10
Checking ambiguous relation (5 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Computing transitive relation over 6 --> 7 --> 9
Checking ambiguous relation (5 9)
Checking ambiguous relation (5 8)
Checking ambiguous relation (3 8)
Checking ambiguous relation (1 8)
Checking ambiguous relation (5 7)
Computing transitive relation over 5 --> 6 --> 7
Checking ambiguous relation (3 7)
Computing transitive relation over 3 --> 6 --> 7
Checking ambiguous relation (1 7)
Computing transitive relation over 1 --> 6 --> 7
Checking ambiguous relation (3 6)
Computing transitive relation over 3 --> 7 --> 6
Checking ambiguous relation (1 6)
Computing transitive relation over 1 --> 7 --> 6
Checking ambiguous relation (0 6)
Computing transitive relation over 0 --> 7 --> 6
Checking ambiguous relation (1 5)
Checking ambiguous relation (0 5)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 4)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
Checking ambiguous relations for (6 8)
Checking ambiguous relation (9 11)
Checking ambiguous relation (6 11)
Computing transitive relation over 6 -> 8 -> 11
Checking ambiguous relation (5 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (6 10)
Computing transitive relation over 6 -> 8 -> 10
Checking ambiguous relation (5 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (6 9)
Computing transitive relation over 6 -> 8 -> 9
Checking ambiguous relation (5 9)
Checking ambiguous relation (5 8)
Computing transitive relation over 5 -> 6 -> 8
Checking ambiguous relation (3 8)
Computing transitive relation over 3 -> 6 -> 8
Checking ambiguous relation (1 8)
Computing transitive relation over 1 -> 6 -> 8
Checking ambiguous relation (5 7)
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 6 -> 6
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 8 -> 6
Checking ambiguous relation (0 6)
Computing transitive relation over 0 -> 8 -> 6
Checking ambiguous relation (1 5)
Checking ambiguous relation (0 5)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 4)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
The evaluation took 163.46 seconds.
(2 1 2 0)
> 33 ambiguous relations:
((9 11) (6 11) (5 11) (4 11) (3 11) (2 11) (1 11) (0 11) (9 10) (6 10) (5 10) (4 10) (3 10) (2 10) (1 10) (0 10) (6 9) (5 9) (5 8) (3 8) (1 8) (5 7) (3 7) (1 7) (3 6) (1 6) (0 6) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))
NIL

> Relation asserted (1 1 2 0) => (5, 11)
Checking ambiguous relations for (5 11)
Checking ambiguous relation (1 2)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 4)
Checking ambiguous relation (3 4)
Checking ambiguous relation (0 5)
Computing transitive relation over 0 -> 11 -> 5
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 11 -> 5
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 6)
Checking ambiguous relation (3 6)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 7)
Checking ambiguous relation (5 7)
Computing transitive relation over 5 -> 11 -> 7
Checking ambiguous relation (1 8)
Checking ambiguous relation (3 8)
Checking ambiguous relation (5 8)
Computing transitive relation over 5 -> 11 -> 8
Checking ambiguous relation (5 9)
Computing transitive relation over 5 -> 11 -> 9
Checking ambiguous relation (6 9)
Checking ambiguous relation (0 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (5 10)
Computing transitive relation over 5 -> 11 -> 10
Checking ambiguous relation (6 10)
Checking ambiguous relation (9 10)
Checking ambiguous relation (0 11)
Computing transitive relation over 0 -> 5 -> 11
Checking ambiguous relation (1 11)
Computing transitive relation over 1 -> 5 -> 11
Checking ambiguous relation (2 11)
Computing transitive relation over 2 -> 5 -> 11
Checking ambiguous relation (3 11)
Computing transitive relation over 3 -> 5 -> 11
Checking ambiguous relation (4 11)
Computing transitive relation over 4 -> 5 -> 11
Checking ambiguous relation (6 11)
Computing transitive relation over 6 -> 5 -> 11
Checking ambiguous relation (9 11)
Computing transitive relation over 9 -> 5 -> 11
Checking ambiguous relations for (6 11)
Checking ambiguous relation (1 2)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 4)
Checking ambiguous relation (3 4)
Checking ambiguous relation (0 5)
Checking ambiguous relation (1 5)
Checking ambiguous relation (0 6)
Computing transitive relation over 0 -> 11 -> 6
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 11 -> 6
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 11 -> 6
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 8)
Checking ambiguous relation (3 8)
Checking ambiguous relation (5 9)
Checking ambiguous relation (6 9)
Computing transitive relation over 6 -> 11 -> 9
Checking ambiguous relation (0 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (6 10)
Computing transitive relation over 6 -> 11 -> 10
Checking ambiguous relation (9 10)
Checking ambiguous relation (0 11)
Computing transitive relation over 0 -> 6 -> 11
Checking ambiguous relation (1 11)
Computing transitive relation over 1 -> 6 -> 11
Checking ambiguous relation (2 11)
Computing transitive relation over 2 -> 6 -> 11
Checking ambiguous relation (3 11)
Computing transitive relation over 3 -> 6 -> 11
Checking ambiguous relation (4 11)
Computing transitive relation over 4 -> 6 -> 11
Checking ambiguous relation (9 11)
Computing transitive relation over 9 -> 6 -> 11
Checking ambiguous relations for (6 10)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Computing transitive relation over 9 -> 6 -> 10
Checking ambiguous relation (4 10)
Computing transitive relation over 4 -> 6 -> 10
Checking ambiguous relation (3 10)
Computing transitive relation over 3 -> 6 -> 10
Checking ambiguous relation (2 10)
Computing transitive relation over 2 -> 6 -> 10
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 6 -> 10
Checking ambiguous relation (0 10)
Computing transitive relation over 0 -> 6 -> 10
Checking ambiguous relation (6 9)
Computing transitive relation over 6 -> 10 -> 9
Checking ambiguous relation (5 9)
Checking ambiguous relation (3 8)
Checking ambiguous relation (1 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 10 -> 6
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 10 -> 6
Checking ambiguous relation (0 6)
Computing transitive relation over 0 -> 10 -> 6
Checking ambiguous relation (1 5)
Checking ambiguous relation (0 5)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 4)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
Checking ambiguous relations for (5 10)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Computing transitive relation over 9 -> 5 -> 10
Checking ambiguous relation (4 10)
Computing transitive relation over 4 -> 5 -> 10
Checking ambiguous relation (3 10)
Computing transitive relation over 3 -> 5 -> 10
Checking ambiguous relation (2 10)
Computing transitive relation over 2 -> 5 -> 10
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 5 -> 10
Checking ambiguous relation (0 10)
Computing transitive relation over 0 -> 5 -> 10
Checking ambiguous relation (6 9)
Checking ambiguous relation (5 9)
Computing transitive relation over 5 -> 10 -> 9
Checking ambiguous relation (3 8)
Checking ambiguous relation (1 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (1 6)
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 10 -> 5
Checking ambiguous relation (0 5)
Computing transitive relation over 0 -> 10 -> 5
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 4)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
Checking ambiguous relations for (5 8)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Checking ambiguous relation (5 9)
Computing transitive relation over 5 -> 8 -> 9
Checking ambiguous relation (3 8)
Computing transitive relation over 3 -> 5 -> 8
Checking ambiguous relation (1 8)
Computing transitive relation over 1 -> 5 -> 8
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (1 6)
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 8 -> 5
Checking ambiguous relation (0 5)
Computing transitive relation over 0 -> 8 -> 5
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 4)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
Checking ambiguous relations for (5 7)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Checking ambiguous relation (5 9)
Computing transitive relation over 5 -> 7 -> 9
Checking ambiguous relation (3 8)
Checking ambiguous relation (1 8)
Checking ambiguous relation (3 7)
Computing transitive relation over 3 -> 5 -> 7
Checking ambiguous relation (1 7)
Computing transitive relation over 1 -> 5 -> 7
Checking ambiguous relation (3 6)
Checking ambiguous relation (1 6)
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 7 -> 5
Checking ambiguous relation (0 5)
Computing transitive relation over 0 -> 7 -> 5
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 4)
Checking ambiguous relation (1 3)
Checking ambiguous relation (1 2)
The evaluation took 164.67 seconds.
(1 1 2 0)
> 27 ambiguous relations:

((9 11) (4 11) (3 11) (2 11) (1 11) (0 11) (9 10) (4 10) (3 10) (2 10) (1 10) (0 10) (6 9) (5 9) (3 8) (1 8) (3 7) (1 7) (3 6) (1 6) (0 6) (1 5) (0 5) (3 4) (1 4) (1 3) (1 2))

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

NIL

> Relation asserted (1 1 0 1) => (1, 4)
Checking ambiguous relations for (1 4)
Checking ambiguous relation (1 2)
Computing transitive relation over 1 -> 4 -> 2
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 4 -> 3
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 1 -> 4
Checking ambiguous relation (0 5)
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 4 -> 5
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 4 -> 6
Checking ambiguous relation (3 6)
Checking ambiguous relation (1 7)
Computing transitive relation over 1 -> 4 -> 7
Checking ambiguous relation (3 7)
Checking ambiguous relation (1 8)
Computing transitive relation over 1 -> 4 -> 8
Checking ambiguous relation (3 8)
Checking ambiguous relation (5 9)
Checking ambiguous relation (6 9)
Checking ambiguous relation (0 10)
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 4 -> 10
Checking ambiguous relation (2 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (4 10)
Computing transitive relation over 4 -> 1 -> 10
Checking ambiguous relation (9 10)
Checking ambiguous relation (0 11)
Checking ambiguous relation (1 11)
Computing transitive relation over 1 -> 4 -> 11
Checking ambiguous relation (2 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (4 11)
Computing transitive relation over 4 -> 1 -> 11
Checking ambiguous relation (9 11)
Checking ambiguous relations for (1 8)
Checking ambiguous relation (1 2)
Computing transitive relation over 1 -> 8 -> 2
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 8 -> 3
Checking ambiguous relation (3 4)
Checking ambiguous relation (0 5)
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 8 -> 5
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 8 -> 6
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 8)
Computing transitive relation over 3 -> 1 -> 8
Checking ambiguous relation (5 9)
Checking ambiguous relation (6 9)
Checking ambiguous relation (0 10)
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 8 -> 10
Checking ambiguous relation (2 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (9 10)
Checking ambiguous relation (0 11)
Checking ambiguous relation (1 11)
Computing transitive relation over 1 -> 8 -> 11
Checking ambiguous relation (2 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (9 11)
Checking ambiguous relations for (1 7)
Checking ambiguous relation (1 2)
Computing transitive relation over 1 -> 7 -> 2
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 7 -> 3
Checking ambiguous relation (3 4)
Checking ambiguous relation (0 5)
Checking ambiguous relation (1 5)
Computing transitive relation over 1 -> 7 -> 5
Checking ambiguous relation (0 6)
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 7 -> 6
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 7)
Computing transitive relation over 3 -> 1 -> 7
Checking ambiguous relation (3 8)
Checking ambiguous relation (5 9)
Checking ambiguous relation (6 9)
Checking ambiguous relation (0 10)
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 7 -> 10
Checking ambiguous relation (2 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (9 10)
Checking ambiguous relation (0 11)
Checking ambiguous relation (1 11)
Computing transitive relation over 1 -> 7 -> 11
Checking ambiguous relation (2 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (9 11)
The evaluation took 73.65 seconds.

