UNIVERSITY OF CRETE

Computer Science Department

A Temporal Algebra supporting Indeterminacy

Ioannis Makridakis
Master's Thesis

Heraklion, December 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΙΑ ΑΛΓΕΒΡΑ ΧΡΟΝΙΚΟΥ ΛΟΓΙΣΜΟΥ ΜΕ ΥΠΟΣΤΗΡΙΞΗ ΑΣΑΦΕΙΑΣ

Εργασία που υποβλήθηκε από τον
Ιωάννη Μακρυδάκη
ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Ιωάννης Μακρυδάκης, Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Δημήτρης Πλεξουσάκης, Καθηγητής, Επόπτης

Γεώργιος Γεωργακόπουλος, Αναπληρωτής Καθηγητής, Μέλος

Ιωάννης Τζίτζικας, Επίκουρος Καθηγητής, Μέλος

Δεκτή:

Άγγελος Μπίλας, Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών
Ηράκλειο, Δεκέμβριος 2011

ΜΙΑ ΑΛΓΕΒΡΑ ΧΡΟΝΙΚΟΥ ΛΟΓΙΣΜΟΥ ΜΕ ΥΠΟΣΤΗΡΙΞΗ ΑΣΑΦΕΙΑΣ

Ιωάννης Μακρυδάκης
Μεταπτυχιακή Εργασία
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Η αναπαράσταση και επεξεργασία χρονοεξαρτώμενης πληροφορίας αποτελεί μια βασική απαίτηση στα πληροφοριακά συστήματα και αποτελεί μακροχρόνιο πεδίο έρευνας. Στον χώρο των βάσεων δεδομένων η καταγραφή της δυναμικής του κόσμου που αναπαριστάται απαιτεί την καταγραφή του χρόνου κατά τον οποίο ένα δεδομένο είναι έγκυρο. Απαραίτητη είναι και η δυνατότητα επεξεργασίας του χρόνου τόσο για την διατήρηση της ακεραιότητας της χρονικής πληροφορίας όσο και για την εξαγωγή των χρονοεξαρτώμενων δεδομένων. Αρκετά μοντέλα αναπαράστασης και επεξεργασίας του χρόνου έχουν προταθεί που βασίζονται κυρίως σε λογισμό και αποτέλεσαν το έναυσμα για την κατασκευή του δι-χρονικού εννοιολογικού μοντέλου. Οι προτάσεις όμως αυτές κυρίως επικεντρώνονται στο εννοιολογικό επίπεδο ή στο επίπεδο των σχεσιακών τελεστών, δίνοντας λιγότερη προσοχή στο χρονικό μοντέλο στο φυσικό επίπεδο και στην θεωρητική θεμελίωση μέσω μιας άλγεβρας της επεξεργασίας του χρόνου. Ακόμα μικρότερη είναι η προσοχή που έχει δοθεί στην αναπαράσταση και επεξεργασία του χρόνου όταν δημιουργείται ή απαιτείται η καταγραφή του ασαφούς χρόνου εγκυρότητας ενός δεδομένου.

Σκοπός της παρούσας εργασίας αποτελεί την εισαγωγή ενός γενικού χρονικού μοντέλου αναπαράστασης του χρόνου που υποστηρίζει και την χρονική ασάφεια. Αρχικά απαιτείται η αποσαφήνιση της σημασιολογίας των τελεστών στον συσχετισμό της χρονικής ασάφειας με σαφή χρονική πληροφορία. Η χρονική άλγεβρα που προτείνουμε μέσω της θεωρητικής θεμελίωσης των τελεστών μαζί με την υπολογιστική αποτελεσματικότητα, την φιλικότητα προς τον χρήστη, και την συμπαγή αναπαράσταση είναι βασικά στοιχεία για την τεκμηρίωση συστημάτων που επεξεργάζονται το χρόνο. Η υλοποίηση του μοντέλου ως ανεξάρτητες υπομονάδες καθιστά δυνατή την χρήση του στην υλοποίηση συστημάτων είτε εσωτερικά σε συστήματα διαχείρισης βάσεων δεδομένων είτε σε επίπεδο εφαρμογών.

Επόπτης Καθηγητής: ***Δημήτρης Πλεξουσάκης***
Καθηγητής Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

A TEMPORAL ALGEBRA SUPPORTING INDETERMINACY

Ioannis Makridakis
Master's Thesis
Computer Science Department, University of Crete

Abstract

The representation and processing of time-varying information is a basic requirement in the information systems and is a longstanding area of research. In the area of databases recording the dynamics of the world represented; requires recording the time at which a fact is valid. Essential is the ability to process time for both to maintain the integrity of temporal information and for extracting time-related data. Several models of representation and processing time have been proposed mainly based on calculus and triggered the bitemporal conceptual model. However these proposals primarily focus on the conceptual level or the level of relational operators, paying less attention on the time model closer to the physical layer and theoretical foundation through algebra of processing time. Even less attention is given to the representation and processing of time that comes to question when creating or recording the indeterminate time of validity over a fact.

The purpose of this thesis is the introduction of a general temporal model for representing time that supports temporal indeterminacy. Initially demanding is to clarify the semantics of the operators in the correlation of time uncertainty with determinate temporal information. The temporal algebra we suggest with the theoretical foundation of its operators; accompanied with computational efficiency, user friendliness, and the compact representation is fundamental for the documentation of systems that process time. The implementation of the model as independent modules makes possible the use of system deployment either internally in database management systems either at the application level.

Supervisor: Dimitris Plexousakis
Professor

Acknowledgements

There are too many people, directly or indirectly, connected to this long and difficult effort; I would like to thank Professor Dimitris Plexousakis for supervising, guiding and having trust in this work. I would like to deeply thank Associate Professor Georgios Georgakopoulos and Assistant Professor Yiannis Tzitzikas members of the examination comitee on this dissertation for their valuable remarks.
I need to thank all of my friends and beloved persons that supported me during this work but especially I would like to deeply thank Maria Kasnakidi, John Manthios and Antonis Spanakis that supported me on difficult or crucial moments of this effort.
Finally I would like to thank my parents Giorgos and Gerda-Marie and my sister Rika for their tireless, constant and unconditional support during my life that continued all along this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Indeterminate Time and Temporal Algebra

Representing and handling the dynamics of an evolving world requires dealing with the fundamental concept of time due to its pervasive nature on describing phenomena. Although the temporal dimension is indissolubly bound with each event or situation of the real world, this fact is often pursuing to be ignored in information systems since a static view is sometimes good enough for capturing the problem domain we examine, but providing more complete solutions is eventually sure that we have to cope somehow with time. This urge is seen on an ongoing effort by researchers over the last thirty years to introduce temporality, overcoming philosophical questions that arise about the nature of time, and providing qualitative and quantitative enhancements on a conceptual or physical level in many application areas of computer science and artificial intelligence, such as planning, scheduling, record-keeping, multimedia representations, medical information systems, spatio-temporal databases, weather monitoring, knowledge representation and many more.

Data and time are affined in manifold ways; one of the most interesting relations is that of change of data with respect to the real world, which expresses the time validity of that fact and enables to keep a track of its history. Another crucial connection is that of tracking the changes with respect to the system. There is a need to record the associated times to maintain this knowledge in a system but that is not enough. There is also the need to be able to express queries in order to retrieve this knowledge in order to represent what is known for the past the present and the future or what was known about it in different states of the evolution of our system. In the area of databases many proposals can be found about either of the two types of data affiliation to time, most notably TSQL2 [32]or the proposals for incorporating the time dimension into the standard SQL that has not been accomplished. In the area of temporal databases, research has also been made on the conceptual level, and the internal level, so physical temporal data models have been proposed with supporting operations on those models as a calculus or algebraic extensions.

There is still a problem which arises in maintaining and operating on data that is connected with a time expressing its validity on different time periods and for which the affiliated time period is not exactly clear; thus there is an indeterminate time associated with the validity of the fact. Not much research can be found in the literature concerning this aspect. Existing proposals that take into account temporal indeterminacy usually assume that we have a knowledge of the probabilities associated with the indeterminate time period and a user may assign those probabilities to each indeterminate time [21] [16] or expressing constraints [35] accompanying the temporal data. Proposals dealing with indeterminacy also can be found in a fuzzy approach, but all this approaches focus on improving the

computational efficiency neglecting the usability or the cases such as in the medical context where probabilities are difficult to assign.

Thus we come to the topic of this thesis; temporal algebra supporting indeterminacy. Given that any operation in temporal databases has to operate on the associated time attributes we propose algebraic set operations that accommodate precise and indeterminate temporal elements for the convex and non-convex case. The aim is to equip databases with a compact and natural user-friendly way to model indeterminacy bearing in mind data expressiveness, computational efficiency and consistency of this extension in regard to other proposed temporal data models that do not support indeterminate time. One of the main advantages of our approach lies in its formal foundations, which allow us to avoid most of the problems associated with models, which are usually designed using an ad-hoc approach. Another contribution is to use the implementation of the operations accompanied with the proposed data model on the application level supporting solutions that are built over a conventional snapshot database trying to enable temporal support; since it can accommodate temporal dimensions that are modeled in the underlying DBMS with the existing support of timestamps that is offered by vendors.

1.2. Motivating Example

Indeterminacy refers to the fact that we don't know exactly when a fact was valid. Trying to clarify indeterminacy we provide an example, omitting the timestamps of the time each record was asserted in the database and we also use a part of the potential timestamp that would otherwise be used; with a granularity of minutes, just for simplicity.

The example is as follows:

John was hospitalized and was monitored with hypertension stage 2[1] during 18:00 - 20:00, he claimed also that he was in the same situation earlier the same morning during 7:00 to 14:00. We can note that there is an unknown period of time that hypertension stage

Patient	Diagnosis	Type	Valid
John	Hypertension S2	Ind	$\langle 7{:}00, 14{:}00\rangle$
John	Hypertension S2	Det	$\langle 18{:}00, 20{:}00\rangle$

(a) The snapshot of the relation with the determinate and indeterminate data

Patient	Diagnosis	Type	Valid
John	Hypertension S2	Ind	$\langle 7{:}00, 8{:}59\rangle$
John	Hypertension S2	Det	$\langle 9{:}00, 9{:}30\rangle$
John	Hypertension S2	Ind	$\langle 9{:}31, 14{:}00\rangle$
John	Hypertension S2	Det	$\langle 18{:}00, 20{:}00\rangle$

(b) The patient relation after the insertion of the extended determinate knowledge

Figure 1 An example relation with determinate and ideterminate data

[1] Hypertension, Stage 2 (high systolic blood pressure 160-179 mmHg)

2, was the medical situation John was in. There are precise periods of time and indeterminate periods that should be recorded about John.

We know that John determinately was in this situation in the period $\langle 18{:}\,00, 20{:}\,00\rangle$ but also we know by claims from John that it could be true during $\langle 7{:}\,00, 14{:}\,00\rangle$. So we might assert those facts as:

INSERT INTO Patient VALUES (“John” , “Hypertension S2”, “Determinate” , $\langle 18{:}\,00, 20{:}\,00\rangle$)
INSERT INTO Patient VALUES (“John” , “Hypertension S2”, “Indeterminate” , $\langle 7{:}\,00, 14{:}\,00\rangle$)

These assertions would be depicted as in Figure 1(a) where we can see a natural, user friendly representation of the contents of the patient relation with information that was recorded about John. A query on this database snapshot should return both tuples or we should be able to retrieve only determinate or only indeterminate tuples.

Suppose later on new information about John come along since we have a record of John’s visit to the family doctor earlier that morning that states John had hypertension stage 2 during $\langle 9{:}\,00, 9{:}\,30\rangle$. An update of the information should be performed that depicts the current complete knowledge about John. An assertion of the kind:

UPDATE Patient SET Type=”Determinate” AND Valid= $\langle 9{:}\,00, 9{:}\,30\rangle$)

In Figure 1(b) is the illustration of how the added information is expected to be in the relation provided new information was added in the database.

In the following discussions we will rely on our example in order to clarify the use of the operators that are proposed.

1.3. This Thesis

We organize the remaining of this thesis as follows. In Chapter 2 we will explore the terms that are used in the context of temporal databases which are useful for the understanding of the following. We will also examine the contributions already made in the area of temporal logics and algebra and proposed approaches that address imprecise temporal information. The purpose of this chapter is to provide the main frame in which this work steps in. We do not claim by any reason that this is a full survey of all contributions, due to the vast wealth of research over the last three decades, but those that are milestones in the area and those that are related to this work.

Following Chapter 3 will provide the assumptions we do about the timeline and the time elements and fundamental definitions of simple functions and operations on which the following work is based. Also operations and algorithms with their documentation will be provided for the common case of intervals including examples and motivation for their use. These specifications provide the theoretical basis for incorporating indeterminacy in the next chapter.

In Chapter 4 we include the indeterminate time semantics and the changes that have to be applied on the operations and the algorithms in order to accommodate determinate and indeterminate time.

Finally on Chapter 5 a discussion on the proposed model as opposed to other models and line directions for future work, also we draw our conclusion and establish the proposed merits of this work.

Chapter 2

Background and Related Work

2.1. Terms

Time is related to facts or situations and events [30]. Facts and situations are often represented with data that describe the fact or the situation that we want to illustrate but it is often that the facts change over time so they have to be related with time to enhance a complete characterization of the validity of a fact over time. Usually facts span over a time period. Events describe changes that occur on a state of a fact and mostly are related to time instances but could also have duration over the time line. It is obvious that time periods have a special role in the representation and manipulation of facts and events in order to depict the dynamics of phenomena in an abstraction of the real world in a system.

2.1.1. The concept of Time

We will first clarify briefly the concept of time. The severe task to understand the ontological status of time starting with ancient philosophers like Zeno and Aristotle, up to modern philosophers and physicists exhibits contradictory theories and intriguing debates, even about its existence. People perceive time with the subjective and relative consciousness of successive events but accepted in this paradoxical concept an absolute and objective nature, laying a common sense knowledge that helps them deal and cope effectively with the treatment of the temporal aspects of everyday life [58].

Resigning on this consensus about the absolute nature of time, there still remain some vague but important points that need examination in order to make suitable assumptions trying to specify a set of principles as a basis for a formal extension of a theory about time, with technological concerns. The first issue of concern is whether there exists a *quantum of time,* thus a time unit element that cannot be decomposed. This scenario of *discrete* elements that are considered to be isomorphic to the integers($\mathbb{Z}$) is more pervasive in the literature, urged by the intuition and the technological limitations in the measurement of time although without any confirmed evidence [27] about the existence of a limit in the sub divisibility of time. On the other hand there are advocates on the structure of the time dimension as a composition of *dense elements*, isomorphic to the rationals ($\mathbb{Q}$) or even *continuous elements* isomorphic to the reals ($\mathbb{R}$) [3]. A second point of concern is the kind of elements of the time structure. There are approaches that accept an association with *points* [59] and there are many proponents of *intervals* [38]. Some other approaches defend both *points and intervals* [7] [10]. Considering the time dimension as a structure a third core issue arises concerning the ordering of this structure. Most people perceive time as *linear* and *totally ordered*, but there are also suggestions of a *partially ordered* setting. Others even suggest non-linear structures such as *branching* time or *cyclic/periodic* time. Finally an issue

concerns the limits of the time dimension; both *bounded* and *unbounded* proposals can be found in the literature. This matter lets bounded time proposals to restrict the existence of a "next" or/and a "previous" element over the time structure.

Time granularity specifies the duration of a time unit, for example in databases the TIMESTAMP data type determines that the granularity [20] of the timeline measured is one second. Of course the time granularity depends on the real world that will be modeled in the database. The granularity can be even finer to nanoseconds or thicker grained like a day or a year. Multi granularity applications may also exist. The conversion of a granule to a smaller one is a source of indeterminacy, but we are not dealing with the matter of the granularity conversions or the sources of indeterminacy, in general.

2.1.2. Temporal Dimensions

Time varying information in the context of databases is related with time usually on more than one temporal dimension. The most prominent dimensions are that of *valid time*[2] and *transaction time*, [31], [32], [30], [28], but other dimensions as user-defined times and decision times were also proposed.

Valid time [31] is the time during which a fact is true in the modeled reality. This time could be a single instance or even a time period on the time line. A valid time concerns the past, present or the future, since we can record the expected time of the validity of a fact. This time period is independent from the time that this fact was recorded in the system. It strictly concerns the fact itself but also could be updated since the knowledge about the fact in the real world might change for the future or even the past, extending or restricting the time span. Valid time might be bounded or even unbounded if a fact holds forever. An example of a valid time is the times that a deposit was/or will be made on a bank account or the time for the theatrical play.

Transaction time [31] on the other hand is the time period during which a fact is present in the system or database. This time has a starting timepoint which expresses the event that a new fact was committed into the system and holds until an update is performed and the fact is either logically deleted or altered. The transaction time is orthogonal to valid time since they are completely independent [29]. Transaction time starting timepoint is bounded in the past by the creation time of the system and bounded to the real time present since no fact is known to be recorded in the future. This time about a fact is increasing monotonically since on any change a later transaction time is recorded. By construction transaction time has duration from its insertion in the database until its logical deletion. For example transaction time enables to know when a bank account was updated in the system, and possibly the corrections made about this update later on if a correction should occur, in this way we track all modifications that are made in the system enabling to have a history of the evolution of the knowledge recorded in it. For any fact that is time stamped along with transaction time the new insertion is considered to carry the stamp [now, Until-Changed] or $[now, +\infty]$, where now is the commit time, when an update is made then the upper endpoint changes to the current time of the update. If the update was a deletion the fact still exists in the database but is considered to be logically deleted since the "active" period is over.

[2] Real-world time; intrinsic time; Logical time; Data time or History time are synonyms of the term [31].

Other time dimensions have been proposed one of them is *user defined* times with no special semantics. Valid and transaction time are about other attributes while user defined times is an attribute on its own. This means that user defined times are actually times that a DBMS is not responsible to interpret and operate with temporal operations on such a time, it is the user that makes the interpretation and assigns semantics to it. Another time dimension proposed was *decision time* that expresses the time that an event was decided that it will occur, but this time dimension is dependent to valid time and there have been many considerations about its real semantics, since there could be a plenty of decision times [30] and further investigations have implied that it could be represented as transaction, valid or user-defined time.

In the work of Jensen and Snodgrass [29] they proposed the Bitemporal Conceptual Data Model (BCDM) trying to depict the semantics of temporal databases on a conceptual rather than a physical layer. This study enables comprehension of the various data models and time dimensions that were proposed by different researchers. They introduce this conceptual model and the TSQL2 data model [32] and algebra [50].

A proposal for an SQL extension into the temporal dimension was proposed, this proposal was not finally accepted due to objections about how this extension would be compatible with conventional SQL and some flaws in principles of the relational model that were found in the proposal such as the implicit temporal attributes involved that violated the *Information Principle* of the relational model and flaws of the implementation, for a discussion of flaws in the SQL/Temporal proposal refer to [14].

2.1.3. Temporal Indeterminacy

Temporal indeterminacy [4] [21] [31] arises when we do not have the precise information about when a fact is holds but we have a vague knowledge about the time period of its validity, stronger temporal indeterminacy arises when we do not even know if a fact was valid at all at any time period. So temporally indeterminate refers to a fact or object when we do not know when the associated period for a fact begins or ends or even we don't know the duration of its validity. It is rare that we do not have information about when the fact was committed to the system so indeterminacy is not associated with the transaction time dimension. Indeterminacy has a variety of sources and as referred in [21] among other sources it could be a matter of

1. *Granularity divergence.* We record validity in a coarser granularity than the actual fact actually happens. For example we know that a fact actually happened during a month but the exact day is unknown e.g. "John was hospitalized in March" but we don't know the exact days of March that this happened.
2. *Imprecision of Dating techniques.* We have an imprecision inherent on the methods used to extract temporal knowledge. e.g. Carbon-14 dating.
3. *Planning imprecision*. Due to imponderable factors we are unable to express the durations or the beginnings and/or endings of processes.
4. *Vague knowledge.* Only estimations can be made for the exact temporal knowledge, e.g. we don't exactly know the day that Archimedes died, or we don't exactly know the dates Mona Lisa was painted.

It is often also that the source of the temporal knowledge comes cannot be verified or is considered unreliable in scientific or legal terms. For example the times that a murder suspect claims about its presence at the scene or the times a patient claims that some of the symptoms were realized.

2.1.4. Temporal Databases

Databases are [54], [33], [15] separated into categories depending on the extension of temporal support they provide, snapshot databases are atemporal in nature probably supporting user defined times. In a *snapshot database* only current data are hold in the relations[3] describing the modeled world of the present in any of the previously mentioned temporal dimensions. We are unable to retrieve corrections that are made or the validity of facts in the past or future unless by the use of user-defined times and the use of error-prone and ad-hoc application solutions there is some temporal support. On the other hand are the *uni-temporal databases* that internally support one time dimension, in the case they support valid time they contain the history of validity of the contained facts and are called *historical databases* the relations are also referred as *assertion relations*. In case transaction time is supported we have the *rollback databases* that contain versioned relations from which we are allowed to retrieve the history of corrections that are made in the database. Finally the bi-temporal databases incorporate both time dimension that of validity and transaction time enabling a full potential for maintenance of complete knowledge about the versions and the assertions[4] that are recorded about the modeled reality and the system.

Various methods can be found in the literature that supports maintenance of temporal information [51], [15], [32], especially the most researched case of relational data models. Each fact is stamped with a data unit that expresses its temporal validity. The considerations that are made at this level are about the granularity and the kind of timestamp that will be used. There are two levels that are used for timestamping the tuple or object timestamping for object oriented approaches; and the attribute timestamping.

Tuple timestamping in relational data models is used when a relational data model respects first normal form (1NF) relations so each tuple is associated with a timestamp. This causes problems since information about same data might split into multiple tuples causing the *vertical temporal anomaly* that is a redundancy of value equivalent (referring to all non-timestamp values) tuples spreading and causing redundancy in a relation (see also the example in Figure 1). To partially solve this problem the *coalescing* method [9] that strives to keep tuples with maximal timestamp periods unifying adjacent or overlapping timestamps into one is often used, diminishing the information content in case that interval semantics want to be maintained [7]. *Attribute timestamping* although it overcomes the data redundancy caused by tuple timestamping but adding timestamps to attribute values causing Non First Normal Form (NFNF) relations but allowing introduction of more complex timestamps [6].

2.2. Related Work

We start with a brief introduction of qualitative approaches about time that have majorly influenced work on this field, we then continue with work in the context of temporal logics and deductive or constraint databases. An exploration of work on temporal databases follows and finally we address proposals that have been made for the incorporation of indeterminacy.