(1 1 0 1)
> 24 ambiguous relations:
((1 2) (1 3) (3 4) (0 5) (1 5) (0 6) (1 6) (3 6) (3 7) (3 8) (5 9) (6 9) (0 10) (1 10) (2 10) (3 10) (4 10) (9 10) (0 11) (1 11) (2 11) (3 11) (4 11) (9 11))

NIL
> Relation asserted (0 1 2 1) => (1, 5)
Checking ambiguous relations for (1 5)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Computing transitive relation over 1 -> 5 -> 11
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 5 -> 10
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Checking ambiguous relation (5 9)
Computing transitive relation over 5 -> 1 -> 9
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (1 6)
Computing transitive relation over 1 -> 5 -> 6
Checking ambiguous relation (0 6)
Checking ambiguous relation (0 5)
Computing transitive relation over 0 -> 1 -> 5
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 5 -> 3
Checking ambiguous relation (1 2)
Computing transitive relation over 1 -> 5 -> 2
Checking ambiguous relations for (1 2)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Computing transitive relation over 2 \rightarrow 1 \rightarrow 11
Checking ambiguous relation (1 11)
Computing transitive relation over 1 \rightarrow 2 \rightarrow 11
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Computing transitive relation over 2 \rightarrow 1 \rightarrow 10
Checking ambiguous relation (1 10)
Computing transitive relation over 1 \rightarrow 2 \rightarrow 10
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (0 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 \rightarrow 2 \rightarrow 3
Checking ambiguous relations for (0 5)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Computing transitive relation over 0 \rightarrow 5 \rightarrow 11
Checking ambiguous relation (9 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Computing transitive relation over 0 \rightarrow 5 \rightarrow 10
Checking ambiguous relation (6 9)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (0 6)
Computing transitive relation over 0 \rightarrow 5 \rightarrow 6
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Checking ambiguous relations for (1 6)
Checking ambiguous relation (9 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Computing transitive relation over 1 \rightarrow 6 \rightarrow 11
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 6 -> 10
Checking ambiguous relation (0 10)
Computing transitive relation over 6 -> 1 -> 9
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 1 -> 6
Checking ambiguous relation (0 6)
Computing transitive relation over 0 -> 1 -> 6
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 6 -> 3
Checking ambiguous relations for (5 9)
Checking ambiguous relation (9 11)
Computing transitive relation over 9 -> 5 -> 11
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Computing transitive relation over 9 -> 5 -> 10
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (6 9)
Computing transitive relation over 6 -> 5 -> 9
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (0 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
The evaluation took 92.16 seconds.

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>(----)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(1121)</td>
<td>(3121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2120)</td>
<td>(1131)</td>
</tr>
<tr>
<td>01</td>
<td>(2122)</td>
<td>(----)</td>
<td>(1122)</td>
<td>(1123)</td>
<td>(1101)</td>
<td>(0121)</td>
<td>(2121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2122)</td>
<td>(1131)</td>
</tr>
<tr>
<td>02</td>
<td>(2122)</td>
<td>(2121)</td>
<td>(----)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2121)</td>
<td>(2121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2122)</td>
<td>(3131)</td>
</tr>
<tr>
<td>03</td>
<td>(2222)</td>
<td>(2123)</td>
<td>(2022)</td>
<td>(----)</td>
<td>(1133)</td>
<td>(2121)</td>
<td>(2123)</td>
<td>(1133)</td>
<td>(1133)</td>
<td>(2222)</td>
<td>(3133)</td>
</tr>
<tr>
<td>04</td>
<td>(2222)</td>
<td>(2022)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(----)</td>
<td>(2121)</td>
<td>(2120)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2222)</td>
<td>(3121)</td>
</tr>
<tr>
<td>05</td>
<td>(2122)</td>
<td>(0122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(----)</td>
<td>(2122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(2122)</td>
<td>(1122)</td>
</tr>
<tr>
<td>06</td>
<td>(3122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(1123)</td>
<td>(1120)</td>
<td>(1121)</td>
<td>(----)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(3122)</td>
<td>(1121)</td>
</tr>
<tr>
<td>07</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2022)</td>
<td>(2121)</td>
<td>(2022)</td>
<td>(----)</td>
<td>(1120)</td>
<td>(2222)</td>
<td>(2121)</td>
</tr>
<tr>
<td>08</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2323)</td>
<td>(2222)</td>
<td>(2121)</td>
<td>(2222)</td>
<td>(2120)</td>
<td>(----)</td>
<td>(2222)</td>
<td>(2121)</td>
</tr>
<tr>
<td>09</td>
<td>(1120)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(1121)</td>
<td>(3121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(----)</td>
<td>(1311)</td>
</tr>
<tr>
<td>10</td>
<td>(2322)</td>
<td>(2322)</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3122)</td>
<td>(2121)</td>
<td>(2122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(2322)</td>
<td>(----)</td>
</tr>
<tr>
<td>11</td>
<td>(2322)</td>
<td>(2322)</td>
<td>(3322)</td>
<td>(3323)</td>
<td>(3122)</td>
<td>(2121)</td>
<td>(2122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(2322)</td>
<td>(----)</td>
</tr>
</tbody>
</table>

NIL
> Relation asserted (0 1 2 1) => (0, 6)
Checking ambiguous relations for (0 6)
Checking ambiguous relation (1 3)
Checking ambiguous relation (3 4)
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 0 -> 6
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 8)
Checking ambiguous relation (6 9)
Computing transitive relation over 6 -> 0 -> 9
Checking ambiguous relation (0 10)
Computing transitive relation over 0 -> 6 -> 10
Checking ambiguous relation (1 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (4 10)
Checking ambiguous relation (9 10)
Checking ambiguous relation (0 11)
Computing transitive relation over 0 -> 6 -> 11
Checking ambiguous relation (1 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (4 11)
Checking ambiguous relation (9 11)
Checking ambiguous relations for (6 9)
Checking ambiguous relation (9 11)
Computing transitive relation over 9 -> 6 -> 11
Checking ambiguous relation (4 11)
Checking ambiguous relation (3 11)
Checking ambiguous relation (2 11)
Checking ambiguous relation (1 11)
Checking ambiguous relation (0 11)
Checking ambiguous relation (9 10)
Computing transitive relation over 9 -> 6 -> 10
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (9 10)
Computing transitive relation over 9 -> 6 -> 10
Checking ambiguous relation (4 10)
Checking ambiguous relation (3 10)
Checking ambiguous relation (2 10)
Checking ambiguous relation (1 10)
Checking ambiguous relation (0 10)
Checking ambiguous relation (9 10)
Computing transitive relation over 3 -> 9 -> 6
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
The evaluation took 31.59 seconds.
(0 1 2 1)

> 17 ambiguous relations:
((9 11) (4 11) (3 11) (2 11) (1 11) (0 11) (9 10) (4 10) (3 10) (2 10) (1 10) (0 10) (3 8) (3 7) (3 6) (3 4) (1 3))

00 (----) (1121) (1121) (1111) (1111) (1111) (1111) (1111) (2120) (2120) (1131) (1131)
01 (2122) (----) (1122) (1123) (1101) (0121) (2121) (1111) (1111) (1122) (1131) (1131)
02 (2122) (2121) (----) (1101) (1111) (2121) (1121) (1111) (1111) (1122) (3131) (3131)
03 (2222) (2123) (2222) (----) (1133) (2121) (2123) (1133) (1133) (2222) (3133) (3131)
04 (2222) (2222) (2222) (2323) (----) (2121) (2120) (1101) (1111) (2222) (3121) (3121)
05 (2122) (0122) (1122) (1122) (1122) (----) (2122) (1122) (1122) (2122) (1122) (1120)
06 (0122) (1122) (1122) (1123) (1120) (1121) (----) (1101) (1111) (2122) (1121) (1121)
07 (2222) (2222) (2222) (2323) (2222) (2121) (2022) (----) (1120) (2222) (2121) (2121)
08 (2222) (2222) (2222) (2323) (2222) (2121) (2222) (2120) (----) (2222) (2121) (2121)
09 (1120) (1121) (1121) (1111) (1111) (1121) (1121) (1111) (1111) (1111) (----) (1131) (1131)
NIL

> Relation asserted (0 1 2 1) => (3, 10)
Checking ambiguous relations for (3 10)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 10 -> 3
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 10 -> 4
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 10 -> 6
Checking ambiguous relation (3 7)
Computing transitive relation over 3 -> 10 -> 7
Checking ambiguous relation (3 8)
Computing transitive relation over 3 -> 10 -> 8
Checking ambiguous relation (0 10)
Computing transitive relation over 0 -> 3 -> 4
Checking ambiguous relation (1 10)
Computing transitive relation over 1 -> 3 -> 10
Checking ambiguous relation (2 10)
Checking ambiguous relation over 2 -> 3 -> 10
Checking ambiguous relation (4 10)
Computing transitive relation over 4 -> 3 -> 10
Checking ambiguous relation (9 10)
Computing transitive relation over 9 -> 3 -> 10
Checking ambiguous relation (0 11)
Checking ambiguous relation (1 10)
Checking ambiguous relation (2 11)
Checking ambiguous relation (3 11)
Computing transitive relation over 3 -> 10 -> 11
Checking ambiguous relation (4 11)
Checking ambiguous relation (9 11)
Checking ambiguous relations for (3 11)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 11 -> 3
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 11 -> 4
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 11 -> 6
Checking ambiguous relation (2 7)
Computing transitive relation over 2 -> 3 -> 7
Checking ambiguous relation (3 8)
Computing transitive relation over 3 -> 11 -> 8
Checking ambiguous relation (0 11)
Computing transitive relation over 0 -> 3 -> 11
Checking ambiguous relation (1 11)
Computing transitive relation over 1 -> 3 -> 11
Checking ambiguous relation (2 11)
Computing transitive relation over 2 -> 3 -> 11
Checking ambiguous relation (4 11)
Computing transitive relation over 4 -> 3 -> 11
Checking ambiguous relation (9 11)
Computing transitive relation over 9 -> 3 -> 11
Checking ambiguous relations for (9 11)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Checking ambiguous relations for (4 11)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 11 -> 4
Checking ambiguous relation (1 3)
Checking ambiguous relations for (2 11)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Checking ambiguous relations for (1 11)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 11 -> 3
Checking ambiguous relations for (0 11)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Checking ambiguous relations for (9 10)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Checking ambiguous relations for (4 10)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 10 -> 4
Checking ambiguous relation (1 3)
Checking ambiguous relations for (2 10)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Checking ambiguous relations for (1 10)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 10 -> 3
Checking ambiguous relations for (0 10)
Checking ambiguous relation (3 8)
Checking ambiguous relation (3 7)
Checking ambiguous relation (3 6)
Checking ambiguous relation (3 4)
Checking ambiguous relation (1 3)
The evaluation took 81.51 seconds.
5 ambiguous relations:

((3 8) (3 7) (3 6) (3 4) (1 3))

NIL

> Relation asserted (1 1 1 1) => (3, 7)
Checking ambiguous relations for (3 7)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 7 -> 3
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 7 -> 4
Checking ambiguous relation (3 6)
Computing transitive relation over 3 -> 7 -> 6
Checking ambiguous relation (3 8)
Computing transitive relation over 3 -> 7 -> 8
Checking ambiguous relations for (3 8)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 8 -> 3
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 8 -> 4
Checking ambiguous relations for (3 6)
Checking ambiguous relation (1 3)
Computing transitive relation over 1 -> 6 -> 3
Checking ambiguous relation (3 4)
Computing transitive relation over 3 -> 6 -> 4
The evaluation took 13.79 seconds.

(1 1 1 1)

> 2 ambiguous relations:

((1 3) (3 4))

NIL

> error: Invalid relation provided.
> 2 ambiguous relations:

```plaintext
( (1 3) (3 4) )
```

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>(---)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(1121)</td>
<td>(0121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2120)</td>
</tr>
<tr>
<td>01</td>
<td>(2122)</td>
<td>(---)</td>
<td>(1122)</td>
<td>(1123)</td>
<td>(1101)</td>
<td>(0121)</td>
<td>(2121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2122)</td>
</tr>
<tr>
<td>02</td>
<td>(2122)</td>
<td>(2121)</td>
<td>(---)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2121)</td>
<td>(2121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2122)</td>
</tr>
<tr>
<td>03</td>
<td>(2222)</td>
<td>(2123)</td>
<td>(2022)</td>
<td>(---)</td>
<td>(1131)</td>
<td>(2121)</td>
<td>(2121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(2222)</td>
</tr>
<tr>
<td>04</td>
<td>(2222)</td>
<td>(2022)</td>
<td>(2222)</td>
<td>(2322)</td>
<td>(---)</td>
<td>(2121)</td>
<td>(2120)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2222)</td>
</tr>
<tr>
<td>05</td>
<td>(2122)</td>
<td>(0122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(---)</td>
<td>(2122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(2122)</td>
</tr>
<tr>
<td>06</td>
<td>(0122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(1120)</td>
<td>(1121)</td>
<td>(---)</td>
<td>(1101)</td>
<td>(1111)</td>
<td>(2122)</td>
</tr>
<tr>
<td>07</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2022)</td>
<td>(2121)</td>
<td>(2022)</td>
<td>(---)</td>
<td>(1120)</td>
<td>(2222)</td>
</tr>
<tr>
<td>08</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2222)</td>
<td>(2121)</td>
<td>(2222)</td>
<td>(2122)</td>
<td>(2122)</td>
<td>(2122)</td>
<td>(---)</td>
</tr>
<tr>
<td>09</td>
<td>(1120)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(1121)</td>
<td>(1121)</td>
<td>(1111)</td>
<td>(1111)</td>
<td>(---)</td>
<td>(1111)</td>
</tr>
<tr>
<td>10</td>
<td>(2222)</td>
<td>(2122)</td>
<td>(2022)</td>
<td>(0122)</td>
<td>(1122)</td>
<td>(2121)</td>
<td>(2122)</td>
<td>(1122)</td>
<td>(1122)</td>
<td>(2222)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> error: Invalid relation provided.

>
APPENDIX D

LARGER TEST CASES
This appendix presents data generated for five test case runs. The purpose of these tests were to verify that the performance complexity of the algorithm was not worse than \( n^2 \) and to provide empirical data indicating the typical performance of the algorithm under different conditions.

Five tests were generated and executed. Each of the five tests had 20, 40, 60, 80, and 100 intervals resulting in a relation matrix of 400, 1,600, 3,600, 6,400, and 10,000 relations, respectively. A relation assignment was selected randomly for the last interval in the system and the new interval as it was added to the system.

The test routines then started with the set of ambiguous relations and selected a relation assignment that was compatible with the ambiguous relation and asserted the unambiguous relation into the system, initiating the transitive closure algorithm. This was performed for \( n \) ambiguous intervals, where \( n \) = the number of intervals in the system. The test routines printed out the number of ambiguous relations prior to the assertion of the unambiguous relation, the relation asserted, the time taken to perform the transitive closure computation, and the number of ambiguous relations at the end of the transitive closure computation.

**Twenty Interval Test Case**

```lisp
> (load "test20")
(load "test20")
; loading "test20.lsp"
System initialized.
Relation asserted (2 1 2 2) => (0, 1)
Relation asserted (2 0 2 2) => (1, 2)
Relation asserted (1 1 0 1) => (2, 3)
Relation asserted (0 1 2 1) => (3, 4)
Relation asserted (2 1 2 2) => (4, 5)
```
Relation asserted (2 1 2 0) => (5, 6)
Relation asserted (1 1 2 2) => (6, 7)
Relation asserted (0 1 2 1) => (7, 8)
Relation asserted (2 0 2 2) => (8, 9)
Relation asserted (0 1 2 0) => (9, 10)
Relation asserted (0 1 2 2) => (10, 11)
Relation asserted (0 1 2 0) => (11, 12)
Relation asserted (2 0 2 2) => (12, 13)
Relation asserted (2 2 2 2) => (13, 14)
Relation asserted (2 0 2 2) => (14, 15)
Relation asserted (2 1 2 0) => (15, 16)
Relation asserted (1 1 2 0) => (16, 17)
Relation asserted (2 2 2 2) => (17, 18)
Relation asserted (2 2 2 2) => (18, 19)

There are 107 ambiguous relations.

Start: 107 Ambiguous Relations
Relation asserted (2 2 2 2) => (16, 19)
CPU 1.17 sec., Real 0.00 sec.
End: 105 ambiguous relations
Start: 105 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 19)
CPU 3.73 sec., Real 2.00 sec.
End: 99 ambiguous relations
Start: 99 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 19)
CPU 0.45 sec., Real 0.00 sec.
End: 98 ambiguous relations
Start: 98 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 3)
CPU 1.82 sec., Real 1.00 sec.
End: 96 ambiguous relations
Start: 96 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 4)
CPU 7.28 sec., Real 4.00 sec.
End: 79 ambiguous relations
Start: 79 Ambiguous Relations
Relation asserted (2 2 2 2) => (16, 18)
CPU 0.78 sec., Real 0.00 sec.
End: 77 ambiguous relations
Start: 77 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 18)
CPU 2.00 sec., Real 1.00 sec.
End: 72 ambiguous relations
Start: 72 Ambiguous Relations
Relation asserted (2 1 2 1) => (2, 5)
CPU 1.55 sec., Real 1.00 sec.
End: 69 ambiguous relations
Start: 69 Ambiguous Relations
Relation asserted (1 1 2 0) => (15, 17)
CPU 0.43 sec., Real 0.00 sec.
End: 68 ambiguous relations
Start: 68 Ambiguous Relations
Relation asserted (2 1 2 2) => (3, 5)
CPU 1.13 sec., Real 1.00 sec.
End: 66 ambiguous relations
Start: 66 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 7)
CPU 4.52 sec., Real 3.00 sec.
End: 36 ambiguous relations
Start: 36 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 7)
CPU 0.90 sec., Real 1.00 sec.
End: 26 ambiguous relations
Start: 26 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 7)
CPU 0.58 sec., Real 0.00 sec.
End: 16 ambiguous relations
Start: 16 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 8)
CPU 0.10 sec., Real 0.00 sec.
End: 15 ambiguous relations
Start: 15 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 17)
CPU 0.35 sec., Real 0.00 sec.
End: 12 ambiguous relations
Start: 12 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 8)
CPU 0.23 sec., Real 0.00 sec.
End: 8 ambiguous relations
Start: 8 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 8)
CPU 0.03 sec., Real 0.00 sec.
End: 7 ambiguous relations
Start: 7 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 14)
CPU 0.07 sec., Real 1.00 sec.
End: 6 ambiguous relations
Start: 6 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 8)
CPU 0.02 sec., Real 0.00 sec.
End: 5 ambiguous relations
Start: 5 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 13)
CPU 0.10 sec., Real 0.00 sec.
End: 2 ambiguous relations

Forty Interval Test Case

; loading "test40.lsp"
System initialized.
Relation asserted (2 1 2 0) => (0, 1)
Relation asserted (1 1 1 1) => (1, 2)
Relation asserted (1 1 2 0) => (2, 3)
Relation asserted (1 1 1 1) => (3, 4)
Relation asserted (2 1 2 2) => (4, 5)
Relation asserted (2 1 2 1) => (5, 6)
Relation asserted (2 0 2 2) => (6, 7)
Relation asserted (0 1 2 2) => (7, 8)
Relation asserted (1 1 2 2) => (8, 9)
Relation asserted (1 1 2 2) => (9, 10)
Relation asserted (0 1 2 2) => (10, 11)
Relation asserted (0 1 2 2) => (11, 12)
Relation asserted (2 1 2 1) => (12, 13)
Relation asserted (2 0 2 2) => (13, 14)
Relation asserted (0 1 2 1) => (14, 15)
Relation asserted (2 1 2 0) => (15, 16)
Relation asserted (1 1 1 1) => (16, 17)
Relation asserted (0 1 2 1) => (17, 18)
Relation asserted (0 1 2 2) => (18, 19)
Relation asserted (1 1 2 0) => (19, 20)
Relation asserted (2 1 2 1) => (20, 21)
Relation asserted (0 1 2 2) => (21, 22)
Relation asserted (2 0 2 2) => (22, 23)
Relation asserted (0 1 2 0) => (23, 24)
Relation asserted (1 1 2 2) => (24, 25)
Relation asserted (2 1 2 0) => (25, 26)
Relation asserted (1 1 2 0) => (26, 27)
Relation asserted (2 1 2 0) => (27, 28)
Relation asserted (1 1 2 2) => (28, 29)
Relation asserted (1 1 2 0) => (29, 30)
Relation asserted (1 1 2 1) => (30, 31)
Relation asserted (1 1 2 2) => (31, 32)
Relation asserted (2 1 2 2) => (32, 33)
Relation asserted (1 1 1 1) => (33, 34)
Relation asserted (0 1 2 1) => (34, 35)
Relation asserted (0 1 2 0) => (35, 36)
Relation asserted (2 1 2 1) => (36, 37)
Relation asserted (0 1 2 1) => (37, 38)
Relation asserted (0 1 2 0) => (38, 39)
There are 651 ambiguous relations.
Start: 651 Ambiguous Relations
Relation asserted (2 1 2 1) => (33, 39)
CPU 42.12 sec., Real 26.00 sec.
End: 648 ambiguous relations
Start: 648 Ambiguous Relations
Relation asserted (2 1 2 1) => (0, 5)
CPU 233.07 sec., Real 141.00 sec.
End: 621 ambiguous relations
Start: 621 Ambiguous Relations
Relation asserted (2 1 2 2) => (32, 39)
CPU 84.50 sec., Real 51.00 sec.
End: 612 ambiguous relations
Start: 612 Ambiguous Relations
Relation asserted (2 1 2 1) => (1, 5)
CPU 73.83 sec., Real 44.00 sec.
End: 603 ambiguous relations
Start: 603 Ambiguous Relations
Relation asserted (2 1 2 2) => (31, 39)
CPU 35.72 sec., Real 26.00 sec.
End: 600 ambiguous relations
Start: 600 Ambiguous Relations
Relation asserted (2 1 2 2) => (4, 6)
CPU 7.98 sec., Real 6.00 sec.
End: 599 ambiguous relations
Start: 599 Ambiguous Relations
Relation asserted (2 1 2 2) => (30, 39)
CPU 249.18 sec., Real 155.00 sec.
End: 566 ambiguous relations
Start: 566 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 13)
CPU 27.48 sec., Real 18.00 sec.
End: 562 ambiguous relations
Start: 562 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 13)
CPU 8.68 sec., Real 6.00 sec.
End: 561 ambiguous relations
Start: 561 Ambiguous Relations
Relation asserted (2 1 2 2) => (29, 39)
CPU 28.05 sec., Real 20.00 sec.
End: 558 ambiguous relations
Start: 558 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 13)
CPU 8.40 sec., Real 10.00 sec.
End: 557 ambiguous relations
Start: 557 Ambiguous Relations
Relation asserted (2 1 2 2) => (28, 39)
CPU 26.32 sec., Real 17.00 sec.
End: 554 ambiguous relations
Start: 554 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 13)
CPU 7.83 sec., Real 5.00 sec.
End: 553 ambiguous relations
Start: 553 Ambiguous Relations
Relation asserted (2 2 2 2) => (27, 39)
CPU 55.98 sec., Real 36.00 sec.
End: 546 ambiguous relations
Start: 546 Ambiguous Relations
Relation asserted (2 1 2 2) => (7, 13)
CPU 23.78 sec., Real 15.00 sec.
End: 542 ambiguous relations
Start: 542 Ambiguous Relations
 relation asserted (2 2 2 2) => (8, 13)
CPU 7.23 sec., Real 5.00 sec.
End: 541 ambiguous relations
Start: 541 Ambiguous Relations
Relation asserted (2 2 2 2) => (26, 39)
CPU 79.92 sec., Real 51.00 sec.
End: 525 ambiguous relations
Start: 525 Ambiguous Relations
Relation asserted (2 2 2 2) => (24, 39)
CPU 79.60 sec., Real 53.00 sec.
End: 509 ambiguous relations
Start: 509 Ambiguous Relations
Relation asserted (2 2 2 2) => (19, 39)
CPU 108.30 sec., Real 72.00 sec.
End: 488 ambiguous relations
Start: 488 Ambiguous Relations
Relation asserted (2 1 2 2) => (9, 13)
CPU 6.08 sec., Real 4.00 sec.
End: 487 ambiguous relations
Start: 487 Ambiguous Relations
Relation asserted (2 2 2 2) => (16, 39)
CPU 246.77 sec., Real 181.00 sec.
End: 368 ambiguous relations
Start: 368 Ambiguous Relations
Relation asserted (2 1 2 2) => (10, 13)
CPU 3.63 sec., Real 2.00 sec.
End: 367 ambiguous relations
Start: 367 Ambiguous Relations
Relation asserted (2 1 2 2) => (31, 33)
CPU 4.50 sec., Real 3.00 sec.
End: 366 ambiguous relations
Start: 366 Ambiguous Relations
Relation asserted (2 1 2 2) => (11, 13)
CPU 3.45 sec., Real 2.00 sec.
End: 365 ambiguous relations