---

[3] Or conventional relations

[4] Referred as Asserted Versioning [33]

2.2.1. Interval and Point Algebra

The first important paper on representing and processing time was a qualitative approach of intervals by Allen's interval algebra [2]. The purpose was to reason about time and he proposed a constraint propagation algorithm that is not tractable but later work [37] [5] identified tractable subclasses of that proposed by Allen. This work contains thirteen mutual exclusive relationships between two intervals and influenced any later work on intervals quantitative or qualitative.

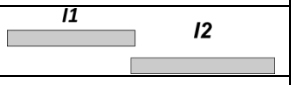
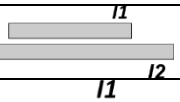
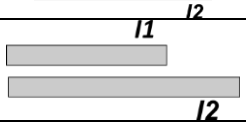
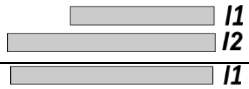
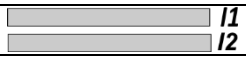
Relation	Symbol	Endpoint Constraints	Pictorial
I_1 before I_2	b	$I_1^+ < I_2^-$	
I_2 after I_1	b^{-1}		
I_1 meets I_2	m	$I_1^+ = I_2^-$	
I_2 met by I_1	m^{-1}		
I_1 overlaps I_2	o	$(I_1^- < I_2^-) \wedge (I_2^- < I_1^+) \wedge (I_1^+ < I_2^+)$	
I_2 overlapped by I_1	o^{-1}		
I_1 during I_2	d	$(I_2^- < I_1^-) \wedge (I_1^+ < I_2^+)$	
I_2 includes I_1	d^{-1}		
I_1 starts I_2	s	$(I_1^- = I_2^-) \wedge (I_1^+ < I_2^+)$	
I_2 started by I_1	s^{-1}		
I_1 finishes I_2	f	$(I_2^- < I_1^-) \wedge (I_1^+ = I_2^+)$	
I_2 finished by I_1	f^{-1}		
I_1 equalsI_2	e	$(I_1^- = I_2^-) \wedge (I_1^+ = I_2^+)$	

Table 1 : Allen's 13 mutual exclusive, interval to interval relations

On his work Allen presented intervals as primitive constructs but in aid of understanding the relationships the intervals are described in terms of their endpoints. An interval is formed by an ordered pair of endpoints $\langle I^-, I^+ \rangle$ such that $I^- < I^+$. These mutual exclusive relationships are depicted on Table 1. Allen proposes the construction of a network with known relationships and uses a propagation algorithm based on transition rules that may deduct new relationships between intervals that are added to the network. An example of a transitive rule is:

$$I_1 \; starts \; I_2 \; \wedge \; I_2 \; during \; I_3 \Rightarrow I_1 \; during \; I_3 \; .$$

The algorithm tries to detect inconsistencies, but may not detect all inconsistencies. A survey on extensions and tractable subclasses can be found in [36].

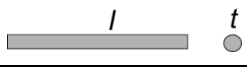
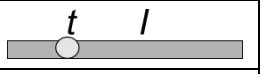
Relation	Symbol	Endpoint Constraints	Pictorial
t before I	b	$t < I^-$	
I after t	b^{-1}		
I before t	a	$I^+ < t$	
t after I	a^{-1}		
t during I	d	$(I^- < t) \wedge (t < I^+)$	
I includes t	d^{-1}		
t starts I	s	$t = I^-$	
I started by t	s^{-1}		

Table 2 : Point t to interval I relationships

In the work of Vilain and Kautz [60] that showed the incompleteness of the interval Algebra of Allen, they also devised a reference approach that introduced timepoints as

primitive constructs and allows to relate timepoints using the binary comparative qualitative relations of <, > and = and their disjunctions. The same work extended interval algebra to a point sable subclass. The relationships devised by this work or even earlier work by Vilain [59] are shown in Table 2 and Table 3.

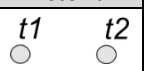
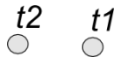
Relation	Symbol	Pictorial
t_1 before t_2	$<$	
t_2 after t_1	$>$	
t_1 after t_2	$>$	
t_2 before t_1	$<$	
t_1 equals t_2	=	

Table 3 : Timepoint t1 to timepoint t2 relationships

Aiming to support non-convex intervals work that presented taxonomy [40] of non-convex intervals and operations [41] on them has been proposed. A formal investigation on time granularities representation and conversions has also been explored [6], though we will not proceed in our work incorporating the time granularity concern.

2.2.2. Logical Frameworks about time

First order temporal logic has been introduced as an extension of first order logic that includes some modal operators from tense logic. These operators are highly used for expressing temporal integrity constraints empowering classical first order logic to take into account in its formulae states of facts or databases not only in the present state but also in the past and future. Operators enable specification of quantified propositions qualified in terms of time. The most common operators are:

Previous (●): s at time t entails p if p is satisfied in the previous state of s, meaning that p is true in the previous state of s. Example: $s_t \vDash \bullet p$
Next (○): s at time t entails p if p is satisfied in the next state of s, meaning that p is true in the next state of s. Example: $s_t \vDash \circ p$
Since (⧫): s at time t entails p since it is true on this state and any subsequent state until the state that q is true. Example: $s_t \vDash p \blacklozenge q$
Until (◊): s at time t entails p since it is true on this state and any previous state after the state where q is true. Example: $s_t \vDash p \lozenge q$

Using those operators and other operators like always in the past, or always in the future we may query in means of time a database or fact in terms of satisfying propositions [8]. Many variations of First order temporal logic are used with the most common those dealing only with the past or only with the future, namely Past Temporal Logic (PTL) and Future Temporal Logic (FTL) respectively.

Many proposals for deductive and constraint databases can be found in the bibliography that combine general purpose applications for representing knowledge or data with temporal reasoning techniques. One of the proposals is the Telos framework [45], [46], that offers an object centered knowledge representation language. In the Telos language the "proposition" is a uniform representation of the primitive constituents that are entities (referred as individuals) and attributes. Any individual represents an object while an attribute represents relationships among entities. Both individuals and attributes might be concrete or abstract facilitating a three-dimensions organization of a semantic multilevel network over classification, generalization and aggregation, with theoretically infinite levels.

This network allows specifying generic classes and multi-meta classes over abstract propositions. As far as the temporal support is concerned Telos supports transaction time[5] and valid time[6] over a linear unbound timeline and employs the mutual exclusive relationships proposed by Allen [2], however elaborating a tractable subset of its framework. The work of Plexousakis [46] introduced a simplification method for the integrity constraint rules in this framework; which rules rely on a sorted first order logic with quantifiers over the domains of classes.

2.2.3. Indeterminacy in Temporal Relational Models

In this early work of Gadia et al [22] they deal in the context of temporal relational databases with incomplete information such as missing values of attributes at points in time, or values that are unknown if they hold on certain points in time. They consider *definite tuples* and *maybe tuples* as those that are known to be true at some point in time and those that might be true, respectively. Along with their data model they define algebra to query incomplete information, in such a way that the semantics of complete information querying need no change in syntax and is maximal in the terms that it returns reliable information.

Their work is based on a tree-valued logic that except *true* and *false* may also return *undefined* and by the use of an evaluation function over a tuple to support for the selection relational operator. They introduce a pair of temporal elements containing a determinate non-convex interval l and an indeterminate one u for each tuple $\langle l, u\rangle$ where the determinate interval is a subset of the indeterminate $l \subseteq u$. They apply common set operators over temporal elements such as union, intersection, difference, complement and subset, see Table 4.

Union	$\langle l_1, u_1\rangle \cup \langle l_2, u_2\rangle = \langle l_1 \cup l_2, u_1 \cup u_2\rangle$
Intersection	$\langle l_1, u_1\rangle \cap \langle l_2, u_2\rangle = \langle l_1 \cap l_2, u_1 \cap u_2\rangle$
Difference	$\langle l_1, u_1\rangle - \langle l_2, u_2\rangle = \langle l_1 - u_2, u_1 - l_2\rangle$
Complement	$\neg\langle l_1, u_1\rangle = \langle \neg l_1, \neg u_1\rangle$
Subset	$\langle l_1, u_1\rangle \subseteq \langle l_2, u_2\rangle = \begin{cases} True, & if\ u_1 \subseteq l_2 \\ False, & if\ l_1 \subsetneq u_2 \\ Undefined, & otherwise \end{cases}$

Table 4 : Gadia et al operations on Temporal Partial Elements

Based on temporal element operations they also extend the relational operators of union, difference, projection and selection. The union $r \cup s$ of schema equivalent relations r and s that have the same key, contains tuples of the both relations except that key equivalent tuples are contained with a unified representation of their temporal elements. The difference $r - s$ of schema equivalent relations r and s that have the same key, contains tuples of r that are not in s and tuples that have a key equivalent tuple in s after a difference is applied on their partial temporal elements. For the projection no special requirements are expressed but for the selection operation a boolean expression is equipped with a partial temporal element that intersects with partial temporal elements of

[5] Referred as *belief time*

[6] Reffered as *history time*

tuples, if the intersection is not empty the result is copied to the result. With the partial temporal element we restrict the query to a specific range and the output tuple temporal elements are respectively restricted.

Dyreson and Snodgrass [21] have presented the most complete work on indeterminacy where the basic time element is an instant. The indeterminate instant is represented by a lower and an upper support and a probability mass function (a_*, a^*, P_a). In this way indeterminacy is equipped with probability and the time instant may be one time instant in the convex region enclosed by the timepoint of the lower a_* and upper a^* support with a probability weight $P_a(i)$ for each enclosed time point. The probability mass function is user defined according to the modeled domain only by its name. Based on such an instant they also suggest representations of periods and intervals.

They propose extensions in the SQL that support indeterminacy in the data and in the query. For data they propose the *correlation credibility* that replaces an indeterminate element with a certain value. On the other hand for a query the *ordering plausibility* controls the answers by controlling the relationships among the data. The user is allowed to determine both, for the correlation credibility ranges among *{indeterminate, expected, max, min}*, where for example *indeterminate* leaves untouched the indeterminacy and the values that replace indeterminate data might be the *expected value* that is the probabilistic mean. The ordering plausibility is an integer ranging in [1,100] where the greater the plausibility the greater the definiteness of the query result. For example a plausibility of 100 returns only definite answers. In this work the determinate region of each element is that where the probability reaches 1, they provide extensions to alter a table, for the selection and creation of a table, they also provide semantics for the coalescing operation where the coalesced version contains the skyline of the probabilities. Later work by [16] Dekthyar et al extended the work in probabilistic temporal databases introducing an algebra that incorporates dependent probabilities and partially known distributions. We will not extend to the Probabilistic Data Model since we are not dealing with probabilities.

Griffiths and Theodoulidis [26] introduced the SQL+i temporal relational database management system that assumes a dense timeline. They support spans, instants and periods allowing multiple granularities for determinate, indeterminate and relative time. For relative temporal expressions they allow inter-tuple temporal relationships. To support relative time they utilize a temporal constraint network (TCN) in order to maintain integrity of temporally relative facts. They use the relationships of Allen [2] and its algorithm, that rolls back the database on detected inconsistencies.

A constraint database where the temporal data model supports global and local inequality constraints for the valid time of an event is proposed in the work of Koubarakis [34], [35], in his model time is linear, dense and unbounded and in the data models he proposed a modal temporal calculus using *possible* and *certain* modal operators for the indefinite instant. Elaborating variables for instants and intervals as attribute values that are constraint globally expressing endpoints or durations of intervals.

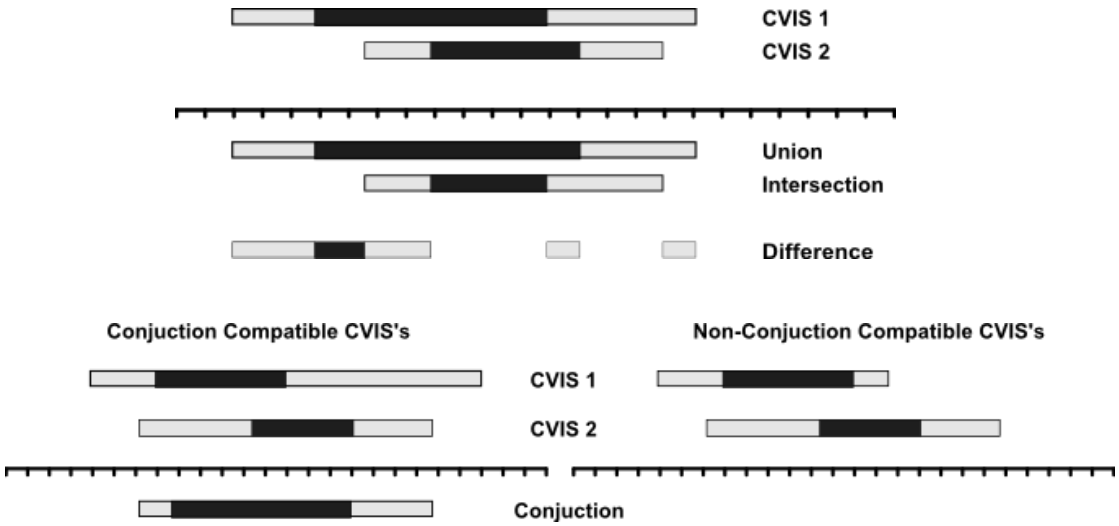


Figure 2 The Convex Valid Interval Stamps and Operations in [12]

In the work of Cowley and Plexousakis [12], [13] they devise a model where time is linear, discrete and unbounded the basic temporal element is constructed from four timestamps. Each intervals format is $\langle I_s, D_s, D_e, I_e \rangle$ where D_s and D_e represents the determinate region endpoints as starting and ending timepoint respectively. The accompanying endpoints I_s and I_e are the endpoints of the lower indeterminate and the upper indeterminate region embracing the determinate region. Operations such as conjunction, union, intersection and difference are specified. They also handle the non-convex intervals case. In their work they propose also modal extensions of the 13 relations proposed by Allen [2] for potential and definite satisfaction of the relations for the convex valid interval stamps. The model proposed may handle in the context of valid interval stamps also timepoints and fully determinate and indeterminate intervals.

In a recent paper of Anselma et al, [4] they propose a family of data models that support valid-time indeterminacy in temporal relational databases. They introduce a reference approach and extensionally devise approaches that are user-friendly, compact and computationally efficient. They evaluate each approach according to its data and query expressiveness, its consistency as an extension of the BCDM, its correctness and closure in terms of set operators. The proposed approaches deal with sets of time instants for the determinate and indeterminate case.

Our work uses a linear, discrete and unbounded time line that resembles to that proposed by Cowley and Plexousakis [12], they use the isomorphism to an extension of the set of integers with a set of three special timepoints that we also use on our approach. We use some of the ordering and containment functions they propose due to the similarities of our timeline. The indeterminacy in all proposals is encoded in the temporal elements except for the work of [34] that is encoded in the inequality constraints approaching the temporal constraint networks, as does the paper of [26] that use the TCN to enforce integrity constraints, and they both use the dense timeline. In the work of [26] they are also concerned with granularity issues and propose extensions for the SQL. Extensions are also proposed from [21] to incorporate indeterminacy in the query language. They deal with indeterminacy in the probabilistic context while rather in our approach we do not deal with probabilities but it could be seen as every operation involving indeterminate time respects a uniform probability mass function, though we don't incorporate them in the model, other proposals also are out of the scope of the probabilistic approach except [16] that extends the work of Dyreson and Snodgrass. The work of Gadia et al. incorporates relational

operators as the Anselma et al. [4] also does, we are not referring to relational operators but we determine equivalently the set of temporal operations over our data model.

The proposal that follows resembles to one of the proposed approaches of [4] in that one of their devised models where determinate and indeterminate sets of time instants are used while our temporal elements are sets of convex intervals for indeterminate and determinate time. Their evaluation for the resembling data model still holds for our model in terms of expressiveness, and compactness, that is user friendly and computationally efficient.

Chapter 3

The Time Model

3.1. Time Points

We formally represent time as a frame with structure $T = (P, <_t)$ such that P is a set of time points[7] $P = \{t_i | i \in \mathbb{Z}\} = \{\dots, t_{-1}, t_0, t_1, t_2, \dots\}$ and a binary relation $<_t$ on P. This structure forms a linear discrete time line with a second order property of homogeneity. The binary relation $<_t$ over the time points of P is called a *precedence relation* and for a set of two points t_i and t_j that satisfies the relation $t_i <_t t_j$ we say that t_i is earlier than t_j. The set of time points is considered to be isomorphic to the set of integers $\mathbb{Z}$, thus $P \cong \mathbb{Z}$ and the bijective mapping function from P to $\mathbb{Z}$ ($\iota: P \to \mathbb{Z}$) is defined as:

$$\iota(t) = i, \qquad for\ t = t_i$$

and its inverse function

$$\iota^{-1}(i) = t_i, \qquad \forall i \in \mathbb{Z}$$

providing the mapping from $\mathbb{Z}$ to P ($\iota^{-1}: \mathbb{Z} \to P$).

The precedence relation $<_t$ over P according to this bijective morphism is defined as:

$$t_i <_t t_j \Leftrightarrow \iota(t_i) < \iota(t_j) \Leftrightarrow i < j$$

so we will, for the following discussions, use the notation of the *less than* (<) binary relation over the integers assuming translations in terms of the mapping function defined above. Similar to the *earlier than* relation we are able to map every comparison binary relation over the set of integers to the set of timepoints P, avoiding once again the use of the t subscript but implicitly referring to the mapping function of the isomorphism.

In [57] the following set of complete axioms about the time line was stated, that also are assumed to hold on out structure:

- Transitivity: $\forall x, y, z \in P, x < y < z \Rightarrow x < z$
- Irreflexivity: $\forall x \in P, x \not< x$
- Linearity: $\forall x, y \in P, x < y \vee x = y \vee x > y$
- Succesion:

[7] Also referred as, *time instances* or *chronons*.

$$\forall x \in P, \exists y \ \in P, x < y \text{ (Right succession)},$$

$$\forall x \in P, \exists y \ \in P, y < x \text{ (Left succession)}$$

- Discreteness: $\forall x, y \in P, x < y \Rightarrow$

$$\exists z \in P, x < z \ \wedge \nexists u \in P, x < u < z$$

$$\wedge$$

$$\exists z \in P, z < y \ \wedge \nexists u \in P, z < u < y$$

Although the time line is considered discrete as opposed to dense the time line is continuous.

We will extend the model letting the set of timepoints T contain a set of special characters $\{-\infty, +\infty, nil\}$. The element $-\infty$ will be also referred as the *beginning of time* and $+\infty$ will carry the semantics of the *ending of time*, and will be called together as *final points*[8]. Finally the element nil will be referred as the *empty element*[9]. The use of this special character set will be restricted in the context of intervals, and will be explained further in following sections. The derived structure of time will be $T' = (P', <_{t'})$ and the semantics of the precedence binary operator over $P' = P \cup \{-\infty, +\infty, nil\}$ has to be extended.

The set P' is isomorphic to $\mathbb{Z} \cup \{-\infty, +\infty, nil\}$ by the bijective translation function $\iota' : P' \to \mathbb{Z}'$:

$$\iota'(t) = \begin{cases} t, & for\ t \in \{-\infty, +\infty, nil\} \\ \iota(t), & for\ t \in P \end{cases}$$

And the inverse function $\iota'^{-1} : \mathbb{Z}' \to P'$

$$\iota'^{-1}(i) = \begin{cases} i, & for\ i \in \{-\infty, +\infty, nil\} \\ \iota^{-1}(i), & for\ i \in \mathbb{Z} \end{cases}$$

The precedence relation $<_{t'}$ carries the same meaning as $<_t$ for elements in P, and intuitively each timepoint except the empty element lies between the final points:

$$\forall t \in P'^* = P' - \{nil\}, -\infty <_{t'} t <_{t'} + \infty$$

The *final points* do not hold the irreflexivity property and the right and left succession property breaks for $-\infty$ and $+\infty$ respectively as there are no timepoints before the beginning of time or after the ending of time. The *empty element* is incomparable, so any comparison to any timepoint in $P'^* = P' - \{nil\}$ will be undefined. The semantics of the extended precedence operator and the equality operator is shown in Table 5 and Table 6 respectively.

For simplicity in the following we will omit the subscript of the precedence relation except in cases where it has to be clear.

Since the matter of determining the earliest (latest) between two timepoints, will occur very often in the next sections, and for a timepoint under consideration there will also be the need under the succession property to determine the subsequent timepoint and/or

---

[8] The final points will try to equip endpoints of intervals allowing semantics of first order temporal logic operators *always in the past(■)* and *always in the future(□)*, in the algebra.

[9] The empty element will have the use of representing the fact "don't know if a fact happened"

immediately preceding timepoint, under the awkward presence of the special elements the definitions of functions will be needed. These are provided here.

$<_{t'}$	t_j	$-\infty$	$+\infty$	nil
t_i	As for P	False	True	Undefined
$-\infty$	True	True	True	Undefined
$+\infty$	False	False	True	Undefined
nil	Undefined	Undefined	Undefined	Undefined

Table 5 : The extended precedence operator $<_{t'}$ over P'

$=_{t'}$	t_j	$-\infty$	$+\infty$	nil
t_i	As for P	False	False	False
$-\infty$	False	True	False	False
$+\infty$	False	False	True	False
nil	False	False	False	True

Table 6 : The extended equality operator $=_{t'}$ over P'

Definition 1 The functions with signature $P' \times P' \rightarrow P'$ max and min will serve the purpose of *latest* and *earliest* timepoint respectively. The first two $max_p$and min_p will return nil if any of the two operands is nil , while the later operations $max_{vp}$and min_{vp} will return nil only in the case where both operands are nil.

$$max_p(t_i, t_j) = \begin{cases} nil, & if\ t_i = nil \vee\ t_j = nil \\ t_i, & if\ t_j < t_i \\ t_j, & otherwise \end{cases}$$

$$min_p(t_i, t_j) = \begin{cases} nil, & if\ t_i = nil \vee\ t_j = nil \\ t_i, & if\ t_i < t_j \\ t_j, & otherwise \end{cases}$$

$$max_{vp}(t_i, t_j) = \begin{cases} nil, & if\ t_i = nil \wedge\ t_j = nil \\ t_i, & if\ t_j = nil \\ t_j, & if\ t_i = nil \\ t_i, & if\ t_j < t_i \\ t_j, & otherwise \end{cases}$$

$$min_{vp}(t_i, t_j) = \begin{cases} nil, & if\ t_i = nil \wedge\ t_j = nil \\ t_i, & if\ t_j = nil \\ t_j, & if\ t_i = nil \\ t_i, & if\ t_i < t_j \\ t_j, & otherwise \end{cases}$$

Definition 2. The functions with signature $P' \to P'$ previous and next will carry the same meaning as the modal operators ●, ○ of the temporal logic. Special characters will be returned in a reflexive way while timepoints over P will make use of the isomorphism to integers.

$$previous(t) = \begin{cases} t, & if\ t \in \{-\infty, +\infty, nil\} \\ \iota^{-1}(\iota(t) - 1), & for\ t \in P \end{cases}$$

$$next(t) = \begin{cases} t, & if\ t \in \{-\infty, +\infty, nil\} \\ \iota^{-1}(\iota(t) + 1), & for\ t \in P \end{cases}$$

Definition 3. For the purpose of metrics over the time line we consider a function with signature $P' \to \mathbb{Z} \cup \{\infty\}$, defining the *duration* of any element.

$$duration_p(t) = \begin{cases} \infty, & if\ t \in \{-\infty, +\infty\} \\ 0, & t = nil \\ 1, & for\ t \in P \end{cases}$$

It is useful to extend the arithmetic operations of $\mathbb{Z}$ to the set $\mathbb{Z} \cup \{\infty\}$, such that addition, subtraction or multiplication with the one operand being infinite ∞ returns an infinite result. The duration for the timepoints over $\mathcal{P}$ implies a metric homogeneous time structure[10].