Start: 365 Ambiguous Relations
Relation asserted (2 1 2 2) => (30, 33)
CPU 3.73 sec., Real 2.00 sec.
End: 364 ambiguous relations

Start: 364 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 15)
CPU 22.82 sec., Real 13.00 sec.
End: 356 ambiguous relations

Start: 356 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 15)
CPU 5.35 sec., Real 3.00 sec.
End: 354 ambiguous relations

Start: 354 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 15)
CPU 5.33 sec., Real 3.00 sec.
End: 352 ambiguous relations

Start: 352 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 15)
CPU 5.22 sec., Real 3.00 sec.
End: 350 ambiguous relations

Start: 350 Ambiguous Relations
Relation asserted (2 2 2 2) => (7, 15)
CPU 34.97 sec., Real 21.00 sec.
End: 336 ambiguous relations

Start: 336 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 17)
CPU 11.42 sec., Real 7.00 sec.
End: 332 ambiguous relations

Start: 332 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 17)
CPU 3.73 sec., Real 2.00 sec.
End: 331 ambiguous relations

Start: 331 Ambiguous Relations
Relation asserted (2 1 2 2) => (29, 33)
CPU 2.98 sec., Real 2.00 sec.
End: 330 ambiguous relations

Start: 330 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 17)
CPU 3.53 sec., Real 2.00 sec.
End: 329 ambiguous relations

Start: 329 Ambiguous Relations
Relation asserted (2 1 2 2) => (28, 33)
CPU 2.30 sec., Real 1.00 sec.
End: 328 ambiguous relations
Sixty Interval Test Case

; loading "test60.lsp"
System initialized.
Relation asserted (0 1 2 2) => (0, 1)
Relation asserted (2 1 2 2) => (1, 2)
Relation asserted (1 1 2 1) => (2, 3)
Relation asserted (2 1 2 2) => (3, 4)
Relation asserted (0 1 2 0) => (4, 5)
Relation asserted (2 2 2 2) => (5, 6)
Relation asserted (0 1 2 2) => (6, 7)
Relation asserted (0 1 2 2) => (7, 8)
Relation asserted (0 1 2 2) => (8, 9)
Relation asserted (2 1 2 1) => (9, 10)
Relation asserted (0 1 2 1) => (10, 11)
Relation asserted (0 1 2 0) => (11, 12)
Relation asserted (1 1 2 2) => (12, 13)
Relation asserted (2 2 2 2) => (13, 14)
Relation asserted (2 2 2 2) => (14, 15)
Relation asserted (2 0 2 2) => (15, 16)
Relation asserted (2 2 2 2) => (16, 17)
Relation asserted (1 1 0 1) => (17, 18)
Relation asserted (1 1 2 2) => (18, 19)
Relation asserted (0 1 2 2) => (19, 20)
Relation asserted (2 1 2 2) => (20, 21)
Relation asserted (2 2 2 2) => (21, 22)
Relation asserted (1 1 0 1) => (22, 23)
Relation asserted (1 1 2 2) => (23, 24)
Relation asserted (1 1 2 2) => (24, 25)
Relation asserted (1 1 2 1) => (25, 26)
Relation asserted (1 1 0 1) => (26, 27)
Relation asserted (0 1 2 1) => (27, 28)
Relation asserted (1 1 2 2) => (28, 29)
Relation asserted (2 1 2 0) => (29, 30)
Relation asserted (0 1 2 1) => (30, 31)
Relation asserted (2 1 2 1) => (31, 32)
Relation asserted (2 1 2 2) => (32, 33)
Relation asserted (0 1 2 0) => (33, 34)
Relation asserted (0 1 2 1) => (34, 35)
Relation asserted (2 0 2 2) => (35, 36)
Relation asserted (1 1 0 1) => (36, 37)
Relation asserted (0 1 2 2) => (37, 38)
Relation asserted (2 2 2 2) => (38, 39)
Relation asserted (0 1 2 2) => (39, 40)
Relation asserted (2 1 2 1) => (40, 41)
Relation asserted (1 1 0 1) => (41, 42)
Relation asserted (2 1 2 2) => (42, 43)
Relation asserted (1 1 0 1) => (43, 44)
Relation asserted (0 1 2 2) => (44, 45)
Relation asserted (2 1 2 1) => (45, 46)
Relation asserted (2 1 2 2) => (46, 47)
Relation asserted (1 1 2 0) => (47, 48)
Relation asserted (1 1 2 1) => (48, 49)
Relation asserted (1 1 2 0) => (49, 50)
Relation asserted (1 1 2 2) => (50, 51)
Relation asserted (2 2 2 2) => (51, 52)
Relation asserted (1 1 1 1) => (52, 53)
Relation asserted (1 1 0 1) => (53, 54)
Relation asserted (2 1 2 1) => (54, 55)
Relation asserted (1 1 2 0) => (55, 56)
Relation asserted (2 0 2 2) => (56, 57)
Relation asserted (1 1 2 2) => (57, 58)
Relation asserted (1 1 2 2) => (58, 59)

There are 1629 ambiguous relations.

Start: 1629 Ambiguous Relations
Relation asserted (2 2 2 2) => (55, 59)
CPU 156.92 sec., Real 94.00 sec.
End: 1626 ambiguous relations
Start: 1626 Ambiguous Relations
Relation asserted (2 1 2 2) => (0, 3)
CPU 385.57 sec., Real 232.00 sec.
End: 1617 ambiguous relations
Start: 1617 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (53, 59)\)
CPU 60.88 sec., Real 37.00 sec.
End: 1616 ambiguous relations
Start: 1616 Ambiguous Relations
Relation asserted \((2 \ 1 \ 2 \ 2) \Rightarrow (1, 3)\)
CPU 42.12 sec., Real 26.00 sec.
End: 1615 ambiguous relations
Start: 1615 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (52, 59)\)
CPU 71.03 sec., Real 42.00 sec.
End: 1613 ambiguous relations
Start: 1613 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (50, 59)\)
CPU 56.37 sec., Real 34.00 sec.
End: 1612 ambiguous relations
Start: 1612 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (0, 4)\)
CPU 132.58 sec., Real 80.00 sec.
End: 1608 ambiguous relations
Start: 1608 Ambiguous Relations
Relation asserted \((2 \ 1 \ 2 \ 2) \Rightarrow (2, 4)\)
CPU 166.17 sec., Real 99.00 sec.
End: 1602 ambiguous relations
Start: 1602 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (0, 10)\)
CPU 67.48 sec., Real 41.00 sec.
End: 1600 ambiguous relations
Start: 1600 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (2, 10)\)
CPU 66.12 sec., Real 40.00 sec.
End: 1598 ambiguous relations
Start: 1598 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (4, 10)\)
CPU 64.87 sec., Real 39.00 sec.
End: 1596 ambiguous relations
Start: 1596 Ambiguous Relations
Relation asserted \((2 \ 1 \ 2 \ 2) \Rightarrow (6, 10)\)
CPU 39.32 sec., Real 23.00 sec.
End: 1595 ambiguous relations
Start: 1595 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (49, 59)\)
CPU 53.38 sec., Real 32.00 sec.
End: 1594 ambiguous relations
Start: 1594 Ambiguous Relations
Relation asserted \((2 \ 1 \ 2 \ 2) \Rightarrow (7, 10)\)
CPU 38.65 sec., Real 23.00 sec.
End: 1593 ambiguous relations
Start: 1593 Ambiguous Relations
Relation asserted (2 2 2 2) => (48, 59)
CPU 52.70 sec., Real 32.00 sec.
End: 1592 ambiguous relations
Start: 1592 Ambiguous Relations
Relation asserted (2 1 2 2) => (8, 10)
CPU 37.97 sec., Real 23.00 sec.
End: 1591 ambiguous relations
Start: 1591 Ambiguous Relations
Relation asserted (2 2 2 2) => (47, 59)
CPU 238.10 sec., Real 143.00 sec.
End: 1586 ambiguous relations
Start: 1586 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 11)
CPU 352.75 sec., Real 213.00 sec.
End: 1572 ambiguous relations
Start: 1572 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 11)
CPU 345.18 sec., Real 208.00 sec.
End: 1558 ambiguous relations
Start: 1558 Ambiguous Relations
Relation asserted (2 2 2 2) => (4, 11)
CPU 337.62 sec., Real 203.00 sec.
End: 1544 ambiguous relations
Start: 1544 Ambiguous Relations
Relation asserted (2 1 2 2) => (6, 11)
CPU 59.13 sec., Real 36.00 sec.
End: 1542 ambiguous relations
Start: 1542 Ambiguous Relations
Relation asserted (2 1 2 2) => (7, 11)
CPU 59.12 sec., Real 35.00 sec.
End: 1540 ambiguous relations
Start: 1540 Ambiguous Relations
Relation asserted (2 1 2 2) => (8, 11)
CPU 59.02 sec., Real 36.00 sec.
End: 1538 ambiguous relations
Start: 1538 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 13)
CPU 680.35 sec., Real 410.00 sec.
End: 1517 ambiguous relations
Start: 1517 Ambiguous Relations
Relation asserted (2 2 2 2) => (43, 59)
CPU 42.65 sec., Real 26.00 sec.
End: 1516 ambiguous relations
Start: 1516 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 14)
CPU 318.12 sec., Real 192.00 sec.
End: 1504 ambiguous relations
Start: 1504 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 18)
CPU 229.75 sec., Real 138.00 sec.
End: 1494 ambiguous relations
Start: 1494 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 18)
CPU 225.53 sec., Real 135.00 sec.
End: 1484 ambiguous relations
Start: 1484 Ambiguous Relations
Relation asserted (2 2 2 2) => (4, 18)
CPU 221.25 sec., Real 133.00 sec.
End: 1474 ambiguous relations
Start: 1474 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 18)
CPU 418.38 sec., Real 252.00 sec.
End: 1454 ambiguous relations
Start: 1454 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 18)
CPU 404.33 sec., Real 244.00 sec.
End: 1434 ambiguous relations
Start: 1434 Ambiguous Relations
Relation asserted (2 2 2 2) => (14, 18)
CPU 131.45 sec., Real 79.00 sec.
End: 1429 ambiguous relations
Start: 1429 Ambiguous Relations
Relation asserted (2 2 2 2) => (41, 59)
CPU 247.27 sec., Real 149.00 sec.
End: 1417 ambiguous relations
Start: 1417 Ambiguous Relations
Relation asserted (2 2 2 2) => (36, 59)
CPU 28.17 sec., Real 17.00 sec.
End: 1416 ambiguous relations
Start: 1416 Ambiguous Relations
Relation asserted (2 2 2 2) => (15, 18)
CPU 126.68 sec., Real 77.00 sec.
End: 1411 ambiguous relations
Start: 1411 Ambiguous Relations
Relation asserted (2 2 2 2) => (28, 59)
CPU 48.03 sec., Real 29.00 sec.
End: 1409 ambiguous relations
Start: 1409 Ambiguous Relations
Relation asserted (2 2 2 2) => (26, 59)
CPU 24.17 sec., Real 15.00 sec.
End: 1408 ambiguous relations
Start: 1408 Ambiguous Relations
Relation asserted (2 2 2 2) => (16, 18)
CPU 123.57 sec., Real 75.00 sec.
End: 1403 ambiguous relations
Start: 1403 Ambiguous Relations
Relation asserted (2 2 2 2) => (25, 59)
CPU 23.20 sec., Real 14.00 sec.
End: 1402 ambiguous relations
Start: 1402 Ambiguous Relations
Relation asserted (2 1 2 1) => (17, 21)
CPU 87.42 sec., Real 53.00 sec.
End: 1398 ambiguous relations
Start: 1398 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 23)
CPU 152.63 sec., Real 91.00 sec.
End: 1390 ambiguous relations
Start: 1390 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 23)
CPU 148.93 sec., Real 90.00 sec.
End: 1382 ambiguous relations
Start: 1382 Ambiguous Relations
Relation asserted (2 2 2 2) => (4, 23)
CPU 145.47 sec., Real 88.00 sec.
End: 1374 ambiguous relations
Start: 1374 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 23)
CPU 280.43 sec., Real 168.00 sec.
End: 1358 ambiguous relations
Start: 1358 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 23)
CPU 276.13 sec., Real 166.00 sec.
End: 1342 ambiguous relations
Start: 1342 Ambiguous Relations
Relation asserted (2 2 2 2) => (14, 23)
CPU 64.90 sec., Real 39.00 sec.
End: 1338 ambiguous relations
Start: 1338 Ambiguous Relations
Relation asserted (2 2 2 2) => (15, 23)
CPU 64.07 sec., Real 39.00 sec.
End: 1334 ambiguous relations
Start: 1334 Ambiguous Relations
Relation asserted (2 2 2 2) => (16, 23)
CPU 63.67 sec., Real 38.00 sec.
End: 1330 ambiguous relations
Start: 1330 Ambiguous Relations
Relation asserted (2 2 2 2) => (17, 23)
CPU 251.57 sec., Real 152.00 sec.
End: 1314 ambiguous relations
Start: 1314 Ambiguous Relations
Relation asserted (2 2 2 2) => (21, 23)
CPU 59.73 sec., Real 36.00 sec.
End: 1310 ambiguous relations
Start: 1310 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 26)
CPU 45.37 sec., Real 27.00 sec.
End: 1308 ambiguous relations
Start: 1308 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 26)
CPU 44.28 sec., Real 27.00 sec.
End: 1306 ambiguous relations
Start: 1306 Ambiguous Relations
Relation asserted (2 2 2 2) => (4, 26)
CPU 43.20 sec., Real 26.00 sec.
End: 1304 ambiguous relations
Start: 1304 Ambiguous Relations
Relation asserted (2 2 2 2) => (6, 26)
CPU 73.42 sec., Real 45.00 sec.
End: 1300 ambiguous relations
Start: 1300 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 26)
CPU 70.65 sec., Real 43.00 sec.
End: 1296 ambiguous relations
Start: 1296 Ambiguous Relations
Relation asserted (2 2 2 2) => (14, 26)
CPU 22.47 sec., Real 13.00 sec.
End: 1295 ambiguous relations
Start: 1295 Ambiguous Relations
Relation asserted (2 2 2 2) => (24, 59)
CPU 9.30 sec., Real 5.00 sec.
End: 1294 ambiguous relations
Start: 1294 Ambiguous Relations
Relation asserted (2 2 2 2) => (15, 26)
CPU 21.95 sec., Real 14.00 sec.
End: 1293 ambiguous relations
Start: 1293 Ambiguous Relations
Relation asserted (2 2 2 2) => (23, 59)
CPU 9.08 sec., Real 6.00 sec.
End: 1292 ambiguous relations
Start: 1292 Ambiguous Relations
Relation asserted (2 2 2 2) => (16, 26)
CPU 21.42 sec., Real 13.00 sec.
End: 1291 ambiguous relations
Eighty Interval Test Case

; loading "test80.lsp"
System initialized.
Relation asserted (0 1 2 1) => (0, 1)
Relation asserted (2 1 2 2) => (1, 2)
Relation asserted (1 1 0 1) => (2, 3)
Relation asserted (0 1 2 1) => (3, 4)
Relation asserted (2 1 2 2) => (4, 5)
Relation asserted (0 1 2 1) => (5, 6)
Relation asserted (0 1 2 0) => (6, 7)
Relation asserted (2 1 2 0) => (7, 8)
Relation asserted (0 1 2 0) => (8, 9)
Relation asserted (2 1 2 0) => (9, 10)
Relation asserted (0 1 2 1) => (10, 11)
Relation asserted (1 1 2 1) => (11, 12)
Relation asserted (2 2 2 2) => (12, 13)
Relation asserted (1 1 2 1) => (13, 14)
Relation asserted (1 1 2 2) => (14, 15)
Relation asserted (2 1 2 0) => (15, 16)
Relation asserted (0 1 2 0) => (16, 17)
Relation asserted (2 1 2 0) => (17, 18)
Relation asserted (2 1 2 1) => (18, 19)
Relation asserted (2 1 2 0) => (19, 20)
Relation asserted (1 1 2 2) => (20, 21)
Relation asserted (2 1 2 2) => (21, 22)
Relation asserted (2 2 2 2) => (22, 23)
Relation asserted (1 1 2 1) => (23, 24)
Relation asserted (1 1 0 1) => (24, 25)
Relation asserted (0 1 2 2) => (25, 26)
Relation asserted (1 1 0 1) => (26, 27)
Relation asserted (2 1 2 0) => (27, 28)
Relation asserted (2 2 2 2) => (28, 29)
Relation asserted (1 1 1 1) => (29, 30)
Relation asserted (0 1 2 2) => (30, 31)
Relation asserted (1 1 2 2) => (31, 32)
Relation asserted (1 1 2 1) => (32, 33)
Relation asserted (1 1 2 0) => (33, 34)
Relation asserted (1 1 2 0) => (34, 35)
Relation asserted (0 1 2 1) => (35, 36)
Relation asserted (1 1 2 2) => (36, 37)
Relation asserted (2 1 2 2) => (37, 38)
Relation asserted (0 1 2 1) => (38, 39)
Relation asserted (1 1 1 1) => (39, 40)
Relation asserted (1 1 1 1) => (40, 41)
Relation asserted (2 0 2 2) => (41, 42)
Relation asserted (1 1 0 1) => (42, 43)
Relation asserted \((1 \ 1 \ 1 \ 1) \Rightarrow (43, 44)\)
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (44, 45)\)
Relation asserted \((2 \ 0 \ 2 \ 2) \Rightarrow (45, 46)\)
Relation asserted \((0 \ 1 \ 2 \ 2) \Rightarrow (46, 47)\)
Relation asserted \((2 \ 0 \ 2 \ 2) \Rightarrow (47, 48)\)
Relation asserted \((2 \ 1 \ 2 \ 1) \Rightarrow (48, 49)\)
Relation asserted \((2 \ 1 \ 2 \ 1) \Rightarrow (49, 50)\)
Relation asserted \((1 \ 1 \ 0 \ 1) \Rightarrow (50, 51)\)
Relation asserted \((2 \ 1 \ 2 \ 1) \Rightarrow (51, 52)\)
Relation asserted \((1 \ 1 \ 0 \ 1) \Rightarrow (52, 53)\)
Relation asserted \((1 \ 1 \ 2 \ 1) \Rightarrow (53, 54)\)
Relation asserted \((1 \ 1 \ 0 \ 1) \Rightarrow (54, 55)\)
Relation asserted \((1 \ 1 \ 2 \ 1) \Rightarrow (55, 56)\)
Relation asserted \((0 \ 1 \ 2 \ 1) \Rightarrow (56, 57)\)
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (57, 58)\)
Relation asserted \((1 \ 1 \ 2 \ 1) \Rightarrow (58, 59)\)
Relation asserted \((1 \ 1 \ 2 \ 2) \Rightarrow (59, 60)\)
Relation asserted \((1 \ 1 \ 2 \ 2) \Rightarrow (60, 61)\)
Relation asserted \((1 \ 1 \ 1 \ 1) \Rightarrow (61, 62)\)
Relation asserted \((2 \ 1 \ 2 \ 2) \Rightarrow (62, 63)\)
Relation asserted \((2 \ 1 \ 2 \ 0) \Rightarrow (63, 64)\)
Relation asserted \((2 \ 1 \ 2 \ 1) \Rightarrow (64, 65)\)
Relation asserted \((0 \ 1 \ 2 \ 1) \Rightarrow (65, 66)\)
Relation asserted \((1 \ 1 \ 2 \ 2) \Rightarrow (66, 67)\)
Relation asserted \((1 \ 1 \ 2 \ 0) \Rightarrow (67, 68)\)
Relation asserted \((1 \ 1 \ 0 \ 1) \Rightarrow (68, 69)\)
Relation asserted \((1 \ 1 \ 2 \ 2) \Rightarrow (69, 70)\)
Relation asserted \((1 \ 1 \ 1 \ 1) \Rightarrow (70, 71)\)
Relation asserted \((0 \ 1 \ 2 \ 0) \Rightarrow (71, 72)\)
Relation asserted \((1 \ 1 \ 2 \ 0) \Rightarrow (72, 73)\)
Relation asserted \((1 \ 1 \ 2 \ 2) \Rightarrow (73, 74)\)
Relation asserted \((1 \ 1 \ 2 \ 1) \Rightarrow (74, 75)\)
Relation asserted \((2 \ 0 \ 2 \ 2) \Rightarrow (75, 76)\)
Relation asserted \((1 \ 1 \ 2 \ 0) \Rightarrow (76, 77)\)
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (77, 78)\)
Relation asserted \((2 \ 0 \ 2 \ 2) \Rightarrow (78, 79)\)

There are 2953 ambiguous relations.

Start: 2953 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (76, 79)\)
CPU 162.72 sec., Real 98.00 sec.

End: 2952 ambiguous relations
Start: 2952 Ambiguous Relations
Relation asserted \((2 \ 1 \ 2 \ 2) \Rightarrow (0, 2)\)
CPU 116.13 sec., Real 69.00 sec.