3.2. Time Intervals

Having defined the time line structure it is possible to construct sets of discrete timepoints. The most interesting and most useful among the possible sets that can be constructed by timepoints, are convex subsets of the totally ordered time domain, thus sets of consecutive timepoints: $I = \{t_s, t_{s+1}, \dots, t_{e-1}, t_e\} = \langle t_s, t_e \rangle$. Such a convex set could be anchored on the timeline, meaning that each point of the set can be precisely located over the timeline, and in this case called a *time period*[11], while a set of convex timepoints that is not anchored is called an *interval*[12], bearing the meaning of the time between the two endpoints[13] $\langle t_s, t_e \rangle$.

3.2.1. Convex Intervals

Under the semantics of either time periods or intervals we define

Definition 4. A *convex interval* is a set of consecutive time instances $\mathrm{I} = \langle \mathrm{t_s}, \mathrm{t_e} \rangle$, where the endpoints $\mathrm{t_s}, \mathrm{t_e} \in \mathrm{P}', \mathrm{t_s} \leq \mathrm{t_e}$ are members of the interval, in a both-end closed notation. The endpoint $\mathrm{t_s}$ is the *lowest* and $\mathrm{t_e}$ is the *highest* timepoint of the interval. The comparison operator of the definition allows both endpoints to be identical $\langle \mathrm{t}, \mathrm{t} \rangle$, therefore incorporating time instances to the set of intervals. Special elements of the time structure can be used, so

---

[10] We assume that every timepoint is of the same granularity, else the metrics function for time elements in $\mathcal{P}$, has to be properly modified.

[11] A *time period* was also called a *time interval*, but this definition is deprecated in order to lift any confusion with the concept of *interval.*

[12] An *interval* was also given names as *duration*, *span*, *time period*, or *time distance.*

[13] Sometimes out of the context of databases an interval is considered as a directed duration, which has specific length but no specific endpoints, therefore unanchored. Under these semantics it is possible to have positive and negative intervals denoting motion to the future or the past respectively.

in the presence of $-\infty$ as the lowest endpoint $\langle -\infty, \mathrm{t_e} \rangle$, we have a right infinite interval, while when $+\infty$ appears in the place of the highest endpoint $\langle \mathrm{t_s}, +\infty \rangle$ we regard it as a left infinite interval. If both endpoints are infinite the interval is *totally infinite* $\langle -\infty, +\infty \rangle$. Both endpoints may be nil, constructing the empty interval $\langle \mathrm{nil}, \mathrm{nil} \rangle$. The indefinability of the precedence operator over nil excludes intervals with only the one of its endpoints being the empty element. The set of all convex intervals as defined here will be referred as $\mathcal{CI}$. The set of convex intervals excluding the empty interval will be called $\mathcal{CI}^*$.

Inclusion of timepoints in a convex interval must be addressed. A timepoint $t \in \mathcal{P}$ is included ($\in_{ci}$) in a convex interval $\mathrm{I} = \langle I_s, I_e \rangle, I \in \mathcal{CI}^*$ if and only if it lies between the endpoints of I.

$$\forall t \in \mathcal{P}, \quad t \in_{ci} I \Leftrightarrow I_s \leq t \leq I_e :$$

We will remove the subscript from the inclusion operator, since convex interval inclusion carries the set theoretic meaning. It is straightforward that the totally infinite convex interval includes all timepoints over $\mathcal{P}$.

In the set of convex intervals $\mathcal{CI}^*$ the thirteen mutually exclusive binary relations of Allen's Algebra [2], as referred in Table 1 are applied with respect to the well-defined endpoints of our model.

It is meaningful to define the concepts of intersection, adjacency and mergence for convex intervals over the set of $\mathcal{CI}^*$.

Definition 5. We say that two convex intervals I_1 and I_2 *intersect* if they share at least one timepoint in common:

$$\exists t \in \mathcal{P}: t \in I_1 \wedge t \in I_2,$$

or more efficiently, making use of the total ordering of the time line:

$$I_{1_s} \leq I_{2_e} \wedge I_{2_s} \leq I_{1_e}$$

In terms of Allen's Algebra:

$$(I_1\ overlaps\ I_2) \vee (I_1\ meets\ I_2) \vee (I_1\ starts\ I_2) \vee (I_1\ finishes\ I_2) \vee$$
$$(I_1\ during\ I_2) \vee (I_1\ contains\ I_2) \vee$$
$$(I_1\ started\_by\ I_2) \vee (I_1\ finished\_by\ I_2) \vee (I_1\ met\_by\ I_2) \vee (I_1\ overlaped\_by\ I_2)$$

The empty interval $\langle \mathrm{nil}, \mathrm{nil} \rangle$ does not intersect with any interval due to the indefinability of the precedence relation. We also note that the totally infinite interval $\langle -\infty, +\infty \rangle$ intersects with any non-empty interval.

Definition 6. Adjacency for two convex intervals I_1 and I_2, holds if the subsequent of the highest endpoint of the one is the lowest endpoint of the other convex interval, that is:

$$I_{1_s} = next(I_{2_e}) \vee next(I_{1_e}) = I_{2_s},$$

Or more formally there is no timepoint between non-intersecting intervals I_1 and I_2

$$\forall x \in I_1 \forall y \in I_2,$$
$$\Big(x < y \wedge \forall z \in P, x \leq z \leq y \Rightarrow z \in I_1 \vee z \in I_2\Big) \vee$$
$$\Big(y < x \wedge \forall z \in P, y \leq z \leq x \Rightarrow z \in I_1 \vee z \in I_2\Big)$$

In terms of Allen's Algebra:

$$(I_1\ before\ I_2 \wedge \nexists I_3 (I_1\ before\ I_3) \wedge (I_3\ before\ I_2)) \vee$$

$$(I_1\ after\ I_2 \wedge \nexists I_3(I_3\ after\ I_2) \wedge (I_1\ after\ I_3))$$

Note once again that the empty interval is not adjacent to any convex interval, as the totally infinite convex interval does also.

Definition 7. If two convex intervals I_1 and I_2 either intersect or are adjacent we say that they are *mergeable*[14].

Whenever two convex intervals are able to merge we can construct a new convex interval that contains all timepoints of both. This is actually the set union for non-disjoint sets. We provide the following function.

Definition 8. The *merging* of two convex intervals I_1 and I_2, that are *mergeable* is a new convex interval $I_m = \langle I_s, I_e \rangle$ with lowest endpoint I_s the earliest of the lowest of the intervals and highest endpoint I_e the highest of the two intervals. The signature of this union is: $\mathcal{CI}^* \times \mathcal{CI}^* \rightarrow \mathcal{CI}^*$.

$$I_m = merge(I_1, I_2) = \langle I_s, I_e \rangle,\ I_s = min_{vp}\left(I_{1_s}, I_{2_s}\right),\ I_e = max_{vp}\left(I_{1_e}, I_{2_e}\right)$$

Theorem 1 . Every timepoint that is contained in any of two convex intervals I_1 and I_2 is also contained in the merged convex interval I_m: $\forall t \in \mathcal{P}, (\ t \in I_1 \vee t \in I_2 \Leftrightarrow t \in I_m)$

Proof:

Case (⇒).Let t ∈ I_1 then t is later or equal than the lowest timepoint of the interval $\mathrm{I}_{1\min}$ and earlier or equal than the latest timepoint of the interval I_{1max}:

$$I_{1min} \leq t \leq I_{1max}. \qquad (1)$$

By construction the lowest timepoint of the merged interval I_m is earlier or equal than the lowest timepoint of the interval I_{1min} and the latest timepoint of the merged interval is later or equal than the highest timepoint of the interval I_{1max}, therefore:

$$I_{mmin} \leq I_{1min} \text{ and } I_{1max} \leq I_{mmax} \text{ (2)}$$

By (1) and (2) we entail that $I_{mmin} \leq t \leq I_{mmax}$, thus $t \in I_m$.The same may be entailed for $t \in I_2$.

Case (⇐).Let $t \in I_m$ then t is later or equal than the lowest timepoint of the merged interval I_{mmin} and earlier or equal than the latest timepoint of the merged interval I_{mmax}. :

$$I_{mmin} \leq t \leq I_{mmax}$$

By construction the lowest timepoint of the merged interval I_{mmin} is either equal with the lowest timepoint of I_1 or equal with the lowest timepoint of I_2:

$$I_{mmin} = I_{1min} \text{ or } I_{mmin} = I_{2min},$$

and the latest timepoint of the merged interval is either equal with the highest timepoint of I_1 or equal with the highest timepoint of I_2:

$$I_{mmax} = I_{1max} \text{ or } I_{mmax} = I_{2max}$$

[14] The term *coalesce* [9]or *unify* can be also used

1. If $I_{mmin} = I_{1min}$ and $I_{mmax} = I_{1max}$, then it is straightforward that $t \in I_1$.
2. If $I_{mmin} = I_{2min}$ and $I_{mmax} = I_{2max}$, then it is straightforward that $t \in I_2$.
3. If $I_{mmin} = I_{1min}$ and $I_{mmax} = I_{2max}$, then either $t \leq I_{1max}$ and $t \in I_1$, or $t > I_{1max}$, but since the convex intervals are mergeable $t > I_{1max}$ also entails that $t \geq I_{2min}$, thus $t \in I_2$.
4. If $I_{mmin} = I_{2min}$ and $I_{mmax} = I_{1max}$, then either $t \leq I_{2max}$ and $t \in I_2$, or $t > I_{2max}$, but since the convex intervals are mergeable $t > I_{2max}$ also entails that $t \geq I_{1min}$, thus $t \in I_1$.

The equivalence has been proven, both ways. In the presence of the empty interval this function returns the non-empty interval and if both operands are empty it will return the empty interval.

Theorem 2 . The merging function is commutative and associative.

Proof: In the case of mergeable intervals it is straightforward from the commutative and associative nature of the definitions of max_{vp} and min_{vp}.

Theorem 3 . The identity element of merge is the empty interval.

Proof: In the case one of the merge operands is the empty interval then the operation of max_{vp} and min_{vp} returns as endpoints of the merge the endpoints of the non-empty interval.

So adjacency, intersection and mergence are well defined in the set $\mathcal{CI}$.

Measuring intervals by means of the count of the contained timepoints in an interval will be of practical use in later discussions. A metric function that returns the length of an interval is defined next.

Definition 9. The metric function $duration_i$ with signature $\mathcal{CI} \rightarrow \mathbb{Z} \cup \{\infty\}$, will return the duration of an interval $\mathrm{I} = \langle \mathrm{I_s}, \mathrm{I_e} \rangle$. Intuitively an infinite interval of any kind will return an infinite duration; the empty interval will return 0; while any other interval will return the sum of the durations of all timepoint members of the interval.

$$duration_i(I) = \begin{cases} \infty, & if\ I_s \in \{-\infty, +\infty\} \vee\ I_e \in \{-\infty, +\infty\} \\ 0, & I = \langle nil, nil \rangle \\ \sum_{I_i \in I} duration_p(I_i), & otherwise \end{cases}$$

This definition allows a time structure with non-homogeneous metric semantics, but since each timepoint of $\mathcal{P}$ has the same metric duration (see *Definition 3*) it is more efficient in terms of performance to calculate the duration of intervals whose endpoints belong to $\mathcal{P}$, as: $\sum_{I_i \in I} duration_p(I_i) = \sum_{I_i \in I} 1 = \iota(I_e) - \iota(I_s) + 1$.

3.2.2. Non Convex Intervals

Extending the concept of convex intervals to sets of convex intervals has more practical use, as it generalizes intervals and closed operators can be defined.

Definition 10. A *non-convex interval* is a set of convex intervals. This set comprises of non mergeable convex intervals, if two members are mergeable they are replaced by the coalesced version that contains both, keeping non-convex intervals as compact as possible. The set of non-convex intervals containing non-mergeable convex intervals is referred

as $NC\mathcal{I}$, while in the absence of the empty interval of its constituents will be referred as $NC\mathcal{I}^*$.

A convex interval I can be also a trivial non-convex one, the conversion is straightforward for $NC\mathcal{I}$. In any non-convex set the empty interval will be denoted as { }.

Definition 11 A non-empty convex interval I is converted to a non-convex one $N = \{I\}$ with a single member that of the convex interval itself. The empty interval is converted to the empty convex interval:

$$N = \begin{cases} \{I\}, if\ I\ non-empty \\ \{\ \}, if\ I = \langle nil, nil \rangle \end{cases}$$

We might want to check the inclusion of a convex interval in a non-convex:

Definition 12 A convex interval I is included in a non-convex interval, when there is an interval in N that equals to I, or somehow contains I. More formally:

$$\forall I \in \mathcal{CI}, I \in N \Rightarrow \exists I_n \in N, (I\ equals\ I_n) \vee (I\ during\ I_n) \vee (I\ starts\ I_n) \vee (I\ finishes\ I_n)$$

Definition 13 A time point p is included in a non-convex interval N when there is a convex member I in N that includes the timepoint: $\forall p \in P, p \in N \Leftrightarrow \exists I \in N \wedge p \in I$.

An ordering may be applied on the members of a non-convex interval for sets that do not include the empty interval. For example an interval in $NC\mathcal{I}^*$, $N = \{I_1, I_2, \dots, I_{k-1}, I_k\}$ such that , $(I_1\ before\ I_2) \wedge \dots \wedge\ (I_{k-1}\ before\ I_k)$. In this case we can define the earliest $I_1$and the latest I_k interval in the set.

Definition 14. The functions with signature $NC\mathcal{I}^* \rightarrow\ C\mathcal{I}^*$ return respectively the earliest and the latest convex member over the time line:

$$earliest(N) = \begin{cases} I_1 \in\ N \wedge \nexists I_i \in\ N, I_i\ before I_1 \\ \langle nil, nil \rangle, \quad if\ N = \{\ \} \end{cases}$$

$$latest(N) = \begin{cases} I_k \in\ N \wedge \nexists I_i \in\ N, I_i\ after\ I_k \\ \langle nil, nil \rangle, \quad if\ N = \{\ \} \end{cases}$$

A non-convex interval holds the property of the count of membership:

Definition 15 The length of a non-convex interval in terms of the number of members that are included, is a function with signature $NC\mathcal{I}^* \rightarrow \mathbb{N}$, and will be denoted as $|N|$.

$$|N| = count_{nc}(N) = \sum\nolimits_{I_i \in N} 1$$

Extending the duration function of the convex to the non-convex case:

Definition 16 The duration of a non-convex interval$N \in\ NC\mathcal{I}$, is a function with signature $NC\mathcal{I}^* \rightarrow \mathbb{Z} \cup \{\infty\}$ that returns the aggregate duration of all its constituent convex intervals:

$$duration_{nc}(N) = \sum\nolimits_{I_i \in N} duration_i(I_i)$$

Note that if the duration of one member is infinite the whole duration will be infinite; while the empty NCI exhibits zero duration.

The non-convex case exhibit topological properties, so we define the diameter, an interval that comprises from the lower timepoint of the earliest member as lower endpoint, and the latest endpoint of the latest member of the non-convex interval. This helps for a coarse-grained representation of the non-convex case.

Definition 17 The function with signature $NC\mathcal{I} \rightarrow C\mathcal{I}$, which returns the topological diameter of the region of time that contains all the convex members of the non-convex interval N; N_s is the lowest endpoint of $earliest(N)$, while N_e is the highest endpoint of $latest(N)$.

$$\delta(N) = \begin{cases} \langle nil, nil \rangle, & if\ N = \{\ \} \\ \langle N_s, N_e \rangle\ , & otherwise \end{cases}$$

We observe that if the NCI contains a left infinite the diameter will be left infinite, in case of a right infinite membership the diameter will also be right infinite. If both left and right infinite members are contained the diameter will be the total infinite interval. The empty NCI has the empty convex interval as diameter.

The NCI that contains one convex interval has a diameter that equals with the only member of the non-convex representation.

3.3. Operations on Intervals

Operations on convex and non-convex intervals are of practical use. We define the set theoretic operations of union, complement, intersection and difference. We also provide algorithms performing these tasks. For the following discussion we consider convex intervals and non-convex intervals. We will assume implicitly non-convex intervals of the set $NC\mathcal{I}^*$.

3.3.1. Convex Union

There are plenty of cases that two or more intervals have to be joined in order to get an accumulated version of the times a fact holds; therefore we provide the union of intervals.

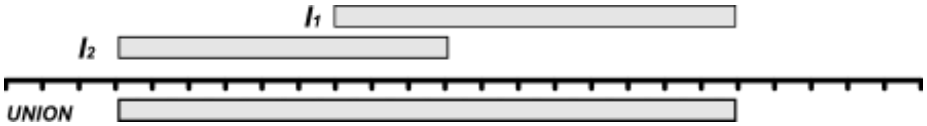


Figure 3: Convex Interval Union

Definition 18: We define the union of two convex intervals I_1 and I_2 ,$I_1 \cup^{C\mathcal{I}} I_2$ as the non-convex interval with members either the merged convex interval I_m (see *Definition 8*) if they are mergeable or both intervals otherwise, except of the case that both are the empty convex interval:

$$I_U^{C\mathcal{I}} = I_1 \bigcup^{C\mathcal{I}} I_2 = \begin{cases} \{I_m\}, & if\ I_1, I_2\ mergeable \\ \{\ \}, & if\ I_1\ and\ I_2\ are\ \langle nil, nil \rangle \\ \{I_1\}, & if\ I_2\ is\ \langle nil, nil \rangle \\ \{I_2\}, & if\ I_1 is\ \langle nil, nil \rangle \\ \{I_1, I_2\}, & otherwise \end{cases}$$

Note that if we permit both operands to be the empty convex interval because of case 2 of the above definition the result remains in the set of $NC\mathcal{I}^*$, if this case was allowed we would

have the result in the $NC\mathcal{I}$ set of non-convex intervals. This union is a function with signature $C\mathcal{I} \times C\mathcal{I} \rightarrow NC\mathcal{I}^*$.

The following algorithm executes the union of *Definition 18*:

Algorithm 1: ($\cup^{C\mathcal{I}}$). The algorithm for the convex interval union is quite straightforward. In the case of non-mergeable operands we enforce ordering in the union respecting the total ordering of the timeline. There are five cases:

Case 1: If convex interval I_1 is before I_2 then $I_U^{C\mathcal{I}} = \{ I_1, I_2 \}$.

Case 2: If convex interval I_2 is before I_1 then $I_U^{C\mathcal{I}} = \{ I_2, I_1 \}$.

Case 3: If convex interval I_1 is adjacent or intersecting I_2 then merge the intervals into I_m and return $I_U^{C\mathcal{I}} = \{ I_m \}$.

Case 4: If both of the intervals are empty then return $I_U^{C\mathcal{I}} = \{ \}$.

Case 5: If the one of the two is the empty interval then return the other.

In the case that one of the operands is the empty interval then the union contains only the other operand, if both operands are the empty interval then the union is a the empty non-convex interval.

The complexity of this algorithm is $\mathcal{O}(1)$ since all comparisons and the merge function are performed in $\mathcal{O}(1)$ and no recursion or iteration will be performed.

Theorem 4: Correctness of $\cup^{C\mathcal{I}}$. The union of convex intervals contains all timepoints that are included either in the first convex interval or the second one.

$$\forall t \in \mathcal{P}'^* \left(t \in I_1 \vee t \in I_2 \Leftrightarrow t \in I_U^{C\mathcal{I}} \right)$$

Proof: In the first case where we have a mergeable result it is proven by *Theorem 1*. In case any of the operands is the empty convex interval the empty non-convex interval is returned. In the last case it is straightforward since if $t \in I_1$ then it is contained in the I_1 member of the union. Equivalently if $t \in I_2$ then it is contained in the I_2 member of the union. And if $t \in I_U^{C\mathcal{I}}$ then it either is included in the I_1 member or the I_2 member of the union.

Corollary 1: The union of convex intervals is commutative and associative

Proof: For the cases 2, 3, 4 and 5 it is trivial. For case 1 it is proven by *Theorem 2*.

Corollary 2: The identity element for union of convex intervals is the empty convex interval $\langle nil, nil \rangle$.

Proof: For the case 2, 3, 4 and 5 it is trivial. For case 1 it is proven by *Theorem 3*.

3.3.2. Non-Convex Union

We will now provide an algorithm that performs the union of a convex I and a non-convex $N = \{I_1, \dots, I_i, \dots, I_n\}$ interval of n members: $|N| = n$. The union is a function with signature $C\mathcal{I} \times NC\mathcal{I}^* \rightarrow NC\mathcal{I}^*$.

Algorithm 2 ($\cup^{C\mathcal{I}-NC\mathcal{I}}$: Convex over non-convex union). If I is the empty interval then $N_U = N$ else an iteration (or recursion) will be needed over the members of the non-convex

interval. Inputs are the convex interval I and the non-convex interval $N = \{I_1, \dots, I_i, \dots, I_n\}$ and output is $N_\cup$. We have three cases:

Case 1: if I is before a member I_i of the N, we insert I_i in the output $N_\cup$ and then we continue with the next member N .

Case 2: if I is after a member I_i of the N, we insert I_i in the output $N_\cup$ and then we continue with the next member N .

Case 3: if I and I_i are mergeable a merge should take place that results I_m. We then continue without insertion in the $N_\cup$ on the next member I_{i+1} of N and in the place of I we will use the merged I_m convex interval.