End: 2951 ambiguous relations
Start: 2951 Ambiguous Relations
Relation asserted \((2 \ 2 \ 2 \ 2) \Rightarrow (74, 79)\)
CPU 153.35 sec., Real 92.00 sec.
<table>
<thead>
<tr>
<th>Start Time</th>
<th>Ambiguous Relations</th>
<th>Relation asserted</th>
<th>Time CPU</th>
<th>Time Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>2950</td>
<td>2950 Ambiguous Relations</td>
<td>(1 1 2 0) =&gt; (0, 3)</td>
<td>345.50 sec.</td>
<td>208.00 sec.</td>
</tr>
<tr>
<td>2947</td>
<td>2947 Ambiguous Relations</td>
<td>(2 2 2 2) =&gt; (73, 79)</td>
<td>151.42 sec.</td>
<td>91.00 sec.</td>
</tr>
<tr>
<td>2946</td>
<td>2946 Ambiguous Relations</td>
<td>(1 1 2 0) =&gt; (1, 4)</td>
<td>112.53 sec.</td>
<td>68.00 sec.</td>
</tr>
<tr>
<td>2945</td>
<td>2945 Ambiguous Relations</td>
<td>(2 2 2 2) =&gt; (72, 79)</td>
<td>169.12 sec.</td>
<td>102.00 sec.</td>
</tr>
<tr>
<td>2943</td>
<td>2943 Ambiguous Relations</td>
<td>(2 1 2 2) =&gt; (0, 5)</td>
<td>318.77 sec.</td>
<td>192.00 sec.</td>
</tr>
<tr>
<td>2942</td>
<td>2942 Ambiguous Relations</td>
<td>(2 2 2 2) =&gt; (70, 79)</td>
<td>140.60 sec.</td>
<td>84.00 sec.</td>
</tr>
<tr>
<td>2939</td>
<td>2939 Ambiguous Relations</td>
<td>(2 1 2 2) =&gt; (69, 79)</td>
<td>142.15 sec.</td>
<td>85.00 sec.</td>
</tr>
<tr>
<td>2938</td>
<td>2938 Ambiguous Relations</td>
<td>(2 1 2 2) =&gt; (0, 11)</td>
<td>910.52 sec.</td>
<td>550.00 sec.</td>
</tr>
</tbody>
</table>
End: 2897 ambiguous relations
Start: 2897 Ambiguous Relations
Relation asserted (2 2 2 2) => (67, 79)
CPU 128.93 sec., Real 78.00 sec.
End: 2896 ambiguous relations
Start: 2896 Ambiguous Relations
Relation asserted (2 1 2 1) => (2, 12)
CPU 153.05 sec., Real 92.00 sec.
End: 2894 ambiguous relations
Start: 2894 Ambiguous Relations
Relation asserted (2 1 2 1) => (5, 12)
CPU 462.70 sec., Real 279.00 sec.
End: 2888 ambiguous relations
Start: 2888 Ambiguous Relations
Relation asserted (2 1 2 1) => (8, 12)
CPU 305.42 sec., Real 183.00 sec.
End: 2884 ambiguous relations
Start: 2884 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 13)
CPU 150.47 sec., Real 90.00 sec.
End: 2882 ambiguous relations
Start: 2882 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 14)
CPU 1262.20 sec., Real 759.00 sec.
End: 2862 ambiguous relations
Start: 2862 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 14)
CPU 462.07 sec., Real 278.00 sec.
End: 2857 ambiguous relations
Start: 2857 Ambiguous Relations
Relation asserted (2 2 2 2) => (66, 79)
CPU 616.50 sec., Real 371.00 sec.
End: 2852 ambiguous relations
Start: 2852 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 14)
CPU 1330.40 sec., Real 801.00 sec.
End: 2837 ambiguous relations
Start: 2837 Ambiguous Relations
Relation asserted (2 2 2 2) => (61, 79)
CPU 117.37 sec., Real 71.00 sec.
End: 2836 ambiguous relations
Start: 2836 Ambiguous Relations
Relation asserted (2 2 2 2) => (8, 14)
CPU 594.80 sec., Real 358.00 sec.
End: 2826 ambiguous relations
Start: 2826 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 14)
CPU 1252.37 sec., Real 754.00 sec.
End: 2811 ambiguous relations
Start: 2811 Ambiguous Relations
Relation asserted (2 2 2 2) => (60, 79)
CPU 113.78 sec., Real 68.00 sec.
End: 2810 ambiguous relations
Start: 2810 Ambiguous Relations
Relation asserted (1 1 2 0) => (13, 15)
CPU 82.95 sec., Real 50.00 sec.
End: 2809 ambiguous relations
Start: 2809 Ambiguous Relations
Relation asserted (2 2 2 2) => (59, 79)
CPU 316.62 sec., Real 190.00 sec.
End: 2806 ambiguous relations
Start: 2806 Ambiguous Relations
Relation asserted (1 1 2 0) => (13, 16)
CPU 136.50 sec., Real 82.00 sec.
End: 2804 ambiguous relations
Start: 2804 Ambiguous Relations
Relation asserted (2 1 2 2) => (14, 16)
CPU 242.68 sec., Real 147.00 sec.
End: 2801 ambiguous relations
Start: 2801 Ambiguous Relations
Relation asserted (2 2 2 2) => (58, 79)
CPU 106.32 sec., Real 64.00 sec.
End: 2800 ambiguous relations
Start: 2800 Ambiguous Relations
Relation asserted (1 1 2 0) => (13, 18)
CPU 79.87 sec., Real 48.00 sec.
End: 2799 ambiguous relations
Start: 2799 Ambiguous Relations
Relation asserted (2 2 2 2) => (55, 79)
CPU 99.28 sec., Real 58.00 sec.
End: 2798 ambiguous relations
Start: 2798 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 19)
CPU 1121.20 sec., Real 674.00 sec.
End: 2778 ambiguous relations
Start: 2778 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 19)
CPU 431.95 sec., Real 259.00 sec.
End: 2773 ambiguous relations
Start: 2773 Ambiguous Relations
Relation asserted (2 2 2 2) => (54, 79)
CPU 96.55 sec., Real 58.00 sec.
End: 2772 ambiguous relations
Start: 2772 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 19)
CPU 1242.45 sec., Real 748.00 sec.
End: 2757 ambiguous relations
Start: 2757 Ambiguous Relations
Relation asserted (2 2 2 2) => (53, 79)
CPU 94.27 sec., Real 61.00 sec.
End: 2756 ambiguous relations
Start: 2756 Ambiguous Relations
Relation asserted (2 2 2 2) => (8, 19)
CPU 535.63 sec., Real 335.00 sec.
End: 2746 ambiguous relations
Start: 2746 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 19)
CPU 1167.22 sec., Real 702.00 sec.
End: 2731 ambiguous relations
Start: 2731 Ambiguous Relations
Relation asserted (2 2 2 2) => (52, 79)
CPU 134.10 sec., Real 81.00 sec.
End: 2729 ambiguous relations
Start: 2729 Ambiguous Relations
Relation asserted (2 2 2 2) => (50, 79)
CPU 563.25 sec., Real 339.00 sec.
End: 2722 ambiguous relations
Start: 2722 Ambiguous Relations
Relation asserted (2 1 2 1) => (13, 19)
CPU 129.07 sec., Real 78.00 sec.
End: 2720 ambiguous relations
Start: 2720 Ambiguous Relations
Relation asserted (2 1 2 2) => (14, 19)
CPU 128.83 sec., Real 78.00 sec.
End: 2718 ambiguous relations
Start: 2718 Ambiguous Relations
Relation asserted (2 2 2 2) => (13, 21)
CPU 1080.30 sec., Real 650.00 sec.
End: 2700 ambiguous relations
Start: 2700 Ambiguous Relations
Relation asserted (2 2 2 2) => (19, 21)
CPU 212.03 sec., Real 128.00 sec.
End: 2697 ambiguous relations
Start: 2697 Ambiguous Relations
Relation asserted (2 2 2 2) => (43, 79)
CPU 124.92 sec., Real 75.00 sec.
End: 2695 ambiguous relations
Start: 2695 Ambiguous Relations
Relation asserted (2 2 2 2) => (42, 79)
CPU 73.48 sec., Real 44.00 sec.
End: 2694 ambiguous relations
Start: 2694 Ambiguous Relations
Relation asserted (2 1 2 2) => (20, 22)
CPU 121.47 sec., Real 73.00 sec.
End: 2692 ambiguous relations
Start: 2692 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 24)
CPU 237.93 sec., Real 143.00 sec.
End: 2688 ambiguous relations
Start: 2688 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 24)
CPU 83.68 sec., Real 51.00 sec.
End: 2687 ambiguous relations
Start: 2687 Ambiguous Relations
Relation asserted (2 2 2 2) => (40, 79)
CPU 70.47 sec., Real 42.00 sec.
End: 2686 ambiguous relations
Start: 2686 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 24)
CPU 244.92 sec., Real 147.00 sec.
End: 2683 ambiguous relations
Start: 2683 Ambiguous Relations
Relation asserted (2 2 2 2) => (39, 79)
CPU 202.40 sec., Real 122.00 sec.
End: 2680 ambiguous relations
Start: 2680 Ambiguous Relations
Relation asserted (2 2 2 2) => (8, 24)
CPU 128.98 sec., Real 78.00 sec.
End: 2678 ambiguous relations
Start: 2678 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 24)
CPU 228.90 sec., Real 138.00 sec.
End: 2675 ambiguous relations
Start: 2675 Ambiguous Relations
Relation asserted (2 2 2 2) => (36, 79)
CPU 117.77 sec., Real 70.00 sec.
End: 2673 ambiguous relations
Start: 2673 Ambiguous Relations
Relation asserted (2 2 2 2) => (34, 79)
CPU 61.95 sec., Real 37.00 sec.
End: 2672 ambiguous relations
Start: 2672 Ambiguous Relations
Relation asserted (2 2 2 2) => (13, 24)
CPU 321.13 sec., Real 193.00 sec.
End: 2666 ambiguous relations
Start: 2666 Ambiguous Relations
Relation asserted (2 2 2 2) => (19, 24)
CPU 68.07 sec., Real 41.00 sec.
End: 2665 ambiguous relations
Start: 2665 Ambiguous Relations
Relation asserted (2 2 2 2) => (33, 79)
CPU 60.05 sec., Real 36.00 sec.
End: 2664 ambiguous relations
Start: 2664 Ambiguous Relations
Relation asserted (2 2 2 2) => (20, 24)
CPU 116.22 sec., Real 70.00 sec.
End: 2662 ambiguous relations
Start: 2662 Ambiguous Relations
Relation asserted (2 2 2 2) => (22, 24)
CPU 64.78 sec., Real 39.00 sec.
End: 2661 ambiguous relations
Start: 2661 Ambiguous Relations
Relation asserted (2 2 2 2) => (32, 79)
CPU 58.78 sec., Real 36.00 sec.
End: 2660 ambiguous relations
Start: 2660 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 25)
CPU 624.88 sec., Real 376.00 sec.
End: 2648 ambiguous relations
Start: 2648 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 25)
CPU 231.10 sec., Real 139.00 sec.
End: 2645 ambiguous relations
Start: 2645 Ambiguous Relations
Relation asserted (2 2 2 2) => (31, 79)
CPU 112.62 sec., Real 68.00 sec.
End: 2643 ambiguous relations
Start: 2643 Ambiguous Relations
Relation asserted (2 2 2 2) => (29, 79)
CPU 306.95 sec., Real 185.00 sec.
End: 2635 ambiguous relations
Start: 2635 Ambiguous Relations
Relation asserted (2 2 2 2) => (26, 79)
CPU 726.32 sec., Real 437.00 sec.
End: 2615 ambiguous relations
Start: 2615 Ambiguous Relations
Relation asserted (2 2 2 2) => (24, 79)
CPU 25.32 sec., Real 15.00 sec.
End: 2614 ambiguous relations
Start: 2614 Ambiguous Relations
Relation asserted (2 2 2 2) => (5, 25)
CPU 642.35 sec., Real 386.00 sec.
End: 2605 ambiguous relations
Start: 2605 Ambiguous Relations
Relation asserted (2 2 2 2) => (23, 79)
CPU 24.17 sec., Real 14.00 sec.
End: 2604 ambiguous relations
Start: 2604 Ambiguous Relations
Relation asserted (2 2 2 2) => (8, 25)
CPU 306.87 sec., Real 185.00 sec.
End: 2598 ambiguous relations
Start: 2598 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 25)
CPU 601.80 sec., Real 362.00 sec.
End: 2589 ambiguous relations
Start: 2589 Ambiguous Relations
Relation asserted (2 2 2 2) => (76, 78)
CPU 143.05 sec., Real 86.00 sec.
End: 2588 ambiguous relations
Start: 2588 Ambiguous Relations
Relation asserted (2 2 2 2) => (13, 25)
CPU 946.05 sec., Real 569.00 sec.
End: 2570 ambiguous relations
Start: 2570 Ambiguous Relations
Relation asserted (2 2 2 2) => (19, 25)
CPU 181.45 sec., Real 109.00 sec.
End: 2567 ambiguous relations
Start: 2567 Ambiguous Relations
Relation asserted (2 2 2 2) => (74, 78)
CPU 132.78 sec., Real 80.00 sec.
End: 2566 ambiguous relations