Finally after the iteration over all members of N we will insert in the result, the interval I, which inserts either a merged version of I and intervals of N or the initial interval I. In the case of the empty interval I the

Lemma 1: ($\cup^{\mathcal{CJ}-\mathcal{NCJ}}$ Correctness).

$$\forall t \in \mathcal{P}'^{*}\left(t \in I \ \vee t \in N \ \Leftrightarrow t \in N_U^{\mathcal{CJ}-\mathcal{NCJ}}\right)$$

Proof: We will examine each case separately:

Case 1: The convex interval I is empty, trivial.

Case 2: The convex interval I is not merging with the non-convex member I_i ; the non-convex member contains timepoints that are in the union and by construction is not merging with other members. Both I and I_i are members of the union. The insertion of I_i satisfies correctness. The convex interval I might still merge with a next member so we do not insert it but continue. In case it merges see case 3, otherwise it is inserted as is at the termination of the iteration.

Case 3: In this case I and the non-convex member I_i contain common timepoints and are merged; by *Theorem 1* the merged version contains all timepoints of both the convex interval and the member the I_i is retracted from the union to remove the redundancy of timepoints. The merged convex interval might still merge with next members so we continue the iteration. In case it merges we repeat this case with the next member, otherwise it is inserted as is at the termination of the iteration.

Lemma 2: $\cup^{\mathcal{CJ}-\mathcal{NCJ}}$ terminates.

Proof: The iteration over the members of the non-convex interval terminates trivially for all cases since the iteration is over the finite members of the non-convex interval.

The time complexity of the algorithm is dependable on the insert operation which is dependable on the data structure that will be used for representing the non-convex interval stamp. The algorithm is of $\mathcal{O}(|N|)$ assuming an insertion cost of $\mathcal{O}(1)$.

Definition 19: We define the union of non-convex intervals N_1 and N_2 , $N_\cup^{\mathcal{NCJ}} = N_1 \cup^{\mathcal{NCJ}} N_2$ as the non-convex interval with members the merged convex interval I_m (see *Definition 8*) if there are mergeable convex members in both intervals and the non-mergeable convex members of both intervals. This union is a function with signature $\mathcal{NCJ}^* \times \mathcal{NCJ}^* \to \mathcal{NCJ}^*$.

Algorithm 3: ($\cup^{NC\mathcal{I}}$) We assume as input the non-convex intervals $N_1 = \{I_{11,} \dots, I_{1i}, \dots, I_{1n}\}$ with count $|N_1| = n$ and $N_2 = \{I_{21,} \dots, I_{2j}, \dots, I_{2m}\}$ with count $|N_2| = m$. Output will be the non-convex interval $N_\cup$. Two iterations must be performed for each member of the one interval over each member of the other. For each member of the first interval I_{1i} the *The union* is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \to NC\mathcal{I}^*$.

Algorithm 2 will be used over N_2. The output of the *The union* is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \to NC\mathcal{I}^*$.

Algorithm 2 will be used with I_{1i+1} on the next iteration. The algorithm terminates when every interval of N_1 is used against intervals of N_2.

1. Initialize $N_\cup^{NC\mathcal{I}} = \{\}$
2. Repeat:
   a. $N_\cup^{NC\mathcal{I}} = I_{1i} \cup^{\mathcal{CI}-NC\mathcal{I}} N_\cup^{NC\mathcal{I}}$ where $I_{1i} \in N_1 or\ I_{1i} = \langle nil, nil \rangle\ if\ \ N_1 = \{\}$

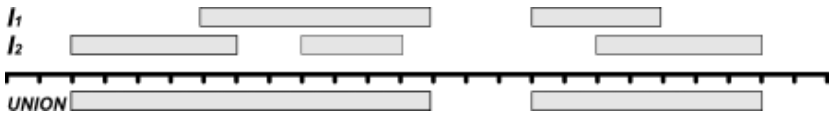


Figure 4 Union of Non-Convex Intervals

Theorem 5: Correctness of $\cup^{NC\mathcal{I}}$ .The union of non-convex intervals contains all timepoints that are included either in the first interval or the second one.

$$\forall t \in \mathcal{P}'^* \left(t \in N_1 \vee t \in N_2 \Leftrightarrow t \in N_\cup^{NC\mathcal{I}} \right)$$

Proof: Each result of the execution of *The union* is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \to NC\mathcal{I}^*$.

Algorithm 2 is correct by Lemma 1 and contains all members of the second non-convex interval as they are unified with the member of the first non-convex interval. By the commutative property of the union and by *Lemma 1* the unification of every following member is also correct.

Theorem 6: Termination of $\cup^{NC\mathcal{I}}$.

Proof: The outer iteration is over the finite set of the first non-convex members, and executes The union is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \to NC\mathcal{I}^*$.

Algorithm 2 that by Lemma 2 terminates.

The complexity of this algorithm is $\mathcal{O}(|N_1||N_2|)$. There have to be performed maximal $|N_2|$ iterations over the members of the second non-convex interval and on each iteration the The union is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \to NC\mathcal{I}^*$.

Algorithm 2 of complexity $\mathcal{O}(|N_1|)$ is performed; therefore the time complexity of the algorithm is $\mathcal{O}(|N_1||N_2|)$.

Corollary 3: The non-convex union is commutative and associative

Proof: Since the union of two convex intervals is associative and commutative by *Theorem 2* it holds also over sets of convex intervals.

Theorem 7: The identity element of the non-convex union is the empty non-convex interval.

Proof: Trivially by Lemma 1.

3.3.3. Convex Intersection

The intersection of convex intervals follows:

Definition 20. The intersection of two convex intervals I_1 and I_2, is a new convex interval $I_I^{\mathcal{CI}} = I_1 \cap^{\mathcal{CI}} I_2$ that contains all timepoints that are included in both intervals. The signature of this function is $\mathcal{CI} \times \mathcal{CI} \rightarrow \mathcal{CI}$:

$$I_I^{\mathcal{CI}} = I_1 \bigcap^{\mathcal{CI}} I_2 = intersection_{\mathcal{CI}}(I_1, I_2) = \langle I_s, I_e \rangle,$$

$$I_s = \begin{cases} max_p(I_{1_s}, I_{2_s}), & if\ I_1\ intersecting\ I_2 \\ nil, & otherwise \end{cases}$$

$$I_e = \begin{cases} min_p(I_{1_e}, I_{2_e}), & if\ I_1\ intersecting\ I_2 \\ nil, & otherwise \end{cases}$$

Theorem 8 . (Correctness of $\cap^{\mathcal{CI}}$) A timepoint $t \in \mathcal{P}$ is included in the intersection iff the timepoint is included in both operands: $\forall t \in \mathcal{P}, (t \in I_1 \wedge t \in I_2 \Leftrightarrow t \in I_I)$.

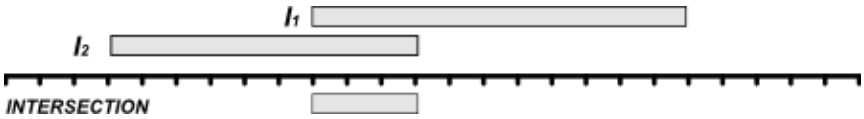


Figure 5 : Convex Interval Intersection

Proof:

Case (⇒).Let $t \in I_1$ and $t \in I_2$ then t is later or equal than the lowest timepoint of both intervals:

$$I_{1s} \leq t \wedge I_{2s} \leq t,$$

If $I_{1s} \leq I_{2s}$ then$I_{2s} \leq t$ and $max_p(I_{1_s}, I_{2_s}) = I_{2_s} = I_{Is}$ that entails and $I_{Is} \leq t$

If $I_{2s} \leq I_{1s}$ then$I_{1s} \leq t$ and $max_p(I_{1_s}, I_{2_s}) = I_{1_s} = I_{Is}$ that entails and $I_{Is} \leq t$

So in any case $I_{Is} \leq t$ (a)

Also $t \in I_1$ and $t \in I_2$ is earlier or equal than the highest timepoint of both intervals:

$I_{1e} \geq t \wedge I_{2e} \geq t,$

If $I_{1e} \leq I_{2e}$ then$t \leq I_{1e}$ and $min_p(I_{1_e}, I_{2_e}) = I_{1_e} = I_{Ie}$ that entails and $t \leq I_{Ie}$

If $I_{2e} \leq I_{1e}$ then$t \leq I_{2e}$ and $min_p(I_{1_e}, I_{2_e}) = I_{2_e} = I_{Ie}$ that entails and $t \leq I_{Ie}$

So in any case $t \leq I_{Ie}$ (b)

From $I_{Is} \leq t$ and $t \leq I_{Ie}$ we conclude that $t \in I_I$.

Case (⇐).Let $t \in I_I$ then $I_{Is} \leq t$ and $t \leq I_{Ie}$

By construction $I_{Ie} = min_p(I_{1_e}, I_{2_e})$ that entails $I_{1e} \geq t \wedge I_{2e} \geq t$, also $I_{Is} = max_p(I_{1_s}, I_{2_s})$ that entails $I_{1s} \leq t \wedge I_{2s} \leq t$, so we conclude $t \in I_1$ and $t \in I_2$.

We note that in the case the empty interval is one of the operands the intersection results an empty interval due to the use of min_p and max_p.

Theorem 9 . The intersection of two intervals is commutative and associative.

Proof: The first case is verified from the commutative and associative nature of the definitions of max_p and min_p. For the case of non-intersecting intervals it is straightforward.

Theorem 10 . The identity element of intersection is the total infinite interval $\langle -\infty, +\infty \rangle$.

Proof: In the case of non-empty intervals, let us assume $\mathrm{I} = \langle I_s, I_e \rangle, I \in \mathcal{CI}^*$ then according to the definition of the precedence operator (see Table 5) $I_{Is}^{\mathcal{CI}} = max_p(-\infty, I_s\ ) = I_s$ and $I_{Ie}^{\mathcal{CI}} = min_p(I_e, +\infty) = I_e$ which entails that $I_{Is}^{\mathcal{CI}} = I$. For the case of the empty interval the use of max_p and min_p will return the empty interval $\langle nil, nil \rangle$.

The complexity of convex interval intersection is $\mathcal{O}(1)$since all comparisons are of $\mathcal{O}(1)$, and no iteration is performed.

We continue and provide definitions and algorithms for the case of non-convex intervals.

3.3.4. Non-Convex Intersection

We first outline an algorithm that performs the intersection of a convex and a non-convex interval as a function with signature $\mathcal{CI} \times N\mathcal{CI}^* \rightarrow N\mathcal{CI}^*$.

Algorithm 4 $\cap^{\mathcal{CI}-N\mathcal{CI}}$Intersecting a convex interval I and a non-convex interval $N = \{I_1, \dots, I_i, \dots, I_n\}$ will result a non-convex interval $N_\cap$. We initialize $N_\cap = \{\ \}$ and an iteration (or recursion) will be needed over the members of the non-convex interval. On each iteration we test Iagainst the current member I_i of N. We have on each iteration three cases:

Case 1: I or N are the empty intervals then return the empty non convex interval.

Case 2: I and I_i are intersecting. We execute the intersection as described in *Definition 20* and will insert the resulting convex interval in the output $N_\cap$ since it contains all common timepoints. We continue with the convex intersection of the next member I_{i+1} of $N$and the convex interval I.

Case 3: I and I_i are not intersecting. We continue with the next member I_{i+1} of N

After the iteration is terminated we will either have the empty non-convex $N_\cap = \{\ \}$ interval if I does not intersect with any member of N, or $N_\cap$ will contain all convex intervals that are the intersecting regions of I and each member of N.

Lemma 3: (Correctness of $\cap^{\mathcal{CI}-N\mathcal{CI}}$). $\forall t \in \mathcal{P}, (\ t \in I \ \ \wedge t \in N \ \ \Leftrightarrow t \in N_\cap)$

Proof:

Case 1: Trivial since the empty convex interval does not intersect with any interval and the empty non-convex interval does not contain convex intervals that would intersect with I .

Case 2: If the non-convex interval contains one element the Trivially by *Theorem 8* for the correctness of the convex interval intersection.

Case 3: If the non-convex interval contains more elements they are by construction non-intersecting. The intersection with the first element will provide timepoints in common of I and I_i those are members of N and are not included in any other member of N . Since other members of N have also common timepoints with I the

repetition over those members ensures their inclusion in the intersection or their elimination if they are not common with I.

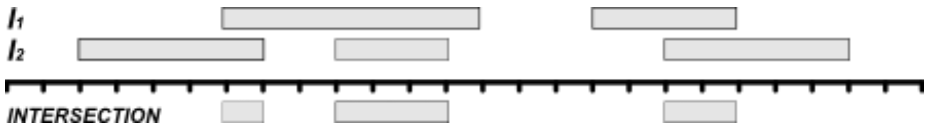


Figure 6: Intersection of Non Convex Intervals

Lemma 4: ($\cap^{\mathcal{CJ}-\mathcal{NCJ}}$ termination) The convex to non-convex intersection terminates.

Proof: Each convex intersection terminates, the iteration is performed over the finite set of a non-convex interval so it is straightforward that it terminates.

The algorithm is of $\mathcal{O}(|N|)$ because of the iteration over the $|N|$ members of the non-convex interval and since on each iteration all checks are of $\mathcal{O}(1)$.

We define the intersection of non-convex intervals as:

Definition 21 The intersection of non-convex intervals N_1 and N_2 , denoted as $N_1 \bigcap^{\mathcal{NCJ}} N_2$ is the non-convex interval with members the intersections of each convex interval of N_1 over each convex interval of N_2. This intersection is a function with signature $\mathcal{NCJ}^* \times \mathcal{NCJ}^* \rightarrow \mathcal{NCJ}^*$.

$$\bigcup_{I \in N_1}^{\mathcal{NCJ}} \left(I \bigcap^{\mathcal{CJ}-\mathcal{NCJ}} N_2 \right)$$

Algorithm 5 ($\cap^{\mathcal{NCJ}}$) Intersecting a non-convex interval $N_1 = \{I_{11,} \dots, I_{1i}, \dots, I_{1n}\}$ with count $|N_1| = n$ with a non-convex interval $N_2 = \{I_{21,} \dots, I_{2j}, \dots, I_{2m}\}$ with count $|N_2| = m$ will result a non-convex interval $N_\cap$. A nested iteration (or recursion) will be needed over the members of the non-convex intervals. The outer iteration selects each member of N_1 while the inner iteration executes *Algorithm 4*, for the selected member of $N_1$and the interval N_2.

1. Initialize $N_\cap = \{\ \ \}$
2. Repeat for each member I_i of N_1 and N_2
    a. Execute *Algorithm 4* providing $N_{\cap i}$
    b. Unify $N_{\cap i}$ and $N_\cap$ to the new $N_\cap$.

After the iteration is terminated we will either have the empty non-convex interval if not any member of N_1 does intersect with any member of N_2, or $N_\cap$ will contain all convex intervals that are the intersecting regions of members of interval N_1 and members of interval N_2.

Theorem 11 . Correctness of $\cap^{\mathcal{NCJ}}$.The intersection of non-convex intervals contains all timepoints that are included either in the first interval or the second one.

$$\forall t \in \mathcal{P}'^*, (\, t \in N_1 \wedge t \in N_2 \Leftrightarrow t \in N_\cap).$$

Proof :

Case 1: If the first non-convex N_1has a single member then trivially by *Lemma 3*.

Case 2: The first non-convex has more members. Let I_i be a member of the first non-convex interval; by *Lemma 3* the intersection will include all common members of I_i as part

of N_1 and N_2 . Timepoints of N_1 are included disjointly in all of its convex members; so the union of intersections of all members of N_1 with N_2 will provide all common timepoints.

Theorem 12: Termination of $\cap^{NC\mathcal{I}}$.

Proof: The repetition over the finite set of members of the N_1terminates since by *Lemma 4* each convex to non-convex intersection terminates and by *Theorem 6* the unification terminates.

Theorem 13: The intersection of two non-convex intervals is commutative and associative.

Proof: The intersection comprises of intersections of each member of the one interval with each member of the second; therefore since by *Theorem 9* convex intersection is commutative and associative the order of iteration still will provide each by each convex intersection that does not change the result.

Theorem 14: The identity element of intersection is the non-convex interval with member the total infinite convex interval $\{\langle -\infty, +\infty\rangle\}$.

Proof: Trivial by Theorem 10 and Lemma 3.

The complexity of this algorithm is $\mathcal{O}(|N_1||N_2|)$. This result stems from the fact that there have to be performed maximal $|N_2|$ iterations over the members of the second non-convex interval and on each iteration the *Algorithm 4* of complexity $\mathcal{O}(|N_1|)$ is performed; therefore the time complexity of the algorithm is $\mathcal{O}(|N_1||N_2|)$.

3.3.5. Convex Complement

In the need to describe time that a fact is not true, we might need the complement of an interval.

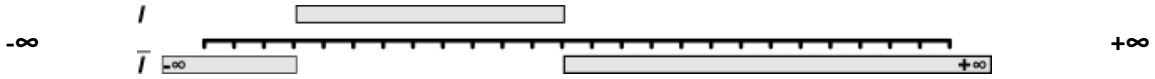


Figure 7 : Convex Interval Complement

Definition 22 The complement $\bar{I}$ of a convex interval I is the non-convex interval with members the convex interval of timepoints before the start of I and the convex interval of timepoints after the end of I:

$$\bar{I} = \begin{cases} \{\langle nil, nil\rangle\}, & if\ I = \langle -\infty, +\infty\rangle \\ \{\langle -\infty, +\infty\rangle\}, & if\ I = \langle nil, nil\rangle \\ \{\langle \text{next}(I_e), +\infty\rangle\}, & if\ I = \langle -\infty, I_e\rangle \\ \{\langle -\infty, previous(I_s)\rangle\}, & if\ I = \langle I_s, +\infty\rangle \\ \{\langle -\infty, previous(I_s)\rangle, \langle \text{next}(I_e), +\infty\rangle\} & otherwise \end{cases}$$

This convex complement is a function with signature $\mathcal{CI} \rightarrow NC\mathcal{I}$.

Theorem 15: (Correctness of $\bar{I}$).

$$\forall t \in \mathcal{P}'^{*}, (\, t \in I \Rightarrow t \notin \bar{I} \wedge t \notin I \Rightarrow t \in \bar{I}).$$

Proof: It is trivial by definition.

The complexity of convex interval complement is $\mathcal{O}(1)$since all comparisons are of $\mathcal{O}(1)$, and no iteration is performed.

3.3.6. Non-Convex Complement

Definition 23 The complement $\overline{N}$ of a non-convex interval N is the non-convex interval with members the convex intervals of timepoints that are not in any members of N.

$$\overline{N} = \bigcap_{I \in N}^{\mathcal{CJ}} \bar{I}$$

Note that in the case of the empty interval, the complement is the one containing the total infinite convex interval.

Figure 8 Complement of a Non-Convex Interval

Algorithm 6 ($\overline{N}$). The complement of a non-convex interval $N = \{I_1, \dots, I_i, \dots, I_n\}$ with count $|N| = n$. We initialize the complement with the convex complement of the first member. The complement of the first member:

Case 1: is the empty element the non-convex interval had the total infinite convex interval as member and by construction it does not contain any more members otherwise they would be merged.

Case 2: is the total infinite interval. The non-convex interval is the empty.

Case 3: is the left infinite interval. The non-convex interval contains has as member the right infinite convex interval.

Case 4: is the right infinite interval. The non-convex interval contains has as member the left infinite convex interval.

Case 5: has two convex intervals.

For cases 1 and 2 we terminate.

For cases 3, 4 and 5 we calculate the complement of the next member and the new complement will contain the intersection of the complements; since the complement of a convex interval contains other members and only common timepoints of the complements are qualified. We repeat for every remaining member.

Theorem 16: (Correctness of $\overline{\mathrm{N}}$).

$$\forall t \in \mathcal{P}'^{*}, (t \in \mathrm{N} \Rightarrow t \notin \overline{\mathrm{N}} \wedge t \notin \mathrm{N} \Rightarrow t \in \overline{\mathrm{N}}).$$

Proof:

If N has one member it is proven by Theorem 15.

1. If N has many members they are disjoint. Let I_1 and I_2 be two of the members their complements will contain each others timepoints. If we intersect their complements of I_1 with the complement $\overline{I_2}$, the timepoints of I_1 in $\overline{I_2}$ will be excluded by the

intersection. The timepoints of I_2 in $\overline{I_1}$ will be excluded since they do not appear in $\overline{I_2}$. So the aggregate complement will contain all timepoints that are neither in the first nor in the second interval.

2. Assume this holds for the complement of the n first intervals. The n+1 interval will have a complement that does not contain its timepoints, so intersecting it with the aggregate complement $\overline{I_{1,..,n}}$ will exclude its timepoint members from the new aggregate complement.

The time complexity of this algorithm is $\mathcal{O}(|N|^2)$[15]. This result comes from the fact that on each member of the convex interval the complement is computed and it might maximally produce two new convex intervals against which the next intersection of all other complements is applied.

3.3.7. Convex Difference

Many times we might want to find out the period of time for a fact except a certain time period that has to be excluded, in such cases we want to subtract a region of time, therefore we define difference of intervals. We will start with the case of convex intervals and then in the non-convex case.