One Hundred Interval Test Case

; loading "test100.lsp"
System initialized.
Relation asserted (2 1 2 2) => (0, 1)
Relation asserted (2 2 2 2) => (1, 2)
Relation asserted (1 1 1 1) => (2, 3)
Relation asserted (1 1 1 1) => (3, 4)
Relation asserted (0 1 2 2) => (4, 5)
Relation asserted (1 1 2 1) => (5, 6)
Relation asserted (2 1 2 1) => (6, 7)
Relation asserted (1 1 2 1) => (7, 8)
Relation asserted (2 0 2 2) => (8, 9)
Relation asserted (2 0 2 2) => (9, 10)
Relation asserted (0 1 2 2) => (10, 11)
Relation asserted (0 1 2 0) => (11, 12)
Relation asserted (2 0 2 2) => (12, 13)
Relation asserted (0 1 2 2) => (13, 14)
Relation asserted (0 1 2 2) => (14, 15)
Relation asserted (1 1 2 1) => (15, 16)
Relation asserted (2 0 2 2) => (16, 17)
Relation asserted (1 1 1 1) => (17, 18)
Relation asserted (0 1 2 0) => (18, 19)
Relation asserted (2 1 2 1) => (19, 20)
Relation asserted (2 1 2 0) => (20, 21)
Relation asserted (2 1 2 1) => (21, 22)
Relation asserted (2 1 2 2) => (22, 23)
Relation asserted (1 1 2 0) => (23, 24)
Relation asserted (2 1 2 1) => (24, 25)
Relation asserted (0 1 2 2) => (25, 26)
Relation asserted (0 1 2 2) => (26, 27)
Relation asserted (1 1 2 2) => (27, 28)
Relation asserted (0 1 2 0) => (28, 29)
Relation asserted (2 1 2 0) => (29, 30)
Relation asserted (1 1 1 1) => (30, 31)
Relation asserted (1 1 2 2) => (31, 32)
Relation asserted (2 0 2 2) => (32, 33)
Relation asserted (1 1 2 1) => (33, 34)
Relation asserted (1 1 2 1) => (34, 35)
Relation asserted (1 1 0 1) => (35, 36)
Relation asserted (1 1 2 1) => (36, 37)
Relation asserted (1 1 1 1) => (37, 38)
Relation asserted (2 2 2 2) => (38, 39)
Relation asserted (1 1 2 2) => (39, 40)
Relation asserted (2 1 2 1) => (40, 41)
Relation asserted (1 1 2 2) => (41, 42)
Relation asserted (0 1 2 1) => (42, 43)
Relation asserted (2 2 2 2) => (43, 44)
Relation asserted (2 2 2 2) => (44, 45)
Relation asserted (1 1 2 0) => (45, 46)
Relation asserted (0 1 2 1) => (46, 47)
Relation asserted (2 2 2 2) => (47, 48)
Relation asserted (1 1 2 0) => (48, 49)
Relation asserted (2 1 2 2) => (49, 50)
Relation asserted (1 1 1 1) => (50, 51)
Relation asserted (2 0 2 2) => (51, 52)
Relation asserted (2 1 2 2) => (52, 53)
Relation asserted (2 2 2 2) => (53, 54)
Relation asserted (2 0 2 2) => (54, 55)
Relation asserted (0 1 2 2) => (55, 56)
Relation asserted (2 1 2 0) => (56, 57)
Relation asserted (0 1 2 0) => (57, 58)
Relation asserted (2 2 2 2) => (58, 59)
Relation asserted (0 1 2 0) => (59, 60)
Relation asserted (1 1 2 2) => (60, 61)
Relation asserted (1 1 1 1) => (61, 62)
Relation asserted (1 1 0 1) => (62, 63)
Relation asserted (0 1 2 1) => (63, 64)
Relation asserted (1 1 2 0) => (64, 65)
Relation asserted (1 1 2 2) => (65, 66)
Relation asserted (2 2 2 2) => (66, 67)
Relation asserted (2 0 2 2) => (67, 68)
Relation asserted (0 1 2 0) => (68, 69)
Relation asserted (1 1 2 0) => (69, 70)
Relation asserted (0 1 2 0) => (70, 71)
Relation asserted (1 1 2 2) => (71, 72)
Relation asserted (2 1 2 2) => (72, 73)
Relation asserted (0 1 2 0) => (73, 74)
Relation asserted (2 1 2 0) => (74, 75)
Relation asserted (0 1 2 1) => (75, 76)
Relation asserted (2 1 2 1) => (76, 77)
Relation asserted (0 1 2 1) => (77, 78)
Relation asserted (2 1 2 1) => (78, 79)
Relation asserted (2 2 2 2) => (79, 80)
Relation asserted (2 1 2 1) => (80, 81)
Relation asserted (2 1 2 1) => (81, 82)
Relation asserted (2 1 2 1) => (82, 83)
Relation asserted (2 2 2 2) => (83, 84)
Relation asserted (2 1 2 0) => (84, 85)
Relation asserted (2 0 2 2) => (85, 86)
Relation asserted (1 1 2 2) => (86, 87)
Relation asserted (1 1 0 1) => (87, 88)
Relation asserted (0 1 2 1) => (88, 89)
Relation asserted (0 1 2 2) => (89, 90)
Relation asserted (0 1 2 0) => (90, 91)
Relation asserted (0 1 2 1) => (91, 92)
Relation asserted (0 1 2 0) => (92, 93)
Relation asserted (2 2 2 2) => (93, 94)
Relation asserted (2 1 2 1) => (94, 95)
Relation asserted (2 2 2 2) => (95, 96)
Relation asserted (2 1 2 0) => (96, 97)
Relation asserted (1 1 1 1) => (97, 98)
Relation asserted (2 2 2 2) => (98, 99)

There are 4521 ambiguous relations.

Start: 4521 Ambiguous Relations
Relation asserted (2 2 2 2) => (97, 99)
CPU 1377.32 sec., Real 830.00 sec.
End: 4495 ambiguous relations

Start: 4495 Ambiguous Relations
Relation asserted (2 2 2 2) => (97, 99)
CPU 210.77 sec., Real 127.00 sec.
End: 4494 ambiguous relations

Start: 4494 Ambiguous Relations
Relation asserted (2 2 2 2) => (97, 99)
CPU 204.22 sec., Real 123.00 sec.
End: 4493 ambiguous relations