Definition 24 The difference of two convex intervals I_1 and I_2, denoted as $I_1 -^{\mathcal{CJ}} I_2$ is a non-convex interval $N_-^{\mathcal{CJ}}$, that contains convex intervals with the time regions that are included in the first operand I_1 but not in the second operand I_2 :

$$I_1 -^{\mathcal{CJ}} I_2 = \begin{cases} \{\ \}, & if\ I_1 = \langle nil, nil\rangle \\ \{\ \}, & if\ I_{2s} \le I_{1s} \wedge I_{1e} \le I_{2e} \\ \{I_1\}, & if\ I_2 = \langle nil, nil\rangle \\ \{I_1\}, & if\ I_{1e} \le I_{2s} \ \vee\ I_{2e} \le I_{1s} \\ \{\langle I_{1s}, previous(I_{2s})\rangle\}, & if\ I_{1s} <\ I_{2s}\ \wedge\ I_{1e} \le\ I_{2e} \\ \{\langle next(I_{2e}), I_{1e}\rangle\}, & if\ I_{2s} \le I_{1s}\ \wedge\ I_{2e} <\ I_{1e} \\ \{\langle I_{1s}, previous(I_{2s})\rangle, \langle next(I_{2e}), I_{1e}\rangle\}, & if\ I_{1s} < I_{2s}\ \wedge\ I_{2e} <\ I_{2e} \end{cases}$$

Or in terms of Allen's interval algebra:

$$I_1 -^{\mathcal{CJ}} I_2 = \begin{cases} \{\ \}, & if\ I_1 = \langle nil, nil\rangle \vee I_1 equals I_2 \vee I_1 starts I_2 \vee I_1 during I_2 \vee I_1 finishes\ I_2 \\ \{I_1\}, & if I_2 = \langle nil, nil\rangle\ \vee\ I_1\ before\ I_2\ \vee\ I_1\ after\ I_2 \\ \{\langle I_{1s}, previous(I_{2s})\rangle\}, & if\ I_1 overlaps\ I_2\ \vee\ I_1 meets\ I_2 \vee\ I_1\ finished\_by\ I_2 \\ \{\langle next(I_{2e}), I_{1e}\rangle\}, & if\ I_1 met\_by\ I_2\ \vee\ I_1 overlaped\_by\ I_2 \vee\ I_1 started\_by\ I_2 \\ \{\langle I_{1s}, previous(I_{2s})\rangle, \langle next(I_{2e}), I_{1e}\rangle\}, & if\ I_1 includes\ I_2 \end{cases}$$

Theorem 17: The difference of convex intervals contains all timepoints that are included in the first operand but not in the second.

$$\forall t \in\ \mathcal{P}\left(\ t \in I_1 \wedge t \notin I_2 \Leftrightarrow\ t \in\ N_-^{\mathcal{CJ}}\right)$$

Proof : We examine each case separately:

Case 1: Trivial since if the first operand is empty there are no timepoint in the difference
Case 2: The first operand is totally contained in the second there are no timepoint that are contained in the first but not in the second operand.

[15] $O(2N^2 - 2N)$

Case 3: Trivial.

Case 4: The first operand is either after or before the second operand, so there are no timepoints in common.

Case 5: Some timepoints of the first operand are before the timepoints of the second. The common timepoints begin with the lowest timepoint of the second operand. The region of non-common timepoints spans from the beginning of the first operand to the previous timepoint of the lower of the second operand.

Case 6: Some timepoints of the first operand are after the timepoints of the second. The common timepoints begin with the next timepoint of the highest of the second operand and spans to the highest timepoint of the first operand.

Case 7: All timepoints of the second operand are included in the first. There are non-common timepoint before and after the endpoints of the second operand, and are calculated as in cases 5 and 6.

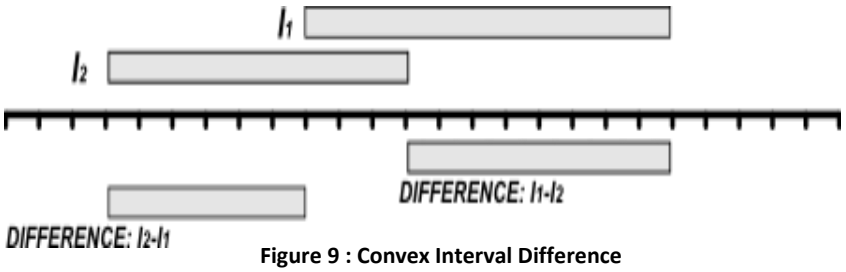


Figure 9 : Convex Interval Difference

Note that difference is neither commutative nor associative. The complexity of convex interval difference is $\mathcal{O}(1)$since all comparisons are of $\mathcal{O}(1)$, and no iteration is performed.

3.3.8. Non-Convex Difference

We continue with the non-convex case providing first algorithms for the difference of convex by non-convex and non-convex by convex intervals.

Algorithm 7 ($-^{NCJ-CJ}$) The difference of a non-convex interval N by a convex I , is the non-convex interval $N_{-}^{N-C} = N - I$ with members the difference of each member I_i of the non-convex interval by I . Iteration over the members of N will be needed.

1. Initialize $N_{-}^{N-C} = \{\ \}$.
2. Repeat over members I_i of N .
   a. Case 1: I_i intersects I , then we use the difference $N_{-} = I_i -^{CJ} I$ as described in *Definition 24* the members of the resulting non-convex interval if any; are copied in the output N_{-}^{CJ} and we continue with I_{i+1}.
   b. Case 2: I_i and I are disjoint, then we continue with I_{i+1}.

Lemma 5:(Correctness of $-^{NCJ-CJ}$).

$$\forall t \in \mathcal{P}\left(t \in \mathrm{N} \wedge t \notin I \ \Leftrightarrow t \in N_{-}^{N-C}\right)$$

Proof: If the NCI has a single member or is empty it is straightforward by convex difference and Theorem 17. If more members are in NCI they are disjoint by construction. In the case the convex member I_i and I are disjoint; then all timepoints of I_i are included in the result. In case I_i and I the convex difference $N_{-} = I_i -^{CJ} I$ will contain all members of I_i that are not in I . After the termination of the iteration in the result will be all timepoints of N (since they are members of the convex members of N) that are not in I . The reverse is trivial.

Lemma 6:(Termination of $-^{NCJ-CJ}$).

Proof: In the first case since the convex difference executed will produce maximum two members the insertion will also terminate. Both cases of the iteration terminate; therefore the iteration over the finite set of N will terminate.

The complexity of this algorithm is $\mathcal{O}(|N_2|)$ since computing the difference of convex intervals is $\mathcal{O}(1)$, and there is an iteration that performs the difference over the $|N_2|$ members of the non-convex interval.

Definition 25 The difference of non-convex intervals N_1 and N_2 , denoted as $N_1 -^{NCJ} N_2$ is

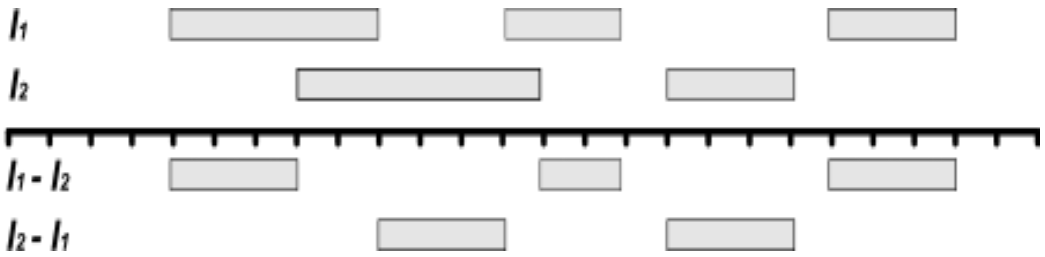


Figure 10: Difference of two Non-Convex Intervals

the non-convex interval with members the difference of each member of the first operand by all members of the second operand.

Algorithm 8:($-^{NCJ}$) Subtracting a non-convex interval $N_1 = \{I_{11,} \dots, I_{1i}, \dots, I_{1n}\}$ with count $|N_1| = n$ by a non-convex interval $N_2 = \{I_{21,} \dots, I_{2j}, \dots, I_{2m}\}$ with count $|N_2| = m$ will result a non-convex interval $N_{\_}$.

1. Initialize $N_{\_} = \{\ \}$
2. Repeat over members I_i of N_1:
    a. N_T=$\{ I_i\}$
    b. Use *Algorithm 7* for $N_{i-j} = N_T -^{NCJ-CJ} N_2$
    c. Insert all possible N_{i-j} in $N_{\_}$.

Theorem 18: (Correctness of $-^{NCJ}$). The difference of non-convex intervals contains all timepoints that are included in the first operand but not in the second.

$$\forall t \in \mathcal{P}\left(t \in N_1 \wedge t \notin N_2 \Leftrightarrow t \in N_{-}^{CJ}\right)$$

Proof: The second step of the inner loop calculates all timepoints of I_i (that are part of N_1) and not included in any member of N_2 by *Lemma 5* is correct. The repetition over the convex members of N_1 are satisfying that all timepoints of N_1 are taken in account.

Theorem 19: (Termination of $-^{NCJ}$).

Proof: By *Lemma 6* the second step of the loop terminates. The insertion of each result also terminates, and since the iteration is over the finite set of convex members of N_1 the algorithm terminates.

The complexity of this algorithm is $\mathcal{O}(|N_1||N_2|)$. This complexity is derived from the algorithm since there is an iteration over the$|N_1|$ members of the first non-convex interval and in each iteration the difference of all the $|N_2|$ members of the second non-convex interval is performed and the insertion in the resulting non-convex interval.

3.4. Discussion

All operations described for the convex and non-convex case can be used in the context of relational temporal databases.

In the case we want to insert a tuple in the database we have two cases, first if there are not existing value equivalent tuples we insert the tuple in the relation. If at least a value equivalent tuple exists in the database then the union operation over the existing temporal elements (intervals) convex or non-convex is executed. If multiple equivalent tuples exist then we construct a non-convex interval of the existing tuples and apply union with the intended for insertion temporal interval. The extracted interval will be inserted in the database and the previously existing will be logically deleted. In a bitemporal[16] relation only previously non-existing time periods will be inserted with a new transaction time start or merged intervals while previously existing intervals that were merged with new ones will be logically deleted.

Patient	Diagnosis	Valid
John	Hypertension S2	$\langle 7{:}00, 14{:}00\rangle$
John	Hypertension S2	$\langle 18{:}00, 20{:}00\rangle$

(a) The snapshot of the relation with before the insertion

Patient	Diagnosis	Valid
John	Hypertension S2	$\langle 7{:}00, 15{:}00\rangle$
John	Hypertension S2	$\langle 18{:}00, 20{:}00\rangle$
John	Breathing problems	$\langle 19{:}15, 23{:}30\rangle$

(b) The patient relation after the insertion

Figure 11 A tuple insertion

For example suppose we want to insert in a relation of Figure 14 (a) the tuples:

('John', 'Breathing problems', ' $\langle 22{:}00, 23{:}30\rangle$ ')
('John', 'Hypertension S2', ' $\langle 13{:}00, 15{:}00\rangle$ ')

The first insertion since there is no value equivalent tuple will be inserted in the database, but the second tuple will be merged with an existing tuple in the relation see Figure 14(b).

If we want to delete a tuples from the database given a temporal interval of the deletion the difference operation is applied over the temporal elements of valid time on tuples in the databases that qualify for deletion according to the conditions specified. The result of this difference could yield the empty interval, then the associated tuple is logically deleted, else tuples that have a remaining non-empty interval will be logically deleted and reinserted with temporal elements the non-empty intervals that were produced by the difference.

For updates there are cases we will perform deletions and insertions. Suppose we want to update the diagnosis of John for the times 16:00 to 18:45 to hypertension stage 1, then the

[16] The example following does not include the transaction time, though semantics of logical deletion with the respected changes on transaction times should take place as described.

second tuple in Figure 14(a) should be changed with an updated time of $\langle 18:46, 20:00\rangle$ while an insertion of a tuple ('John', 'Hypertension S1', '$\langle 16:00, 18:45\rangle$ ') should be performed. In this case deletion semantics and insertion semantics were applied. An update to the temporal elements can also be performed.

For a selection that wants to retrieve data for example about the times that John had hypertension problems and breathing problems the intersection of intervals could be performed.

Chapter 4

Determinate and Indeterminate Intervals

Up to this point the time model and its representation have been investigated. Time is related to facts that represent events or situations and obeys rules and with it, the correlated data. We are able to state the time (instant, period or interval) that a certain fact was, is or will be true. If we are certain about the time of validity of a fact, we call this time determinate, but for a variety of reasons as discussed earlier in chapter 2; often we have a vague knowledge about the time related to a fact and willing to record or refer to this knowledge; indeterminacy about time arises. We will now examine this aspect of time.

4.1. Two sorted intervals

For indeterminate time there is no needs to define a new time model, other than that we already have presented in the foregoing sections. Indeterminate time has explicitly different semantics, but still is not different in comparison to determinate. The time domain consists of timepoint over P' and indeterminacy refers not to the existence of a timepoint but its evaluation in relation to a fact. Relations and operations over solely indeterminate time remains the same as does for determinate time the evaluation will remain indeterminate. So there is no need to redefine intersection, mergence or adjacency in the presence of the same evaluation. The functionality though changes when we have to relate a determinate and an indeterminate interval where the evaluation over the same timepoint is different. In the need of operationally accommodating both characters of evaluating time, we will investigate the integration choices in terms of functions among the two sorts. In the following we examine the union, intersection, difference and complement.

4.1.1. Representations

For the convex case any interval would need for its representation an additional field of its type in order to distinguish the two sorts and the representation of a sorted interval would be of the form $\mathrm{I}^s = \{\langle \mathrm{t_s}, \mathrm{t_e} \rangle, sort\}$, where sort is either determinate or indeterminate, we refer to the set of sorted convex intervals as $\mathcal{SCI}$. In the following a determinate interval $\{\langle \mathrm{t_s}, \mathrm{t_e} \rangle, determinate\}$ will be referred for simplicity as I^d while an indeterminate $\{\langle \mathrm{t_s}, \mathrm{t_e} \rangle, indeterminate\}$ will be referred as I^d. The set of determinate intervals $\mathcal{DCI}$ is a proper subset of the sorted set $\mathcal{DCI} \subset \mathcal{SCI}$ and the set of indeterminate intervals $\mathcal{ICI}$ is a proper subset of the sorted set $\mathcal{ICI} \subset \mathcal{SCI}$.

The non-convex interval that comprises with members of the set $\mathcal{SCI}$ is a set of non-merging sorted convex intervals and will be referred as $\mathcal{SNCI}$. A representation of a non-

convex interval with n members of the set $SNCI$ would be $N^S = \{I_i^S, i \in [1,n] \wedge I_i^S \in SCI\}$ where the members do not merge. A fully determinate non-convex interval is notated as $N^d = \{I_i^d, i \in [1,n] \wedge I_i^d \in DCI\}$ and a fully indeterminate is notated as $N^i = \{I_i^i, i \in [1,n] \wedge I_i^i \in ICI\}$. So in general a sorted non-convex interval is a union $N^S = N^d \cup N^i$ so that not any member of the determinate subset or the indeterminate subset is merging in terms of convex intervals of the set CI.

Another choice of representation for the non-convex case is construct a non-convex sets of convex intervals in the set CI, and an additional field sort that evaluates any members of the set: $N^d = \{\{I_i \ , i \in [1,n] \wedge I_i \in CI\}, sort\}$ with the sort field to refer to determinate or indeterminate. The set $DNCI$ with non-merging convex interval members of the set CI that all evaluate to determinate would have members of the form: $N^d = \{\{I_i \ , i \in [1,n] \wedge I_i \in CI\}, determinate\}$. The set $INCI$ with non-merging convex interval members of the set CI that all evaluate to indeterminate would have members of the form: $N^d = \{\{I_i \ , i \in [1,n] \wedge I_i \in CI\}, indeterminate\}$. In the general case of both determinate and indeterminate a sorted non-convex interval would have two non-convex members $N^S = \{N^d, N^i\}$ such that not any member of the determinate or the indeterminate non-convex members merge, but a sorted interval is still a union $N^S = N^d \cup N^i$ of determinate and indeterminate intervals even if they are kept separated. This set will be referred as $NCIS$

In the following we follow the representation scheme of $SNCI$. Modifications for the $NCIS$ set will be discussed explicitly when needed.

For the set $SNCI$ the functions that provide the earliest and latest convex interval member need modifications in order to

Definition 26. The functions with signature $SNCI \rightarrow DCI$ return respectively the earliest and the latest determinate convex member over the time line, of a non-convex sorted interval:

$$earliest_d(N) = \begin{cases} I_1 \in N^d \wedge \nexists I_i \in N^d, I_i \, before \, I_1 \\ \langle nil, nil \rangle, \quad if \; N^d = \{\ \} \end{cases}$$

$$latest_d(N) = \begin{cases} I_k \in N^d \wedge \nexists I_i \in N^d, I_i \, after \, I_k \\ \langle nil, nil \rangle, \quad if \; N^d = \{\ \} \end{cases}$$

Definition 27. The functions with signature $SNCI \rightarrow ICI$ return respectively the earliest and the latest indeterminate convex member over the time line, of a non-convex sorted interval:

$$earliest_i(N) = \begin{cases} I_1 \in N^i \wedge \nexists I_i \in N^i, I_i \, before \, I_1 \\ \langle nil, nil \rangle, \quad if \; N^i = \{\ \} \end{cases}$$

$$latest_i(N) = \begin{cases} I_k \in N^i \wedge \nexists I_i \in N^i, I_i \, after \, I_k \\ \langle nil, nil \rangle, \quad if \; N^i = \{\ \} \end{cases}$$

The functions of *Definition 14* without further modifications are operational in the case of intervals over the $NCIS$ set.

4.2. Union

In the presence of both determinate and indeterminate intervals it is necessary to investigate the semantics of unifying two convex intervals and hence two non-convex intervals. Indeterminate time expresses a belief or a possibility that a fact holds over that time period. So for the intersecting region of two intervals a determinate D and an indeterminate I we have two choices

1. The region will be evaluated as indeterminate accepting the minimum possibility of the two intervals
2. The region will be evaluated as determinate accepting the maximum possibility of the two regions

The later semantics prevails since we can think of union as the disjunction of evaluations $TRUE \vee POSSIBLY\ TRUE = TRUE$ of the corresponding timepoints. For two sorted convex intervals A and B, the result of the union as far as timepoints are concerned, will be as depicted in Table 7. In terms of possibilities the maximum possibility prevails. So we will assume that for the intersecting region of the two intervals $D \cup I = D$. Non intersecting regions will include in the result all determinate timepoints left in D and all indeterminate timepoints left in I.

A	B	$A \cup B$
$t \in A \ \wedge t\ is\ D$	$t \in B \ \wedge t\ is\ D$	$t \in A \cup B \ \wedge t\ is\ D$
$t \in A \ \wedge t\ is\ D$	$t \in B \ \wedge t\ is\ I$	$t \in A \cup B \ \wedge t\ is\ D$
$t \in A \ \wedge t\ is\ D$	$t \notin B$	$t \in A \cup B \ \wedge t\ is\ D$
$t \in A \ \wedge t\ is\ I$	$t \in B \ \wedge t\ is\ D$	$t \in A \cup B \ \wedge t\ is\ D$
$t \in A \ \wedge t\ is\ I$	$t \in B \ \wedge t\ is\ I$	$t \in A \cup B \wedge t\ is\ I$
$t \in A \ \wedge t\ is\ I$	$t \notin B$	$t \in A \cup B \wedge t\ is\ I$
$t \notin A$	$t \in B \ \wedge t\ is\ D$	$t \in A \cup B \wedge t\ is\ D$
$t \notin A$	$t \in B \ \wedge t\ is\ I$	$t \in A \cup B \wedge t\ is\ I$
$t \notin A$	$t \notin B$	$t \notin A \cup B$

Table 7 : Evaluation for union of two sorted intervals A and B.

Patient	Diagnosis	Type	Valid
John	Hypertension S2	Ind	$\langle 7{:}00, 14{:}00\rangle$
John	Hypertension S2	Det	$\langle 18{:}00, 20{:}00\rangle$
John	Breathing Probl.	Det	$\langle 19{:}00, 22{:}00\rangle$
Mary	Gripes	Det	$\langle 13{:}30, 14{:}10\rangle$
Mary	Anaphylactic Inc.	Det	$\langle 15{:}00, 15{:}40\rangle$
Mary	Anaphylactic Inc.	Det	$\langle 14{:}30, 14{:}59\rangle$

Figure 12 An example relation with sorted temporal data

The union of sorted intervals might be used to insert a new tuple in the database; in that case the temporal intervals of value equivalent tuples to the tuple intended for insertion have to be examined against the temporal intervals of the new interval. In case that the union provides coalesced intervals the old tuples containing the intervals that were coalesced should be deleted (logically) and a new tuple has to be inserted with the merged interval, regardless if the interval concerns indeterminate or determinate time. The union operation on intervals might also be used on a relational union of schema equivalent relations, or in case a projection that requests among others the temporal attributes and there are tuples with common values on non-temporal attributes for projection and different values on attributes that are not projected. For example in the case of Figure 12 a

projection requesting the patients name and times should return the relation in Figure 16, where tuples for John as tuples for Mary were coalesced.

4.2.1. Convex Union

Definition 28 The union of two convex intervals with different evaluation a determinate I_1^d

Patient	Type	Valid
John	Ind	$\langle 7{:}\,00, 14{:}\,00\rangle$
John	Det	$\langle 18{:}\,00, 22{:}\,00\rangle$
Mary	Det	$\langle 13{:}\,30, 14{:}\,10\rangle$
Mary	Det	$\langle 14{:}\,30, 15{:}\,40\rangle$

Figure 13 :A projection for Patient and Valid Time for relation of Figure 12

and an indeterminate I_1^i is a non-convex interval with two sorted members, such that I_1^d is a member of the union with the evaluation unattached to determinate, and all timepoints left in the indeterminate interval after the convex difference of the interval I_2 by I_1 while we evaluate it as indeterminate:

$$I_U^{TS} = I_1^d \bigcup^{SC\mathcal{J}} I_2^i = \{I_1^d, I^i\}, \qquad where\ I^i = \left(I_2 -^{C\mathcal{J}} I_1\right)^i$$

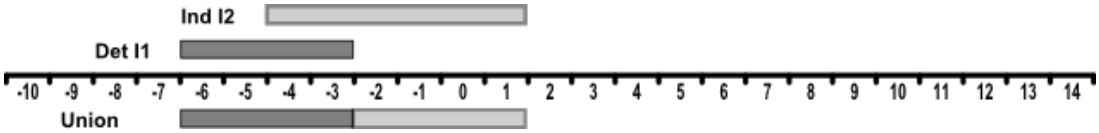


Figure 14 Union of an Indeterminate and Determinate Convex Interval

With a reference to the timepoints:

$\forall t, t \in I^d \Leftrightarrow t \in I_U^d \ \wedge (t \in I^i \wedge t \notin I^d) \Leftrightarrow t \in I_U^i$

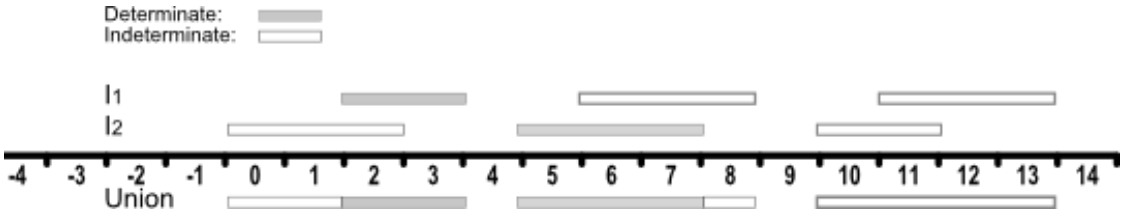


Figure 15 : Union of Non-Convex multi -Sorted Intervals

4.2.2. Non-Convex Union

Definition 29 The union of two non-convex intervals with different evaluation a determinate N_1^d and an indeterminate N_2^i is a non-convex interval with two sorted members, such that:

$$N_U^S = N_1^d \bigcup^{SNC\mathcal{J}} N_2^i = N_1^d \bigcup^{NC\mathcal{J}} N^i \qquad where\ N^i = \left(N_2 -^{NC\mathcal{J}} N_1\right)^i$$

Definition 30 The union of two, sorted non-convex intervals $N_1 = N_1^d \cup^{NC\mathcal{J}} N_1^i$ and $N_2 = N_2^d \cup^{NC\mathcal{J}} N_2^i$ is a non-convex interval with two sorted members, such that:

$$N_U^{TS} = N_1 \bigcup^{SNCJ} N_2 = N^d \bigcup^{NCJ} N^i$$

$$where\ N^i = \left(N_2^i -^{NCJ} N_1^d\right)^i \bigcup^{NCJ} \left(N_1^i -^{NCJ} N_2^d\right)^i$$

$$and\ N^d = N_1^d \bigcup^{NCJ} N_2^d$$

4.3. Intersection

The intersection contains only timepoints of the common time region, and in the presence of both determinate and indeterminate intervals the intersection will contain the same time region as common intersection. It is though important to investigate the evaluation of the resulting interval. The rubric of the intersecting region of two intervals a determinate D and an indeterminate I lies between the next two choices:

1. The region will be evaluated as indeterminate accepting the minimum possibility of the two intervals
2. The region will be evaluated as determinate accepting the maximum possibility of the two regions

The former semantics prevails since we can think of intersection as the conjunction of evaluations $TRUE \wedge POSSIBLY\ TRUE = POSSIBLY\ TRUE$ of the corresponding timepoints. For two sorted convex intervals A and B, the result of the union as far as timepoints are concerned, will be as provided in Table 8. In terms of possibilities the minimum possibility prevails. So we will assume that for the intersecting region of the two intervals $D \cap I = I$. Non intersecting regions will be empty as common intersection (see *Definition 20*).

A	B	$A \cap B$
$t \in A \wedge t\ is\ D$	$t \in B \wedge t\ is\ D$	$t \in A \cap B \wedge t\ is\ D$
$t \in A \wedge t\ is\ D$	$t \in B \wedge t\ is\ I$	$t \in A \cap B \wedge t\ is\ I$
$t \in A \wedge t\ is\ D$	$t \notin B$	$t \notin A \cap B$
$t \in A \wedge t\ is\ I$	$t \in B \wedge t\ is\ D$	$t \in A \cap B \wedge t\ is\ I$
$t \in A \wedge t\ is\ I$	$t \in B \wedge t\ is\ I$	$t \in A \cap B \wedge t\ is\ I$
$t \in A \wedge t\ is\ I$	$t \notin B$	$t \notin A \cap B$
$t \notin A$	$t \in B \wedge t\ is\ D$	$t \notin A \cap B$
$t \notin A$	$t \in B \wedge t\ is\ I$	$t \notin A \cap B$
$t \notin A$	$t \notin B$	$t \notin A \cap B$

Table 8 : Evaluation for intersection of two sorted intervals A and B.

4.3.1. Convex Intersection

Definition 31 The intersection of two convex intervals with different evaluation a determinate I_1^d and an indeterminate I_1^i is a convex interval with an indeterminate member I^i containing all common timepoints.

$$I_U^{TS} = I_1^d \bigcap^{TSCJ} I_2^i = I^i, \qquad where\ I^i = \left(I_1 \bigcap^{CJ} I_2\right)^i$$

4.3.2. Non-Convex Intersection

Definition 32 The intersection of two non-convex intervals regardless of their evaluation, determinate, indeterminate or two-sorted N_1^S and N_2^S is a non-convex interval with indeterminate members, such that:

$$N_{\cap}^{TS} = N_1^S \overset{TSNC\mathcal{I}}{\bigcap} N_2^S = \overset{NC\mathcal{I}}{\bigcup_{I_{i1} \in N_1^S}} \left(\mathrm{I}_{i1}^s \overset{TSC\mathcal{I}}{\bigcap_{I_{i2} \in N_2^S}} \mathrm{I}_{i2}^s \right)$$

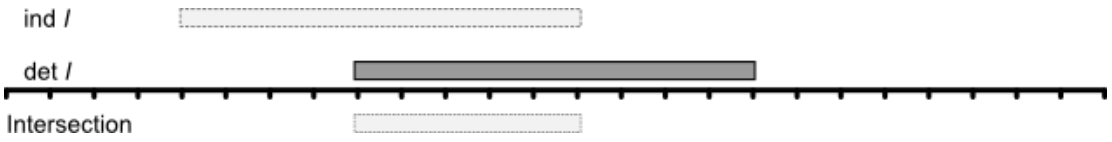


Figure 16 Intersection of an Indeterminate and a Determinate Convex Interval

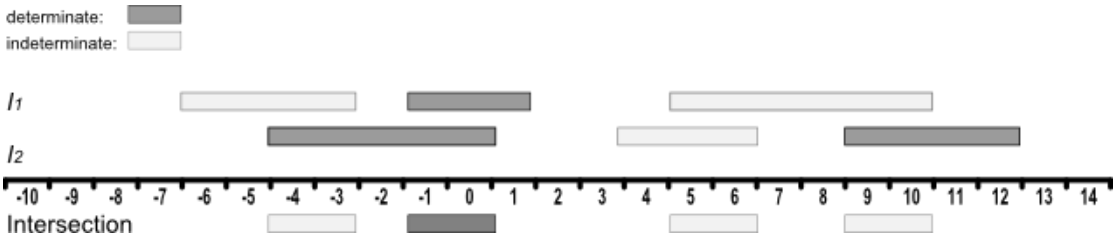


Figure 17 : Intersection of Two Sorted Non-Convex Intervals

4.4. Complement

Having a convex interval then the complement contains all points that are not included in that interval. For determinate intervals the semantics of difference do not change, but in the presence of indeterminate intervals the complement is slightly different. Any timepoint that is not included in an indeterminate interval is definitely included in its complement; therefore we evaluate it as determinate member of the complement. Timepoints that are in an indeterminate region are also possible members of the complement; therefore the whole indeterminate interval with unchanged evaluation as indeterminate is also contained in the complement. With reference to timepoints Table 9 clarifies complement inclusion.

A	$\bar{A}$
$t \in A \ \wedge t \ is \ D$	$t \notin \bar{A}$
$t \in A \ \wedge t \ is \ I$	$t \in \bar{A} \ \wedge t \ is \ I$
$t \notin A$	$t \in \bar{A} \wedge t \ is \ D$

Table 9 : Evaluation for the complement of a convex sorted interval

4.4.1. Convex Complement

Definition 33 The complement of a convex sorted interval I is a non-convex interval that includes the determinate intervals with timepoints not in I and if it is indeterminate the interval I itself.

$$\overline{I^s} = \begin{cases} (\bar{I})^d, & if I \text{ is determinate} \\ (\bar{I})^d \bigcup^{NCJ} I^i, & if I \text{ is indeterminate} \end{cases}$$

4.4.2. Non-Convex Complement

Definition 34 The complement of a non-convex sorted interval N is a non-convex interval that includes the determinate intervals with timepoints not in N and the indeterminate members of N .

$$\overline{N^s} = \bar{N}^d \bigcup^{NCJ} N^i$$

Where $\bar{N}^d$ is the unsorted complement as determinate and N^i are the indeterminate members of N .

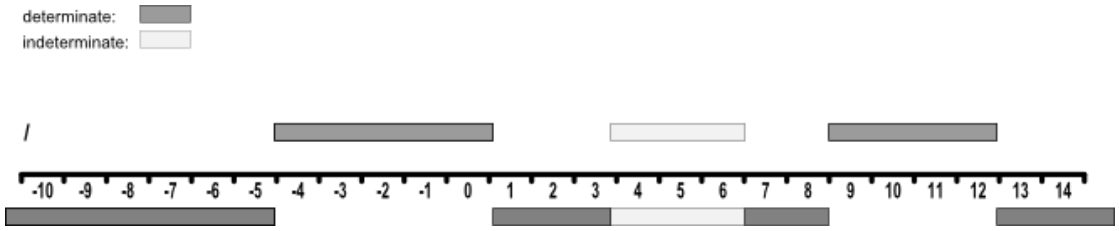


Figure 18 : The Complement of a Non-Convex Sorted Interval

4.5. Difference

The difference contains only timepoints of the first operand that do not appear in the second operand, so the possible common time region is excluded from the result. In the presence of both determinate and indeterminate intervals again the intersecting region evaluation has to be investigated. We will examine two cases due to the non-commutative property of difference.

1. Case 1: The case of difference of indeterminate by a determinate interval. In the common one sorted case every timepoint is excluded from the result since its evaluated as the conjunction of the first operand with the negation of the second. In our case $POSSIBLY\ TRUE \wedge FALSE = FALSE$. So the resulting time interval will not contain any of the timepoints of the intersecting period.
2. Case 1: The case of difference of determinate by an indeterminate interval. The evaluation again relies on the conjunction of the first operand with the negation of the second. But the negation of $POSSIBLY\ TRUE$ is $POSSIBLY\ TRUE$ since $POSSIBLY\ FALSE$ and $POSSIBLY\ TRUE$ express the same meaning. In our case $TRUE \wedge POSSIBLY\ TRUE = POSSIBLY\ TRUE$. So the resulting time interval will contain all the timepoints of the intersecting period but evaluated as indeterminate.

A summary of the semantics over the difference operation on sorted intervals is depicted on Table 10.

A	B	$A - B$
$t \in A \ \wedge t \ is \ D$	$t \in B \ \wedge t \ is \ D$	$t \notin A - B$
$t \in A \ \wedge t \ is \ D$	$t \in B \ \wedge t \ is \ I$	$t \in A - B \ \wedge t \ is \ I$
$t \in A \ \wedge t \ is \ D$	$t \notin B$	$t \in A - B \ \wedge t \ is \ D$
$t \in A \ \wedge t \ is \ I$	$t \in B \ \wedge t \ is \ D$	$t \notin A - B$
$t \in A \ \wedge t \ is \ I$	$t \in B \ \wedge t \ is \ I$	$t \notin A - B$
$t \in A \ \wedge t \ is \ I$	$t \notin B$	$t \in A - B \ \wedge t \ is \ I$
$t \notin A$	$t \in B \ \wedge t \ is \ D$	$t \notin A - B$
$t \notin A$	$t \in B \ \wedge t \ is \ I$	$t \notin A - B$
$t \notin A$	$t \notin B$	$t \notin A - B$

Table 10 : Evaluation for the difference of two sorted intervals A and B.

4.5.1. Convex Difference

Definition 35 The difference of two convex intervals with different evaluation a determinate I_1^d and an indeterminate I_2^i is

1. In case of $I_1^i - I_2^d$ the result is a non-convex interval with indeterminate members I^i containing timepoints of I_1^i but not I_2^d.
$$I_- = I_1^i -^S I_2^d = N^i, \qquad where \ N^i = \left(I_1 -^{\mathcal{CJ}} I_2\right)^i$$
2. In case of $I_1^d - I_2^i$ the result is a non-convex two-sorted interval with indeterminate members I^i containing timepoints of the intersecting region and determinate members I^d containing timepoints of I_1^d that are not in I_2^i.
$$I_- = I_1^i -^S I_2^d = N^{id}, \qquad where \ N^{id} = \left(I_1 -^{\mathcal{CJ}} I_2\right)^d \bigcup^{NC\mathcal{J}} \left(I_1 \cap^{\mathcal{CJ}} I_2\right)^i$$

4.5.2. Non-Convex Difference

Definition 36 The difference of two non-convex intervals regardless of their evaluation, determinate, indeterminate or two-sorted N_1^S and N_2^S is a two-sorted non-convex interval $N_-^{TS} = N^i \cup^{NC\mathcal{J}} N^d$ with determinate members, such that:

$$N_-^{TS} = N_1^S -^{SNC\mathcal{J}} N_2^S = N^i \bigcup^{NC\mathcal{J}} N^d$$

$$where \ N^d = \left(N_1^d -^{NC\mathcal{J}} N_2\right)^d$$

$$and \ N^i = \left(\left(N_1^i -^{NC\mathcal{J}} N_2\right) \bigcup^{NC\mathcal{J}} \left(N_1^d \bigcap^{NC\mathcal{J}} N_2^i \right) \right)^i$$

4.6. Metric Functions

Taking into account the case of sorted non-convex intervals we alter the functions that returns the volume of membership.

Definition 37 The volume of a non-convex interval in terms of the number of convex determinate members that are included that we shall call determinate count, is a function with signature $SNC\mathcal{J} \rightarrow \mathbb{N}$, and will be denoted as $|N|^d$.

$$|N| = count_d(N^S) = \sum_{I^d \in N} 1$$

Definition 38 The volume of a non-convex interval in terms of the number of convex indeterminate members that are included that we shall call determinate count, is a function with signature $SNC\mathcal{I} \rightarrow \mathbb{N}$, and will be denoted as $|N|^i$.

$$|N| = count_i(N^S) = \sum_{I^i \in N} 1$$

The total count will be obtained by the function already defined for the unsorted case (see *Definition 15*).

We extend also the function of the duration for the non-convex sorted interval as:

Definition 39 The determinate duration of a non-convex interval $N \in NC\mathcal{I}$, is a function with signature $SNC\mathcal{I} \rightarrow \mathbb{Z} \cup \{\infty\}$ that returns the aggregate duration of all its determinate constituent convex intervals:

$$duration_d(N^S) = \sum_{I^d \in N} duration_{ci}(I^d)$$

Definition 40 The duration of a non-convex interval$N \in NC\mathcal{I}$, is a function with signature $SNC\mathcal{I} \rightarrow \mathbb{Z} \cup \{\infty\}$ that returns the aggregate duration of all its indeterminate constituent convex intervals:

$$duration_i(N^S) = \sum_{I^i \in N} duration_{ci}(I^i)$$

The topological thick grained diameter may also be modified for each sort as follows: *Definition 41* The function with signature $SNC\mathcal{I} \rightarrow SC\mathcal{I}$, which returns the topological diameter of the region of time that contains all the determinate convex members of the sorted non-convex interval N; where N_s is the lowest endpoint of $earliest_d(N)$, while N_e is the highest endpoint of $latest_d(N)$.

$$\delta_d(N^S) = \begin{cases} \langle nil, nil \rangle, & if\ N^d = \{\ \} \\ \langle N_s, N_e \rangle\ , & otherwise \end{cases}$$

Definition 42 The function with signature $SNC\mathcal{I} \rightarrow SC\mathcal{I}$, which returns the topological diameter of the region of time that contains all the indeterminate convex members of the sorted non-convex interval N; where N_s is the lowest endpoint of$earliest_i(N)$, while N_e is the highest endpoint of $latest_i(N)$.

$$\delta_i(N^S) = \begin{cases} \langle nil, nil \rangle, & if\ N^i = \{\ \} \\ \langle N_s, N_e \rangle\ , & otherwise \end{cases}$$

4.7. Algebraic Properties

It is important to explore some properties of operations introduced in this algebra for indeterminate and determinate intervals. We will examine double negation, the duality principle for union and intersection and the distributivity of the operations for intersection over union and union over intersection.

Theorem 20 (Distributivity of union over intersection). For the sorted intervals A, B, C the following holds:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

Proof: We will examine if each part is a subset of the other for both determinate and indeterminate timepoint inclusion separately. We will first show that $A \cup (B \cap C)$ is a subset of $(A \cup B) \cap (A \cup C)$.

For indeterminate timepoints, if $t \in^{ind} A \cup (B \cap C)$ then we have three cases

Case 1: $t \in^{ind} A \wedge t \in^{ind} (B \cap C)$ that splits in two subcases:

1.1. $t \in^{ind} A \wedge t \in^{ind} B \wedge t \in C$ then $t \in^{ind} (A \cup B)$ and $t \in (A \cup C)$ which concludes that $t \in^{ind} (A \cup B) \cap (A \cup C)$.
1.2. $t \in^{ind} A \wedge t \in B \wedge t \in^{ind} C$ then $t \in (A \cup B)$ and $t \in^{ind} (A \cup C)$ which concludes that $t \in^{ind} (A \cup B) \cap (A \cup C)$.

Case 2: $t \in^{ind} A \wedge t \notin (B \cap C)$ we have three subcases:

2.1 $t \in^{ind} A \wedge t \in B \wedge t \notin C$ then $t \in (A \cup B)$ and $t \in^{ind} (A \cup C)$ which concludes that $t \in^{ind} (A \cup B) \cap (A \cup C)$
2.2 $t \in^{ind} A \wedge t \notin B \wedge t \in C$ then $t \in^{ind} (A \cup B)$ and $t \in (A \cup C)$ which concludes that $t \in^{ind} (A \cup B) \cap (A \cup C)$
2.3 $t \in^{ind} A \wedge t \notin B \wedge t \notin C$ then $t \in^{ind} (A \cup B)$ and $t \in^{ind} (A \cup C)$ which concludes that $t \in^{ind} (A \cup B) \cap (A \cup C)$

Case 3: $t \notin A \wedge t \in^{ind} (B \cap C)$ we have two subcases:

2.1 $\mathrm{t} \notin \mathrm{A} \wedge \mathrm{t} \in^{ind} \mathrm{B} \wedge \mathrm{t} \in \mathrm{C}$ then $\mathrm{t} \in^{\mathrm{ind}} (\mathrm{A} \cup \mathrm{B})$ and $\mathrm{t} \in (\mathrm{A} \cup \mathrm{C})$ which concludes that $\mathrm{t} \in^{\mathrm{ind}} (\mathrm{A} \cup \mathrm{B}) \cap (\mathrm{A} \cup \mathrm{C})$.
2.2 $\mathrm{t} \notin \mathrm{A} \wedge \mathrm{t} \in \mathrm{B} \wedge \mathrm{t} \in^{ind} \mathrm{C}$ then $\mathrm{t} \in (\mathrm{A} \cup \mathrm{B})$ and $\mathrm{t} \in^{\mathrm{ind}} (\mathrm{A} \cup \mathrm{C})$ which concludes that $\mathrm{t} \in^{\mathrm{ind}} (\mathrm{A} \cup \mathrm{B}) \cap (\mathrm{A} \cup \mathrm{C})$

We conclude that $\forall t \in^{ind} A \cup (B \cap C) \Rightarrow t \in^{ind} (A \cup B) \cap (A \cup C)$

For determinate timepoints, if $t \in^{det} A \cup (B \cap C)$ then we have two cases:

Case 1: $t \in^{det} A$ that infers $\mathrm{t} \in^{\mathrm{det}} (\mathrm{A} \cup \mathrm{B}) \wedge \mathrm{t} \in^{\mathrm{det}} (\mathrm{A} \cup \mathrm{C})$, which concludes that $\mathrm{t} \in^{\mathrm{det}} (\mathrm{A} \cup \mathrm{B}) \cap (\mathrm{A} \cup \mathrm{C})$.

Case 2: $t \in^{det} (B \cap C)$ that infers $\mathrm{t} \in^{\mathrm{det}} \mathrm{B} \wedge \mathrm{t} \in^{\mathrm{det}} \mathrm{C}$ which elicits $\mathrm{t} \in^{\mathrm{det}} (\mathrm{A} \cup \mathrm{B}) \wedge \mathrm{t} \in^{\mathrm{det}} (\mathrm{A} \cup \mathrm{C})$, which concludes that $\mathrm{t} \in^{\mathrm{det}} (\mathrm{A} \cup \mathrm{B}) \cap (\mathrm{A} \cup \mathrm{C})$.

We conclude that $\forall t \in^{det} A \cup (B \cap C) \Rightarrow t \in^{det} (A \cup B) \cap (A \cup C)$

So for either determinate or indeterminate timepoints we conclude that:

$$A \cup (B \cap C) \subset (A \cup B) \cap (A \cup C).$$

We will proceed showing that $(A \cup B) \cap (A \cup C)$ is a subset of $A \cup (B \cap C)$.

For indeterminate timepoints, if $t \in^{ind} (A \cup B) \cap (A \cup C)$ then we have two cases

Case 1: $t \in^{ind} (A \cup B) \wedge t \in (A \cup C)$ that splits in four subcases:

1.1 $t \in^{ind} A \wedge t \in^{ind} B \wedge t \in C$ that elicits $t \in^{ind} (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

1.2 $t \in^{ind} A \wedge t \in^{ind} B \wedge t \notin C$ that elicits $t \notin (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

1.3 $t \in^{ind} A \wedge t \notin B$ that elicits $t \notin (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

1.4 $t \notin A \wedge t \in^{ind} B \wedge t \in C$ that elicits $t \in^{ind} (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

Case 2: $t \in (A \cup B) \wedge t \in^{ind} (A \cup C)$ that splits in four subcases:

2.1 $t \in^{ind} A \wedge t \in^{ind} C \wedge t \in B$ that elicits $t \in^{ind} (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

2.2 $t \in^{ind} A \wedge t \in^{ind} C \wedge t \notin B$ that elicits $t \notin (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

2.3 $t \in^{ind} A \wedge t \notin C$ that elicits $t \notin (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

2.4 $t \notin A \wedge t \in^{ind} C \wedge t \in B$ that elicits $t \in^{ind} (B \cap C)$ which concludes that $t \in^{ind} A \cup (B \cap C)$.

We conclude that $\forall t \in^{ind} (A \cup B) \cap (A \cup C) \Rightarrow t \in^{ind} A \cup (B \cap C)$

For determinate timepoints, if $t \in^{det} (A \cup B) \cap (A \cup C)$ then $t \in^{det} (A \cup B) \wedge t \in^{\det} (A \cup C)$ and we have five cases:

Case 1: $t \in^{det} A \wedge t \in^{det} B \wedge t \in^{det} C$ that infers $t \in^{\det} (B \cap C)$, which concludes that $t \in^{\det} A \cup (B \cap C)$.

Case 2: $t \in^{det} A \wedge t \notin B \wedge t \in^{det} C$ that infers $t \notin (B \cap C)$, which concludes that $t \in^{\det} A \cup (B \cap C)$.

Case 3: $t \in^{det} A \wedge t \in^{det} B \wedge t \notin C$ that infers $t \notin (B \cap C)$, which concludes that $t \in^{\det} A \cup (B \cap C)$.

Case 4: $t \in^{det} A \wedge t \notin B \wedge t \notin C$ that infers $t \notin (B \cap C)$, which concludes that $t \in^{\det} A \cup (B \cap C)$.