Start: 4493 Ambiguous Relations
Relation asserted (2 2 2 2) => (86, 99)
CPU 216.18 sec., Real 130.00 sec.
End: 4492 ambiguous relations
Start: 4492 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 3)
CPU 202.88 sec., Real 123.00 sec.
End: 4491 ambiguous relations
Start: 4491 Ambiguous Relations
Relation asserted (2 2 2 2) => (71, 99)
CPU 269.17 sec., Real 162.00 sec.
End: 4489 ambiguous relations
Start: 4489 Ambiguous Relations
Relation asserted (2 2 2 2) => (69, 99)
CPU 266.32 sec., Real 161.00 sec.
End: 4487 ambiguous relations
Start: 4487 Ambiguous Relations
Relation asserted (2 2 2 2) => (65, 99)
CPU 185.93 sec., Real 112.00 sec.
End: 4486 ambiguous relations
Start: 4486 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 4)
CPU 312.13 sec., Real 188.00 sec.
End: 4484 ambiguous relations
Start: 4484 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 4)
CPU 311.88 sec., Real 187.00 sec.
End: 4482 ambiguous relations
Start: 4482 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 6)
CPU 200.43 sec., Real 121.00 sec.
End: 4481 ambiguous relations
Start: 4481 Ambiguous Relations
Relation asserted (2 2 2 2) => (64, 99)
CPU 264.15 sec., Real 159.00 sec.
End: 4479 ambiguous relations
Start: 4479 Ambiguous Relations
Relation asserted (2 2 2 2) => (62, 99)
CPU 176.43 sec., Real 106.00 sec.
End: 4478 ambiguous relations
Start: 4478 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 6)
CPU 197.88 sec., Real 119.00 sec.
End: 4477 ambiguous relations
Start: 4477 Ambiguous Relations
Relation asserted (2 2 2 2) => (61, 99)
CPU 1388.03 sec., Real 836.00 sec.
End: 4468 ambiguous relations
Start: 4468 Ambiguous Relations
Relation asserted (1 1 2 0) => (4, 6)
CPU 195.18 sec., Real 118.00 sec.
End: 4467 ambiguous relations
Start: 4467 Ambiguous Relations
Relation asserted (2 2 2 2) => (60, 99)
CPU 234.05 sec., Real 142.00 sec.
End: 4465 ambiguous relations
Start: 4465 Ambiguous Relations
Relation asserted (2 2 2 2) => (50, 99)
CPU 943.52 sec., Real 575.00 sec.
End: 4458 ambiguous relations
Start: 4458 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 7)
CPU 1613.32 sec., Real 972.00 sec.
End: 4449 ambiguous relations
Start: 4449 Ambiguous Relations
Relation asserted (2 2 2 2) => (48, 99)
CPU 133.07 sec., Real 81.00 sec.
End: 4448 ambiguous relations
Start: 4448 Ambiguous Relations
Relation asserted (2 2 2 2) => (45, 99)
CPU 1599.23 sec., Real 963.00 sec.
End: 4438 ambiguous relations
Start: 4438 Ambiguous Relations
Relation asserted (2 1 2 1) => (2, 7)
CPU 521.07 sec., Real 314.00 sec.
End: 4434 ambiguous relations
Start: 4434 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 8)
CPU 173.85 sec., Real 105.00 sec.
End: 4433 ambiguous relations
Start: 4433 Ambiguous Relations
Relation asserted (2 2 2 2) => (41, 99)
CPU 367.90 sec., Real 221.00 sec.
End: 4430 ambiguous relations
Start: 4430 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 8)
CPU 172.30 sec., Real 103.00 sec.
End: 4429 ambiguous relations
Start: 4429 Ambiguous Relations
Relation asserted (2 2 2 2) => (39, 99)
CPU 120.18 sec., Real 72.00 sec.
End: 4428 ambiguous relations
Start: 4428 Ambiguous Relations
Relation asserted (2 1 2 1) => (2, 8)
CPU 7607.93 sec., Real 4582.00 sec.
End: 4383 ambiguous relations
Start: 4383 Ambiguous Relations
Relation asserted (2 2 2 2) => (37, 99)
CPU 114.48 sec., Real 69.00 sec.
End: 4382 ambiguous relations
Start: 4382 Ambiguous Relations
Relation asserted (2 1 2 2) => (7, 9)
CPU 906.20 sec., Real 546.00 sec.
End: 4374 ambiguous relations
Start: 4374 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 16)
CPU 182.75 sec., Real 110.00 sec.
End: 4373 ambiguous relations
Start: 4373 Ambiguous Relations
Relation asserted (2 2 2 2) => (36, 99)
CPU 112.20 sec., Real 68.00 sec.
End: 4372 ambiguous relations
Start: 4372 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 16)
CPU 181.07 sec., Real 109.00 sec.
End: 4371 ambiguous relations
Start: 4371 Ambiguous Relations
Relation asserted (2 2 2 2) => (35, 99)
CPU 111.10 sec., Real 67.00 sec.
End: 4370 ambiguous relations
Start: 4370 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 16)
CPU 873.78 sec., Real 526.00 sec.
End: 4365 ambiguous relations
Start: 4365 Ambiguous Relations
Relation asserted (2 2 2 2) => (34, 99)
CPU 206.53 sec., Real 124.00 sec.
End: 4363 ambiguous relations
Start: 4363 Ambiguous Relations
Relation asserted (2 2 2 2) => (33, 99)
CPU 105.03 sec., Real 63.00 sec.
End: 4362 ambiguous relations
Start: 4362 Ambiguous Relations
Relation asserted (2 2 2 2) => (7, 16)
CPU 275.87 sec., Real 166.00 sec.
End: 4360 ambiguous relations
Start: 4360 Ambiguous Relations
Relation asserted (2 2 2 2) => (9, 16)
CPU 167.83 sec., Real 101.00 sec.
End: 4359 ambiguous relations
Start: 4359 Ambiguous Relations
Relation asserted (2 2 2 2) => (31, 99)
CPU 103.72 sec., Real 62.00 sec.
End: 4358 ambiguous relations
Start: 4358 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 16)
CPU 478.23 sec., Real 287.00 sec.
End: 4354 ambiguous relations
Start: 4354 Ambiguous Relations
Relation asserted (1 1 2 0) => (14, 16)
CPU 161.45 sec., Real 97.00 sec.
End: 4353 ambiguous relations
Start: 4353 Ambiguous Relations
Relation asserted (2 2 2 2) => (30, 99)
CPU 295.50 sec., Real 178.00 sec.
End: 4350 ambiguous relations
Start: 4350 Ambiguous Relations
Relation asserted (2 1 2 2) => (13, 17)
CPU 475.00 sec., Real 288.00 sec.
End: 4347 ambiguous relations
Start: 4347 Ambiguous Relations
Relation asserted (2 2 2 2) => (27, 99)
CPU 403.23 sec., Real 245.00 sec.
End: 4343 ambiguous relations
Start: 4343 Ambiguous Relations
Relation asserted (2 2 2 2) => (23, 99)
CPU 595.68 sec., Real 359.00 sec.
End: 4337 ambiguous relations
Start: 4337 Ambiguous Relations
Relation asserted (2 2 2 2) => (17, 99)
CPU 1470.15 sec., Real 885.00 sec.
End: 4319 ambiguous relations
Start: 4319 Ambiguous Relations
Relation asserted (2 2 2 2) => (95, 98)
CPU 1288.07 sec., Real 776.00 sec.
End: 4295 ambiguous relations
Start: 4295 Ambiguous Relations
Relation asserted (2 2 2 2) => (87, 98)
CPU 592.93 sec., Real 357.00 sec.
End: 4292 ambiguous relations
Start: 4292 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 18)
CPU 249.12 sec., Real 150.00 sec.
End: 4290 ambiguous relations
Start: 4290 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 18)
CPU 248.10 sec., Real 149.00 sec.
End: 4288 ambiguous relations
Start: 4288 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 18)
CPU 1169.68 sec., Real 705.00 sec.
End: 4278 ambiguous relations
Start: 4278 Ambiguous Relations
Relation asserted (2 2 2 2) => (7, 18)
CPU 497.95 sec., Real 300.00 sec.
End: 4274 ambiguous relations
Start: 4274 Ambiguous Relations
Relation asserted (2 2 2 2) => (9, 18)
CPU 248.12 sec., Real 149.00 sec.
End: 4272 ambiguous relations
Start: 4272 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 18)
CPU 899.05 sec., Real 541.00 sec.
End: 4264 ambiguous relations
Start: 4264 Ambiguous Relations
Relation asserted (1 1 2 0) => (14, 18)
CPU 497.43 sec., Real 299.00 sec.
End: 4260 ambiguous relations
Start: 4260 Ambiguous Relations
Relation asserted (1 1 2 1) => (15, 18)
CPU 246.75 sec., Real 148.00 sec.
End: 4258 ambiguous relations
Start: 4258 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 20)
CPU 243.72 sec., Real 147.00 sec.
End: 4256 ambiguous relations
Start: 4256 Ambiguous Relations
Relation asserted (2 2 2 2) => (1, 20)
CPU 243.38 sec., Real 147.00 sec.
End: 4254 ambiguous relations
Start: 4254 Ambiguous Relations
Relation asserted (2 2 2 2) => (2, 20)
CPU 1148.93 sec., Real 692.00 sec.
End: 4244 ambiguous relations
Start: 4244 Ambiguous Relations
Relation asserted (2 2 2 2) => (7, 20)
CPU 488.57 sec., Real 294.00 sec.
End: 4240 ambiguous relations
Start: 4240 Ambiguous Relations
Relation asserted (2 2 2 2) => (9, 20)
CPU 243.02 sec., Real 146.00 sec.
End: 4238 ambiguous relations
Start: 4238 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 20)
CPU 738.15 sec., Real 445.00 sec.
End: 4232 ambiguous relations
Start: 4232 Ambiguous Relations
Relation asserted (2 1 2 2) => (13, 20)  
CPU 1632.25 sec., Real 982.00 sec.  
End: 4221 ambiguous relations  
Start: 4221 Ambiguous Relations  
Relation asserted (2 2 2 2) => (86, 98)  
CPU 596.03 sec., Real 359.00 sec.  
End: 4218 ambiguous relations  
Start: 4218 Ambiguous Relations  
Relation asserted (2 1 2 1) => (17, 20)  
CPU 448.03 sec., Real 269.00 sec.  
End: 4215 ambiguous relations  
Start: 4215 Ambiguous Relations  
Relation asserted (2 2 2 2) => (71, 98)  
CPU 526.08 sec., Real 317.00 sec.  
End: 4209 ambiguous relations  
Start: 4209 Ambiguous Relations  
Relation asserted (2 2 2 2) => (69, 98)  
CPU 532.75 sec., Real 321.00 sec.  
End: 4203 ambiguous relations  
Start: 4203 Ambiguous Relations  
Relation asserted (2 2 2 2) => (65, 98)  
CPU 506.05 sec., Real 304.00 sec.  
End: 4200 ambiguous relations  
Start: 4200 Ambiguous Relations  
Relation asserted (2 2 2 2) => (0, 22)  
CPU 492.00 sec., Real 296.00 sec.  
End: 4197 ambiguous relations  
Start: 4197 Ambiguous Relations  
Relation asserted (2 2 2 2) => (64, 98)  
CPU 561.28 sec., Real 338.00 sec.  
End: 4191 ambiguous relations  
Start: 4191 Ambiguous Relations  
Relation asserted (2 2 2 2) => (62, 98)  
CPU 478.50 sec., Real 288.00 sec.  
End: 4188 ambiguous relations  
Start: 4188 Ambiguous Relations  
Relation asserted (2 2 2 2) => (1, 22)  
CPU 486.37 sec., Real 293.00 sec.  
End: 4185 ambiguous relations  
Start: 4185 Ambiguous Relations  
Relation asserted (2 2 2 2) => (61, 98)  
CPU 3628.93 sec., Real 2188.00 sec.  
End: 4158 ambiguous relations  
Start: 4158 Ambiguous Relations  
Relation asserted (2 2 2 2) => (2, 22)  
CPU 2280.47 sec., Real 1373.00 sec.  
End: 4143 ambiguous relations  
Start: 4143 Ambiguous Relations
Relation asserted (2 2 2 2) => (60, 98)
CPU 516.08 sec., Real 310.00 sec.
End: 4137 ambiguous relations
Start: 4137 Ambiguous Relations
Relation asserted (2 2 2 2) => (50, 98)
CPU 2414.32 sec., Real 1455.00 sec.
End: 4116 ambiguous relations
Start: 4116 Ambiguous Relations
Relation asserted (2 2 2 2) => (7, 22)
CPU 623.77 sec., Real 376.00 sec.
End: 4110 ambiguous relations
Start: 4110 Ambiguous Relations
Relation asserted (2 2 2 2) => (9, 22)
CPU 440.30 sec., Real 265.00 sec.
End: 4107 ambiguous relations
Start: 4107 Ambiguous Relations
Relation asserted (2 2 2 2) => (48, 98)
CPU 353.17 sec., Real 212.00 sec.
End: 4104 ambiguous relations
Start: 4104 Ambiguous Relations
Relation asserted (2 2 2 2) => (10, 22)
CPU 1095.12 sec., Real 659.00 sec.
End: 4094 ambiguous relations
Start: 4094 Ambiguous Relations
Relation asserted (2 2 2 2) => (13, 23)
CPU 1216.88 sec., Real 733.00 sec.
End: 4082 ambiguous relations
Start: 4082 Ambiguous Relations
Relation asserted (2 2 2 2) => (17, 23)
CPU 226.73 sec., Real 137.00 sec.
End: 4080 ambiguous relations
Start: 4080 Ambiguous Relations
Relation asserted (2 2 2 2) => (20, 23)
CPU 225.97 sec., Real 136.00 sec.
End: 4078 ambiguous relations
Start: 4078 Ambiguous Relations
Relation asserted (2 2 2 2) => (21, 23)
CPU 226.05 sec., Real 136.00 sec.
End: 4076 ambiguous relations
Start: 4076 Ambiguous Relations
Relation asserted (2 2 2 2) => (22, 24)
CPU 130.10 sec., Real 79.00 sec.
End: 4075 ambiguous relations
Start: 4075 Ambiguous Relations
Relation asserted (2 2 2 2) => (45, 98)
CPU 338.65 sec., Real 204.00 sec.
End: 4072 ambiguous relations
Start: 4072 Ambiguous Relations
Relation asserted (2 2 2 2) => (0, 25)  
CPU 588.28 sec., Real 354.00 sec.  
End: 4066 ambiguous relations  
Start: 4066 Ambiguous Relations  
Relation asserted (2 2 2 2) => (1, 25)  
CPU 586.85 sec., Real 354.00 sec.  
End: 4060 ambiguous relations  
Start: 4060 Ambiguous Relations  
Relation asserted (2 2 2 2) => (2, 25)  
CPU 3260.65 sec., Real 1963.00 sec.  
End: 4030 ambiguous relations  
Start: 4030 Ambiguous Relations  
Relation asserted (2 2 2 2) => (7, 25)  
CPU 1152.85 sec., Real 694.00 sec.  
End: 4018 ambiguous relations  
Start: 4018 Ambiguous Relations  
Relation asserted (2 2 2 2) => (9, 25)  
CPU 573.15 sec., Real 346.00 sec.  
End: 4012 ambiguous relations  
Start: 4012 Ambiguous Relations  
Relation asserted (2 2 2 2) => (10, 25)  
CPU 1852.67 sec., Real 1116.00 sec.  
End: 3994 ambiguous relations  
Start: 3994 Ambiguous Relations  
Relation asserted (2 2 2 2) => (13, 25)  
CPU 4520.13 sec., Real 2722.00 sec.  
End: 3957 ambiguous relations  
Start: 3957 Ambiguous Relations  
Relation asserted (2 2 2 2) => (41, 98)  
CPU 852.83 sec., Real 514.00 sec.  
End: 3948 ambiguous relations  
Start: 3948 Ambiguous Relations  
Relation asserted (2 2 2 2) => (17, 25)  
CPU 557.48 sec., Real 336.00 sec.  
End: 3942 ambiguous relations  
Start: 3942 Ambiguous Relations  
Relation asserted (2 2 2 2) => (20, 25)  
CPU 556.08 sec., Real 335.00 sec.  
End: 3936 ambiguous relations  
Start: 3936 Ambiguous Relations  
Relation asserted (2 2 2 2) => (21, 25)  
CPU 554.68 sec., Real 334.00 sec.  
End: 3930 ambiguous relations  
Start: 3930 Ambiguous Relations  
Relation asserted (2 1 2 1) => (23, 25)  
CPU 121.67 sec., Real 73.00 sec.  
End: 3929 ambiguous relations  
Start: 3929 Ambiguous Relations
Relation asserted (2 2 2 2) => (39, 98)
CPU 294.37 sec., Real 178.00 sec.
End: 3926 ambiguous relations
REFERENCES


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.


**Bel87** Bell, C.E., Representing and Reasoning with Disjunctive Temporal Constraints in a Point-Based Model, *University of Iowa, Department of Management Sciences*, 1987.


<table>
<thead>
<tr>
<th>Ref</th>
<th>Author(s)</th>
<th>Title</th>
<th>Conference/Volume/Publication Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>For80</td>
<td>Forbus, K.</td>
<td>Spatial and Qualitative Aspects of Reasoning About Motion</td>
<td>Proceedings of the First National Conference on Artificial Intelligence, 1980, 170-173</td>
</tr>
<tr>
<td>For87</td>
<td>Forbus, K., Neilsen, P., and Faltings, B.</td>
<td>Qualitative Kinematics: A Framework</td>
<td>Proceedings of the Tenth International Conference on Artificial Intelligence, 1987, 430-435</td>
</tr>
<tr>
<td>Fox78</td>
<td>Fox, B.</td>
<td>Planning and Scheduling for Robotic Assembly</td>
<td>Proceedings of the Third Conference on Artificial Intelligence for Space Applications, NASA Marshall Space Flight Center, Huntsville, AL, November 1987</td>
</tr>
<tr>
<td>Fre78</td>
<td>Freuder, E.C.</td>
<td>Synthesizing Constraint Expressions</td>
<td>Communications of the ACM 21 (1), November 1978, 958-966</td>
</tr>
</tbody>
</table>


Tsa87b Tsang, E.P.K., Time Structures for AI, Proceedings of the 10th International Conference on Artificial Intelligence (Milano, Italy), Morgan Kaufmann 1987, 456-461.
Val86 Valdèz-Peréz, R.E., Spatio-Temporal Reasoning and Linear Inequalities, Artificial Intelligence Laboratory, AIM-875, MIT, Cambridge, MA, 1986


<table>
<thead>
<tr>
<th>Date</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAR 1 0 1995</td>
<td></td>
</tr>
<tr>
<td>SEP 0 4 1995</td>
<td></td>
</tr>
<tr>
<td>AUG 9 0 1995</td>
<td></td>
</tr>
<tr>
<td>DEC 2 9 1995</td>
<td></td>
</tr>
<tr>
<td>DEC 0 2 1995</td>
<td></td>
</tr>
<tr>
<td>SEP 2 8 1998</td>
<td></td>
</tr>
<tr>
<td>NOV 2 4 1998</td>
<td></td>
</tr>
<tr>
<td>DEC 2 1 1998</td>
<td></td>
</tr>
<tr>
<td>AUG 2 0 2007</td>
<td></td>
</tr>
<tr>
<td>JUL 2 7 2007</td>
<td></td>
</tr>
</tbody>
</table>

261-2500 Printed in USA