Case 5: $t \notin A \wedge t \in^{det} B \wedge t \in^{det} C$that infers $t \in^{det} (B \cap C)$, which concludes that $t \in^{\det} A \cup (B \cap C)$.

We conclude for determinate timepoints that

$\forall t \in^{det} (A \cup B) \cap (A \cup C) \Rightarrow t \in^{det} A \cup (B \cap C)$

So for either determinate or indeterminate timepoints we conclude that:

$$(A \cup B) \cap (A \cup C) \subset A \cup (B \cap C)$$

Since $A \cup (B \cap C) \subset (A \cup B) \cap (A \cup C)$ and $(A \cup B) \cap (A \cup C) \subset A \cup (B \cap C)$the equality holds.

Theorem 21 (Distributivity of intersection over union). For the sorted intervals A, B, C the following holds:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Proof: We will examine if each part is a subset of the other for both determinate and indeterminate timepoint inclusion separately. We will first show that $A \cap (B \cup C)$ is a subset of $(A \cap B) \cup (A \cap C)$.

For indeterminate timepoints, if $t \in^{ind} A \cap (B \cup C)$ then we have two cases

Case 1: $t \in^{ind} A \wedge t \in (B \cup C)$ that splits in three subcases:

1.1. $t \in^{ind} A \wedge t \in B \wedge t \in C$ then $t \in^{ind} (A \cap B)$ and $t \in^{ind} (A \cap C)$ which concludes that $t \in^{ind} (A \cap B) \cup (A \cap C)$.
1.2. $t \in^{ind} A \wedge t \notin B \wedge t \in C$ then $t \notin (A \cap B)$ but $t \in^{ind} (A \cap C)$ which concludes that $t \in^{ind} (A \cap B) \cup (A \cap C)$.
1.3. $t \in^{ind} A \wedge t \in B \wedge t \notin C$ then $t \in^{ind} (A \cap B)$ but $t \notin (A \cap C)$ which concludes that $t \in^{ind} (A \cap B) \cup (A \cap C)$.

Case 2: $t \in A \wedge t \in^{ind} (B \cup C)$ we have three subcases:

2.1 $t \in A \wedge t \in^{ind} B \wedge t \in^{ind} C$ then $t \in^{ind} (A \cap B)$ and $t \in^{ind} (A \cap C)$ which concludes that $t \in^{ind} (A \cap B) \cup (A \cap C)$.
2.2 $t \in A \wedge t \in^{ind} B \wedge t \notin C$ then $t \in^{ind} (A \cap B)$ and $t \notin (A \cap C)$ which concludes that $t \in^{ind} (A \cap B) \cup (A \cap C)$.
2.3 $t \in A \wedge t \notin B \wedge t \in^{ind} C$ then $t \notin (A \cap B)$ and $t \in^{ind} (A \cap C)$ which concludes that $t \in^{ind} (A \cap B) \cup (A \cap C)$

So for indeterminate timepoints we conclude that

$$\forall t \in^{ind} A \cap (B \cup C) \Rightarrow t \in^{ind} (A \cap B) \cup (A \cap C)$$

For determinate timepoints, if $t \in^{det} A \cap (B \cup C)$ then we have $t \in^{det} A \wedge t \in^{det} (B \cup C)$ with three subcases:

Case 1: $t \in^{det} A \wedge t \in^{det} B \wedge t \in^{det} C$ that infers $\mathrm{t} \in^{\mathrm{det}} (A \cap B) \wedge \mathrm{t} \in^{\mathrm{det}} (A \cap C)$, which concludes that $\mathrm{t} \in^{\mathrm{det}} (A \cap B) \cup (A \cap C)$.

Case 2: $t \in^{det} A \wedge t \notin B \wedge t \in^{det} C$ that infers $\mathrm{t} \notin (A \cap B) \wedge \mathrm{t} \in^{\mathrm{det}} (A \cap C)$, which concludes that $\mathrm{t} \in^{\mathrm{det}} (A \cap B) \cup (A \cap C)$.

Case 3: $t \in^{det} A \wedge t \in^{det} B \wedge t \notin C$ that infers $\mathrm{t} \in^{\mathrm{det}} (A \cap B) \wedge \mathrm{t} \notin (A \cap C)$, which concludes that $\mathrm{t} \in^{\mathrm{det}} (A \cap B) \cup (A \cap C)$.

We conclude for determinate timepoints that $\forall t \in^{det} A \cap (B \cup C) \Rightarrow t \in^{det} (A \cap B) \cup (A \cap C)$

So for either determinate or indeterminate timepoints we conclude that:

$$A \cap (B \cup C) \subset (A \cap B) \cup (A \cap C).$$

We continue proving that $(A \cap B) \cup (A \cap C)$ is also a subset of $A \cap (B \cup C)$.

For indeterminate timepoints, if $t \in^{ind} (A \cap B) \cup (A \cap C)$ then we have three cases

Case 1: $t \in^{ind} (A \cap B) \wedge t \in^{ind} (A \cap C)$ that splits in two subcases:

1.1 $t \in^{ind} A \wedge t \in B \wedge t \in C$ that elicits $t \in (B \cup C)$ which concludes that $t \in^{ind} A \cap (B \cup C)$.
1.2 $t \in A \wedge t \in^{ind} B \wedge t \in^{ind} C$ that elicits $t \in^{ind} (B \cup C)$ which concludes that $t \in^{ind} A \cap (B \cup C)$.

Case 2: $t \in^{ind} (A \cap B) \wedge t \notin (A \cap C)$ that splits in two subcases:

2.1 $t \in^{ind} A \wedge t \in B \wedge t \notin C$ that elicits $t \in (B \cup C)$ which concludes $t \in^{ind} A \cap (B \cup C)$.
2.2 $t \in A \wedge t \in^{ind} B \wedge t \notin C$ that elicits $t \in^{ind} (B \cup C)$ which concludes $t \in^{ind} A \cap (B \cup C)$.

Case 3: $t \notin (A \cap B) \wedge t \in^{ind} (A \cap C)$ that splits in two subcases:

3.1 $t \in^{ind} A \wedge t \notin B \wedge t \in C$ that elicits $t \in (B \cup C)$ which concludes $t \in^{ind} A \cap (B \cup C)$.
3.2 $t \in A \wedge t \notin B \wedge t \in^{ind} C$ that elicits $t \in^{ind} (B \cup C)$ which concludes $t \in^{ind} A \cap (B \cup C)$.

We conclude that $\forall t \in^{ind} (A \cap B) \cup (A \cap C) \Rightarrow t \in^{ind} A \cap (B \cup C)$

For determinate timepoints, if $t \in^{det} (A \cap B) \cup (A \cap C)$ then we have two cases:

Case 1: $t \in^{det} (A \cap B)$ that infers t $\in^{\det} A \wedge t \in^{det} B$ and $t \in^{det} (B \cup C)$, which concludes that t $\in^{\det} A \cap (B \cup C)$.

Case 2: $t \in^{det} (A \cap C)$ that infers t $\in^{\det} A \wedge t \in^{det} C$ and $t \in^{det} (B \cup C)$, which concludes that t $\in^{\det} A \cap (B \cup C)$.

We conclude for determinate timepoints that

$\forall t \in^{det} (A \cap B) \cup (A \cap C) \Rightarrow t \in^{det} A \cap (B \cup C)$

So for either determinate or indeterminate timepoints we conclude that:

$$(A \cap B) \cup (A \cap C) \subset A \cap (B \cup C)$$

Since $A \cap (B \cup C) \subset (A \cap B) \cup (A \cap C)$ and $(A \cap B) \cup (A \cap C) \subset A \cap (B \cup C)$ the equality holds.

Theorem 22 (DeMorgan's law 1) The complement of the union of two sorted intervals A, B equals the intersection of their compliments.

$$\overline{A \cup B} = \bar{A} \cap \bar{B}$$

Proof: We will examine if each part is a subset of the other for both determinate and indeterminate timepoints. We will first show that $\overline{A \cup B}$ is a subset of $\bar{A} \cap \bar{B}$.

For indeterminate timepoints, if $t \in^{ind} \overline{A \cup B}$ then by definition $t \in^{ind} A \cup B$ that allows three cases

Case 1: t $\in^{ind} A \wedge t \in^{ind} B$ but in that case t $\in^{ind} \bar{A} \wedge t \in^{ind} \bar{B}$ that infers $t \in^{ind} \bar{A} \cap \bar{B}$.

Case 2: t $\in^{ind} A \wedge t \notin B$ but in that case t $\in^{ind} \bar{A} \wedge t \in^{det} \bar{B}$ that infers $t \in^{ind} \bar{A} \cap \bar{B}$.

Case 3: t $\notin A \wedge t \in^{ind} B$ but in that case t $\in^{det} \bar{A} \wedge t \in^{ind} \bar{B}$ that infers $t \in^{ind} \bar{A} \cap \bar{B}$.

For indeterminate timepoints we conclude that $\forall t \in^{ind} \overline{A \cup B} \Rightarrow t \in^{ind} \bar{A} \cap \bar{B}$.

In the case of determinate timepoints, if $t \in^{det} \overline{A \cup B}$ then by definition $t \notin A \cup B$, therefore $t \notin A$ and $t \notin B$ that infers t $\in^{det} \bar{A}$ and t $\in^{det} \bar{B}$ respectively, allowing us to conclude that $t \in^{det} \bar{A} \cap \bar{B}$. Resuming for determinate timepoints it holds that $\forall t \in^{det} \overline{A \cup B} \Rightarrow t \in^{det} \bar{A} \cap \bar{B}$. Since it holds for both sorts of timepoints comprising an interval we safely conclude that

$$\overline{A \cup B} \subset \bar{A} \cap \bar{B}$$

We continue to prove the inverse.

For indeterminate timepoints, if $t \in^{ind} \bar{A} \cap \bar{B}$ we have three cases:

Case 1: t $\in^{ind} \bar{A} \wedge t \in^{ind} \bar{B}$ but in that case t $\in^{ind} A \wedge t \in^{ind} B$ respectively, that infers $t \in^{ind} A \cup B$ and we conclude that $t \in^{ind} \overline{A \cup B}$.

Case 2: t $\in^{ind} \bar{A} \wedge t \in^{det} \bar{B}$ but in that case t $\in^{ind} A$ and $t \notin B$ respectively, that infers $t \in^{ind} A \cup B$ and we conclude that $t \in^{ind} \overline{A \cup B}$.

Case 3: t $\in^{det} \bar{A} \wedge t \in^{ind} \bar{B}$ but in that case t $\notin A$ and $t \in^{ind} B$ respectively, that infers $t \in^{ind} A \cup B$ and we conclude that $t \in^{ind} \overline{A \cup B}$

For indeterminate timepoints we conclude that $\forall t \in^{ind} \bar{A} \cap \bar{B} \Rightarrow t \in^{ind} \overline{A \cup B}$.

In the case of determinate timepoints, if $t \in^{det} \bar{A} \cap \bar{B}$ then $t \in^{det} \bar{A} \wedge t \in^{det} \bar{B}$ so by definition $t \notin A$ and $t \notin B$ respectively, therefore $t \notin A \cup B$ and we conclude $t \in^{det} \overline{A \cup B}$.

So for determinate timepoints it has benn proven that $\forall t \in^{det} \bar{A} \cap \bar{B} \Rightarrow t \in^{det} \overline{A \cup B}$. Since it holds for both sorts of timepoints comprising an interval we safely conclude that

$$\bar{A} \cap \bar{B} \subset \overline{A \cup B}$$

So by $\overline{A \cup B} \subset \bar{A} \cap \bar{B}$ and $\bar{A} \cap \bar{B} \subset \overline{A \cup B}$ we have proven that $\overline{A \cup B} = \bar{A} \cap \bar{B}$.

Theorem 23 (DeMorgan's law 2) The complement of the intersection of two sorted intervals A, B equals the union of their compliments.

$$\overline{A \cap B} = \bar{A} \cup \bar{B}$$

Proof: We will examine if each part is a subset of the other for both determinate and indeterminate timepoints. We will first show that $\overline{A \cap B}$ is a subset of $\bar{A} \cup \bar{B}$.

For indeterminate timepoints, if $t \in^{ind} \overline{A \cap B}$ then by definition $t \in^{ind} A \cap B$ that consists of three cases:

Case 1: t $\in^{ind} A \wedge t \in^{ind} B$ that respectively infer t $\in^{ind} \bar{A}$ and $t \in^{ind} \bar{B}$ so we conclude that $t \in^{ind} \bar{A} \cup \bar{B}$.

Case 2: t $\in^{ind} A \wedge t \in^{det} B$ that respectively infer t $\in^{ind} \bar{A}$ and $t \notin \bar{B}$ allowing us to conclude $t \in^{ind} \bar{A} \cup \bar{B}$.

Case 3: t $\in^{det} A \wedge t \in^{ind} B$ that respectively infer t $\notin \bar{A}$ and $t \in^{ind} \bar{B}$ so we conclude that $t \in^{ind} \bar{A} \cup \bar{B}$.

The above cases allow us to conclude that for indeterminate timepoints $\forall t \in^{ind} \overline{A \cap B} \Rightarrow t \in^{ind} \bar{A} \cup \bar{B}$.

In the case of determinate timepoints, if $t \in^{det} \overline{A \cap B}$ then by definition $t \notin A \cap B$, for which we have five cases:

Case 1: t $\notin A \wedge t \notin B$ that respectively infer t $\in^{det} \bar{A}$ and $t \in^{det} \bar{B}$ so we conclude that $t \in^{det} \bar{A} \cup \bar{B}$.

Case 2: t $\in^{det} A \wedge t \notin B$ that respectively infer t $\notin \bar{A}$ and $t \in^{det} \bar{B}$ allowing us to conclude $t \in^{det} \bar{A} \cup \bar{B}$.

Case 3: t $\in^{ind} A \wedge t \notin B$ that respectively infer t $\in^{ind} \bar{A}$ and $t \in^{det} \bar{B}$ allowing us to conclude $t \in^{det} \bar{A} \cup \bar{B}$.

Case 4: t $\notin A \wedge t \in^{det} B$ that respectively infer t $\in^{det} \bar{A}$ and $t \notin \bar{B}$ so we conclude that $t \in^{det} \bar{A} \cup \bar{B}$.

Case 5: t $\notin A \wedge t \in^{ind} B$ that respectively infer t $\in^{det} \bar{A}$ and $t \in^{ind} \bar{B}$ so we conclude that $t \in^{det} \bar{A} \cup \bar{B}$.

For the determinate timepoints it holds that $\forall t \in^{det} \overline{A \cap B} \Rightarrow t \in^{det} \bar{A} \cup \bar{B}$. Since it holds for both sorts of timepoints comprising an interval we safely conclude that

$$\overline{A \cap B} \subset \bar{A} \cup \bar{B}$$

We continue to prove the inverse.

For indeterminate timepoints, if $t \in^{ind} \bar{A} \cup \bar{B}$ we have three cases:

Case 1: t $\in^{ind} \bar{A} \wedge t \in^{ind} \bar{B}$ but in that case t $\in^{ind} A \wedge t \in^{ind} B$ respectively, that infers $t \in^{ind} A \cap B$ and we conclude that $t \in^{ind} \overline{A \cap B}$.

Case 2: t $\in^{ind} \bar{A} \wedge t \notin \bar{B}$ but in that case t $\in^{ind} A$ and $t \in B$ respectively, that infers $t \in^{ind} A \cap B$ and we conclude that $t \in^{ind} \overline{A \cap B}$.

Case 3: t $\notin \bar{A} \wedge t \in^{ind} \bar{B}$ but in that case t $\in A$ and $t \in^{ind} B$ respectively, that infers $t \in^{ind} A \cap B$ and we conclude that $t \in^{ind} \overline{A \cap B}$

For indeterminate timepoints we conclude that $\forall t \in^{ind} \bar{A} \cup \bar{B} \Rightarrow t \in^{ind} \overline{A \cap B}$.

In the case of determinate timepoints, if $t \in^{det} \bar{A} \cup \bar{B}$ then we have two cases

Case 1: t $\in^{det} \bar{A}$ but in that case t $\notin A$ that infers $t \notin A \cap B$ and we conclude that $t \in^{det} \overline{A \cap B}$.

Case 2: t $\in^{det} \bar{B}$ but in that case t $\notin B$ that infers $t \notin A \cap B$ and we conclude that $t \in^{det} \overline{A \cap B}$.

So for determinate timepoints it has benn proven that $\forall t \in^{det} \bar{A} \cup \bar{B} \Rightarrow t \in^{det} \overline{A \cap B}$. Since it holds for both sorts of timepoints comprising an interval we safely conclude that

$$\bar{A} \cup \bar{B} \subset \overline{A \cap B}$$

So by $\overline{A \cap B} \subset \bar{A} \cup \bar{B}$ and $\bar{A} \cup \bar{B} \subset \overline{A \cap B}$ we have proven that $\overline{A \cap B} = \bar{A} \cup \bar{B}$.

Theorem 24 (Involution) The complement $\bar{\bar{A}}$ of the complement $\bar{A}$ of a sorted interval A is the sorted interval A.

$$\bar{\bar{I}^s} = I^s$$

Proof: Trivial by definition of compliment:

For each indeterminate timepoint $t \in^{ind} A \Rightarrow t \in^{ind} \bar{A} \Rightarrow t \in^{ind} \bar{\bar{A}}.$, and for each determinate timepoint $t \in^{det} A \Rightarrow t \notin^{det} \bar{A} \wedge t \notin^{ind} \bar{A} \Rightarrow t \in^{det} \bar{\bar{A}} \wedge t \notin^{ind} \bar{\bar{A}}$, therefore we conclude $A \subset \bar{\bar{A}}$. And the inverse holds since for each indeterminate timepoint $t \in^{ind} \bar{\bar{A}} \Rightarrow t \in^{ind} \bar{\bar{A}} \Rightarrow t \in^{ind} A$. Also for each determinate timepoint $t \in^{det} \bar{\bar{A}} \Rightarrow t \notin^{det} \bar{A} \wedge t \notin^{ind} \bar{A} \Rightarrow t \in^{det} A \wedge t \notin^{ind} A$, therefore we conclude $\bar{\bar{A}} \subset A$. Finally we conclude that $A = \bar{\bar{A}}$.

4.8. Sorted Interval Relationships

The temporal relationships introduced by Allen [2] have been proved useful even as predicates for expressions in various temporal frameworks. In the presence of indeterminate time these relationships have to be reexamined taking into account the inpresice nature of indeterminate intervals. In the following we will address the evaluation of these operations, introducing a modal approach where the satisfaction of relationships is either definite or potential.

Since an indeterminate interval comprises from timepoints that possibly qualify the associated fact as valid, there is a possibility that this fact also is not valid during the indeterminate time period or that it is valid only during a subset of the included timepoints. Assuming that there is a possibility that only a convex subset of the indeterminate interval will qualify to true, we investigate the relationships that potentially hold when an unevaluated relationship between two sorted intervals holds. In the case of two determinate intervals the relationships that hold is exactly the one that holds for the unevaluated case. It is interesting though to examine the cases of $I_1 \, R \, I_2$ when the first interval is indeterminate while the second is determinate $I_1^i \, R \, I_2^d$; the first interval is determinate while the second is indeterminate $I_1^d \, R \, I_2^i$ and both are indeterminate $I_1^i \, R \, I_2^i$. In the following we examine the later three cases, where we assume that time period has been stated, in terms of their endpoints and for the sated time regions one of the mutual exclusive relationships holds. In the last paragraph all potential relationships are examined in terms of endpoint constraint satisfaction on the last section and are shown in Table 14.

4.8.1. Indeterminate Intervals.

In the case that both convex intervals are indeterminate $I_1^i \, R \, I_2^i$ then

1. If interval I_1 is before interval I_2 then it is definitely before.
2. If I_1 meets I_2 then it is possible that I_1 meets I_2 but it is also possible that I_1 is before I_2. In this case we say that the potential relationships are meets and before.
3. If I_1 is met by I_2 then it is possible that I_1 is met by I_2 but it is also possible that I_1 is after I_2. In this case we say that the potential relationships are met_by and after.

4. If interval I_1 is after interval I_2 then it is definitely after.
5. All other relationships that hold between the two indeterminate convex intervals could potentially produce any other relationship.

	Potential Relationships												
R	b	m	o	s	d	f	e	f^{-1}	d^{-1}	s^{-1}	o^{-1}	m^{-1}	b^{-1}
b	✓												
m	✓	✓					(✓)						
o	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
d	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
f	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
e	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
f^{-1}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
d^{-1}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
s^{-1}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
o^{-1}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
m^{-1}							(✓)					✓	✓
b^{-1}													✓

Table 11 : Potential relationships under "Indeterminate" R "Indeterminate"

The potential relationships are shown in Table 11.

4.8.2. Indeterminate to Determinate.

If the first interval in relationship R, is indeterminate while the second is determinate $I_1^i \; R \; I_2^d$, then

1. $I_1^i \; before \; I_2^d$ the definite relationship is before.
2. $I_1^i \; meets \; I_2^d$ then potentially meets and if
    a. The meeting timepoint is false potentially before.
    b. Only the meeting point is true potentially equal.
3. $I_1^i \; overlaps \; I_2^d$ then potentially overlaps and if
    a. All timepoints of I_1^i that are common with I_2^d are false potentially before.
    b. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially meets.
    c. All timepoints of I_1^i before the start of I_2^d are false potentially starts.
    d. All timepoints of I_1^i before the start of I_2^d including its lower timepoint; are false potentially during.
4. $I_1^i \; starts \; I_2^d$ then potentially starts and if
    a. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially during.
    b. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its lowest then potentially meets.
5. $I_1^i \; during \; I_2^d$ the definite relationship is during.
6. $I_1^i \; finishes \; I_2^d$ then potentially finishes and if
    a. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false,, then potentially during.
    b. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its highest then potentially met by.
7. $I_1^i \; equals \; I_2^d$ then potentially equals and if
    a. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its highest then potentially met by.
    b. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its lowest then potentially meets.

c. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially starts.
d. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially finishes.
e. From timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false from the lower and at least its highest from the higher is false, then potentially during.

8. $I_1^i\ finished\ by\ I_2^d$ then potentially finished by and if
   a. All timepoints of I_1^i that are common with I_2^d are false potentially before.
   b. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially meets.
   c. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially overlaps.

   If all timepoints of I_1^i not common with I_2^d are false and
   d. No other timepoint of I_1^i is false potentially equals.
   e. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its highest then potentially met by.
   f. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially starts.
   g. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially finishes.
   h. From timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false from the lower and at least its highest from the higher is false, then potentially during.
9. $I_1^i\ includes\ I_2^d$ Then potentially includes and since any region of the indeterminate interval may prove to be false, we derive any of the 13 relations as potential relationships between the two intervals. This case does not add any knew knowledge since every ordering or inclusion relation can be potentially derived.
10. $I_1^i\ started\ by\ I_2^d$ Then potentially started by and if
    a. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially overlapped by.
    b. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially meets.
    c. All timepoints of I_1^i that are common with $I_2^d$are false potentially after.

    If all timepoints of I_1^i not common with I_2^d are false and
    d. No other timepoint of I_1^i is false potentially equals.
    e. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially starts.
    f. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially finishes.
    g. From timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false from the lower and at least its highest from the higher is false, then potentially during.
11. $I_1^i\ overlapped\ by\ I_2^d$ Then potentially overlapped by and if
    a. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially met by.
    b. All timepoints of I_1^i that are common with $I_2^d$are false potentially after.
    c. If all timepoints of I_1^i not common with I_2^d are false potentially finishes
    d. If all timepoints of I_1^i not common with I_2^d and at least the latest common timepoint are false the potentially during.
12. $I_1^i\ met\ by\ I_2^d$ Then potentially met by and if
    a. The meeting point is false potentially after.
    b. Only the meeting point is true potentially equal.

13. $I_1^i \; after \; I_2^d$ Then potentially after

In the following Table 12 we can see the potential relations that are satisfied, in parentheses are the cases that the indeterminate interval degrades to a timepoint.

	Potential Relationships												
R	b	m	o	s	d	f	e	f^{-1}	d^{-1}	s^{-1}	o^{-1}	m^{-1}	b^{-1}
b	✓												
m	✓	✓											
o	✓	✓	✓	✓	✓								
s		(✓)		✓	✓								
d					✓								
f					✓	✓						(✓)	
e		(✓)		✓	✓	✓	✓					(✓)	
f^{-1}	✓	✓	✓	✓	✓	✓	✓	✓				(✓)	
d^{-1}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
s^{-1}		(✓)		✓	✓	✓	✓			✓	✓	✓	✓
o^{-1}					✓	✓					✓	✓	✓
m^{-1}												✓	✓
b^{-1}													✓

Table 12 : The Potential Relationships when: "Indeterminate" R " Determinate"

4.8.3. Determinate to Indeterminate.

If the first interval in relationship R, is indeterminate while the second is determinate $I_1^i \; R \; I_2^d$, then

1. $I_2^d \; before \; I_1^i$ then potentially before.
2. $I_2^d \; meets \; I_1^i$ then potentially meets and if
    a. The meeting timepoint is false potentially before.
3. $I_2^d \; overlaps \; I_1^i$ then potentially overlaps and if
    a. All timepoints of I_1^i that are common with I_2^d are false potentially before.
    b. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially meets.
    c. All timepoints of I_1^i after the end of I_2^d are false potentially finishes.
    d. All timepoints of I_1^i after the end of I_2^d including the last timepoint of timepoint I_2^d; are false potentially includes.
4. $I_2^d \; starts \; I_1^i$ then potentially starts and if
    a. From lower timepoints of I_1^i , at least its lowest timepoint is false, then potentially overlaps.
    b. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially meets.
    c. All timepoints of I_1^i that are common with $I_2^d$are false potentially before.

    If all timepoints of I_1^i not common with I_2^d are false and
    d. No other timepoint of I_1^i is false potentially equals.
    e. And from common timepoints at least its highest timepoint is false, then potentially started by.
    f. And from common timepoints, at least its lowest timepoint is false, then potentially finished by.

g. And from common timepoints at least its lowest timepoint is false from the lower and at least its highest from the higher is false, then potentially includes.

5. I_2^d $during$ I_1^i Then potentially includes and since any region of the indeterminate interval may prove to be false, we derive any of the 13 relations as potential relationships between the two intervals. This case does not add any knew knowledge since every ordering or inclusion relation can be potentially derived.
6. I_2^d $finishes$ I_1^i then potentially finishes and if
    a. All timepoints of I_1^i that are common with I_2^d are false potentially after.
    b. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially met by.
    c. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially overlapped by.

    If all timepoints of I_1^i not common with I_2^d are false and
    d. No other timepoint of I_1^i is false potentially equals.
    e. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its highest then potentially meets.
    f. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially started by.
    g. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially finished by.
    h. From timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false from the lower and at least its highest from the higher is false, then potentially includes.
7. I_2^d $equals$ I_1^i then potentially equals and if
    a. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its lowest then potentially met by.
    b. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its highest then potentially meets.
    c. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially started by.
    d. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially finished by.
    e. From timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false from the lower and at least its highest from the higher is false, then potentially includes.
8. I_2^d $finished\ by$ I_1^i then potentially finished by and if
    a. From higher timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its highest timepoint is false, then potentially includes.
    b. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its highest then potentially meets.
9. I_2^d $includes$ I_1^i Then potentially includes
10. I_2^d $started\ by$ I_1^i Then potentially started by and if
    a. From lower timepoints of $\mathrm{I}_1^{\mathrm{i}}$, at least its lowest timepoint is false, then potentially includes.
    b. $\mathrm{I}_1^{\mathrm{i}}$ degrades to a timepoint, its lowest then potentially met by.
11. I_2^d $overlapped\ by$ I_1^i Then potentially overlapped by and if
    a. All timepoints of I_1^i that are common with I_2^d are false potentially after.
    b. All timepoints of I_1^i that are common with $I_2^d$are false except the meeting point potentially met by.
    c. All timepoints of I_1^i before the start of I_2^d are false potentially started by.
    d. All timepoints of I_1^i before the start of I_2^d including its lower common timepoint; are false potentially includes.
12. I_2^d $met\ by$ I_1^i Then potentially met by and if
    a. The meeting point is false potentially after.

13. $I_1^i \; after \; I_2^d$ Then potentially after

In the following Table 13 we can see the potential relations that are satisfied, in parentheses are the cases that the indeterminate interval degrades to a timepoint.

	Potential Relationships												
R	b	m	o	s	d	f	e	f^{-1}	d^{-1}	s^{-1}	o^{-1}	m^{-1}	b^{-1}
b	✓												
m	✓	✓											
o	✓	✓	✓					✓	✓				
s	✓	✓	✓	✓			✓	✓	✓	✓		(✓)	
d	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
f		(✓)				✓	✓	✓	✓	✓	✓	✓	✓
e		(✓)					✓	✓	✓	✓		(✓)	
f^{-1}		(✓)						✓	✓				
d^{-1}									✓				
s^{-1}									✓	✓		(✓)	
o^{-1}									✓	✓	✓	✓	✓
m^{-1}												✓	✓
b^{-1}													✓

Table 13: The Potential Relationships when: "Determinate" R "Indeterminate"

4.8.4. Potential Relationships

Exploring the relationships in the presence of indeterminacy we have seen that if a relationship is potentially satisfied there is more than one relationship between the two intervals, seen as unevaluated time periods that are satisfied. If both intervals are determinate then potential satisfaction has the same requirements as definite satisfaction (see Table 1), but potential satisfaction of a relation has weaker endpoint constraint requirements; thus is more flexible than definite satisfaction when at least one interval is indeterminate. Potential relationships are taking into account the possibility that an indeterminate interval may eventually partially fail, under later added knowledge in the system and for that reason will let as express constraints or conditions not as strict as definite relationships do. For example if we know that an employee worked on a project A determinately from June 15 to July 31 and want to add that the same employee will work on project B from July 30 to August 31, while there is a constraint that an employee works only on one project at each time then by a definite expression of this constraint the second fact cannot be inserted into the system since it violates the strict condition, but if we allow potential satisfaction then insertion of this fact with its indeterminate region will be allowed.

Supporting potential relationships, expressions of constraints on the endpoints of intervals are devised, as seen in Table 14. In the following we will see each case separately, only for the cases that an indeterminate interval is one of the operands:

4.8.4.1. Potential Relationships for both Indeterminate Intervals

If both intervals are indeterminate then for the potential relationships R: $I_1^i \; R^{pot} \; I_2^i$we have:

1. Potentially I_1^i before I_2^i requires that the lowest timepoint of the first operand I_1^i is before the highest timepoint of the second operand I_2^i, thus $I_{1s} < I_{2e}$ so there is at least a timepoint of I_1^i that is before I_2^i even if they degrade to timepoints so the

point to interval (Table 2) and point to point (Table 3) relations are also accommodated.

2. Potentially I_1^i meets I_2^i requires that both intervals have at least a common timepoint thus the first interval should end after the start of the second $I_{1e} \geq I_{2s}$ and $I_{1s} \leq I_{2e}$ while there is another timepoint of the first interval before the second interval $I_{1s} < I_{2e}$. So the constraints are $I_{1e} \geq I_{2s}$ and $I_{1s} < I_{2e}$.
3. Potentially I_1^i after I_2^i requires that the lowest timepoint of the first operand I_1^i is after the highest timepoint of the second operand I_2^i, thus $I_{1s} > I_{2e}$ so there is at least a timepoint of I_1^i that is after I_2^i even if they degrade to timepoints so the point to interval (Table 2) and point to point (Table 3) relations are also accommodated.
4. Potentially I_1^i met by I_2^i requires that both intervals have at least a common timepoint thus the first interval should end after the start of the second $I_{1e} \geq I_{2s}$ and $I_{1s} \leq I_{2e}$ while there is another timepoint of the first interval after the second interval $I_{1e} > I_{2s}$. So the constraints are $I_{1e} > I_{2s}$ and $I_{1s} \leq I_{2e}$.
5. Potentially I_1^i met by I_2^i requires that the second interval ends at the beginning or after the beginning of the first interval $I_{1s} \leq I_{2e}$ and the first interval ends at the beginning or after the beginning of the first interval: $I_{1e} > I_{2s}$. In the case that both intervals degrade to a timepoint then the point to point equality holds.
6. All other potential relationships require at least a common region of the two intervals; therefore we require that the ending timepoint of the second interval is after the beginning of the first $I_{1s} < I_{2e}$ and respectively the ending of the first interval is after the beginning of the second interval $I_{1e} > I_{2s}$.

4.8.4.2. Potential Relationships with mixed Intervals

If the two convex intervals are indeterminate and determinate then we have two cases the first interval is determinate while the second is indeterminate or the first interval is indeterminate while the second is determinate. The first case $I_1^d \; R^{pot} \; I_2^i$ is examined in the following while for the second case a potential relationship between a Indeterminate and a Determinate Interval is tested $I_2^i \; R^{pot} \; I_2^d$ there is an equality of the definitions with respect to the definitions of the inverse relationships of first case.

For a potential relationship of a determinate first operand and an indeterminate second operand: $I_1^d \; R^{pot} \; I_2^i$, the potential relationships in terms of the endpoints are defined as:

1. Potentially I_1^d before I_2^i requires that the highest timepoint of the determinate operand I_1^d is before the highest timepoint of the indeterminate operand I_2^i, thus $I_{1e} < I_{2e}$ so all timepoints of the determinate interval are before at least the highest timepoint of the indeterminate interval I_2^i .
2. Potentially I_1^d meets I_2^i requires that both intervals have at least a common timepoint thus the determinate operand I_1^d should end before or at the end of the indeterminate interval I_2^i thus$I_{1e} \leq I_{2e}$ and after or at the starting point of the indeterminate interval $I_{1e} \geq I_{2s}$.
3. Potentially I_1^d overlaps I_2^i requires that both intervals have more than one common timepoints thus the determinate operand I_1^d should end before of the indeterminate interval I_2^i thus $I_{1e} < I_{2e}$ and after the starting point of the indeterminate interval $I_{1e} > I_{2s}$.
4. Potentially I_1^d starts I_2^i then both intervals have more than one common timepoints and the determinate operand I_1^d should start after or at the start of the indeterminate interval I_2^i thus $I_{1s} \geq I_{2s}$ but also the determinate interval should end before the ending point of the indeterminate interval $I_{1e} < I_{2e}$.

Potential Relation	I_1 Indeterminate I_2 Indeterminate	I_1 Determinate I_2 Indeterminate	I_1 Indeterminate I_2 Determinate
I_1 before I_2	$I_{1s} < I_{2e}$	$I_{1e} < I_{2e}$	$I_{1s} < I_{2s}$
I_1 meets I_2	$I_{1s} < I_{2e} \wedge I_{1e} \geq I_{2s}$	$I_{1e} \leq I_{2e} \wedge I_{1e} \geq I_{2s}$	$I_{1s} \geq I_{2s} \wedge I_{1e} \geq I_{2s}$
I_1 overlaps I_2	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1e} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1s} > I_{2s} \wedge I_{1e} > I_{2s}$
I_1 starts I_2	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1e} < I_{2e} \wedge I_{1s} \geq I_{2s}$	$I_{1s} \leq I_{2s} \wedge I_{1e} > I_{2s}$
I_1 during I_2	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1e} < I_{2e} \wedge I_{1s} > I_{2s}$	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$
I_1 finishes I_2	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1e} \leq I_{2e} \wedge I_{1s} > I_{2s}$	$I_{1s} < I_{2e} \wedge I_{1e} \geq I_{2e}$
I_1 equalsI_2	$I_{1s} \leq I_{2e} \wedge I_{1e} \geq I_{2s}$	$I_{1e} \leq I_{2e} \wedge I_{1s} \geq I_{2s}$	$I_{1s} \leq I_{2s} \wedge I_{1e} \geq I_{2e}$
I_2 finished by I_1	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1e} \leq I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1s} < I_{2s} \wedge I_{1e} \geq I_{2e}$
I_2 includes I_1	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1e} > I_{2s} \wedge I_{1s} < I_{2e}$	$I_{1s} < I_{2s} \wedge I_{1e} > I_{2e}$
I_2 started by I_1	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1s} \geq I_{2s} \wedge I_{1s} < I_{2e}$	$I_{1s} \leq I_{2s} \wedge I_{1e} > I_{2e}$
I_2 overlapped by I_1	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1s} > I_{2s} \wedge I_{1s} < I_{2e}$	$I_{1s} < I_{2e} \wedge I_{1e} > I_{2e}$
I_2 met by I_1	$I_{1s} \leq I_{2e} \wedge I_{1e} > I_{2s}$	$I_{1s} \geq I_{2s} \wedge I_{1s} \leq I_{2e}$	$I_{1s} \leq I_{2e} \wedge I_{1e} \geq I_{2e}$
I_2 after I_1	$I_{1s} > I_{2e}$	$I_{1s} > I_{2s}$	$I_{1e} > I_{2e}$

Table 14 : The potential relations in terms of endpoints

5. Potentially I_1^d during I_2^i then both intervals have more than one common timepoints and the determinate operand I_1^d should start after the start of the indeterminate interval I_2^i thus $I_{1s} > I_{2s}$ but also the determinate interval should end before the ending point of the indeterminate interval $I_{1e} < I_{2e}$.
6. Potentially I_1^d finishes I_2^i then the determinate operand I_1^d should start after the start of the indeterminate interval I_2^i thus $I_{1s} > I_{2s}$ and the determinate interval should end before or at the ending point of the indeterminate interval $I_{1e} \leq I_{2e}$.
7. Potentially I_1^d equals I_2^i then the determinate operand I_1^d should start after or at the start of the indeterminate interval I_2^i thus $I_{1s} \geq I_{2s}$ and the determinate interval should end before or at the ending point of the indeterminate interval $I_{1e} \leq I_{2e}$.
8. Potentially I_1^d finished by I_2^i then the determinate operand I_1^d should end after the start of the indeterminate interval I_2^i thus $I_{1e} > I_{2s}$ but the determinate interval should end before or at the ending point of the indeterminate interval $I_{1e} \leq I_{2e}$.
9. Potentially I_1^d includes I_2^i then the determinate operand I_1^d should end after the start of the indeterminate interval I_2^i thus $I_{1e} > I_{2s}$ and the determinate interval should start before the ending point of the indeterminate interval $I_{1s} < I_{2e}$.
10. Potentially I_1^d started by I_2^i then the determinate operand I_1^d should start before the end of the indeterminate interval I_2^i thus $I_{1s} < I_{2e}$ and the determinate interval should start after or at the starting point of the indeterminate interval $I_{1s} \geq I_{2s}$.
11. Potentially I_1^d overlapped by I_2^i then the determinate operand I_1^d should start before the end of the indeterminate interval I_2^i thus $I_{1s} < I_{2e}$ and the determinate interval should start after the starting point of the indeterminate interval $I_{1s} > I_{2s}$.
12. Potentially I_1^d met by I_2^i then the determinate operand I_1^d should start before or at the end of the indeterminate interval I_2^i thus $I_{1s} \leq I_{2e}$ and the determinate interval should start after or at the starting point of the indeterminate interval $I_{1s} \geq I_{2s}$.
13. Potentially I_1^d after I_2^i then the determinate operand I_1^d should start after the starting point of the indeterminate interval $I_{1s} > I_{2s}$.

Chapter 5

Conclusions and Future Work

In this this we have examined we have explored a work on founding a temporal algebra in order to support indeterminacy. We started this chain of work in chapter 3, defining the timeline that consist of time instants on a discrete, linear and unbounded timeline. We followed with the definition of our primitive temporal elements that are interval allowing representations of timepoints as instants. We described the operations of union, intersection, complement and difference for the general case that does not support indeterminacy. We also described metric functions of the duration and count of an interval. All these operations are needed by a database in order to support relational operations. Supporting convex and non-convex intervals enhances previous models about time. The use of simple periods as primitive constructs allows for a user-friendly representation of time that are comprehended well by users, exploiting existing support of time periods in the relational databases without a need of change primitive data types, but still providing a compact representation and computationally efficient.

The definitions of chapter 3 provided the fundaments for the incorporation of determinate and indeterminate time in succeeding chapter 4, as temporal intervals that have a special evaluation. We explored the semantics for each operation in terms of the interrelation of determinate and indeterminate time and using operations of the unevaluated case reconstructed all operations extending the determinate case to handle indeterminacy for both the convex and non-convex intervals. We have also explored the relationships between intervals investigating in the presence of the indeterminacy introduced the relationships that potentially satisfy for each interval relation. This research is novel in the way that first semantics of the interrelation are proposed outside of the probabilistic scope and then operations are introduced respecting the semantics. The accompanied data model inheriting its simplicity of the primitive temporal element of the period allows user-friendliness and efficiency extending the determinate case, while decoupling indeterminate time regions from determinate ones.

There are extensions possible as future directions of the current work, such as devising new data models that respect the interval based semantics in terms of durations of the intervals. Data models that allow a minimum or/and a maximum interval length on the stored data can be devised based on the semantics already proposed. This work may also be exploited by the numerous constraint propagation algorithms based on the interval Algebra by Allen [2] by using the potential relations that are satisfied in the presence of indeterminacy and investigate the results with respect to reasoning power and complexity.Another possible extension would be to deal with the support of multiple granularities or with relative temporal elements.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of Databases*.: Addison-Wesley, 1995.

[2] James F. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, vol. 26, November 1983.

[3] J. F. Allen and Patrick J. Hayes, "A Common-Sense Theory of Time," in *IJCAI'85 Proceedings of the 9th international joint conference on Artificial intelligence*, vol. 1, San Francisco, 1985.

[4] Luca Anselma, Paolo Terenziani, and Richard T. Snodgrass, "Valid-Time Indeterminacy in Temporal Relational Databases: A Family of Data Models," in *TIME*, 2010.

[5] Philippe Balbiani, Jean-Francois Condotta, and Gerard Ligozat, "Reasoning about generalized intervals:Horn representability and tractability," in *TIME '00 Proceedings of the Seventh International Workshop on Temporal Representation and Reasoning (TIME'00)*, 2000.

[6] Claudio Bettini, Sushil Jajodiag, and Sean Wang, *Time granularities in databases, data mining, and temporal reasoning*.: Springer, 2000.

[7] Michael H. Böhlen, Renato Bussato, and Christian S. Jensen, "Point- Versus Interval - based Temporal Data Models," in *Proceedings 14th International Conference on Data Engineering, *, Orlando, 1998.

[8] Michael H. Böhlen, Jan Chomicki, Richard T. Snodgrass, and David Toman, *Querying TSQL2 Databases with Temporal Logic*, 96th ed.: Springer, 1996, vol. 1057.

[9] Michael H. Böhlen, Richard T. Snodgrass, and Michael D. Soo, "Coalescing in Temporal Databases," in *VLDB*, 1996, pp. 180-191.

[10] Mathias Broxvall and Peter Jonsson, "Point algebras for temporal reasoning:Algorithms and complexity," *Artificial Intelligence*, no. 149, pp. 179–220, Feb 2003.

[11] Christophe Claramunt, "Extending Ladkin's algebra on non-convex intervals towards an algebra on union-of regions.," in *Proceedings of the Eighth ACM Symposium on Advances in Geographic Information Systems*, Washington D.C., 2000, pp. 9-14.

[12] Wes Cowley and Dimitris Plexousakis, "An Interval Algebra for Indeterminate Time," in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence* , 2000, pp. 470-475.

[13] Wes Cowley and Dimitris Plexousakis, "Temporal Integrity Constraints with Indeterminacy," in *Proceedings of the 26th VLDB Conference*, Kairo, 2000.

[14] H. Darwen and C. J. Date. (2005, March) An Overview and Analysis of Proposals Based on the TSQL2 Approach. http://www.dcs.warwick.ac.uk/~hugh/TTM/OnTSQL2.pdf.

[15] C. J. Date, Hugh Darwen, and Nikos A. Lorentzos, *Temporal Data And The Relational Model*. San Fransisco: Morgan Kaufmann, 2003.

[16] Alex Dekhtyar, Robert Ross, and V.S. Subrahmanian, "Probabilistic temporal databases, I: algebra," *ACM Transactions on Database Systems*, vol. 26, no. 1, March 2001.

[17] Debabrata Dey, Terence M. Barron, and Veda C. Storey, "A complete temporal relational Algebra," *The VLDB Journal*, 1996.

[18] Joseph Dunn, Sean Davey, Anne Descour, and Richard T. Snodgrass, "Sequenced Subset Operators: Definition and Implementation," in *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[19] Curtis E. Dyreson, "Valid-Time Indeterminacy," The University of Arizona, PhD Thesis 1994.

[20] Curtis E. Dyreson, William S. Evans, Hong Lin, and Richard T. Snodgrass, "Efficiently Supporting Temporal Granularities," vol. 12, pp. 568 - 587, Jul/Aug 2000.

[21] Curtis E. Dyreson and Richard T. Snodgrass, "Supporting Valid-Time Indeterminacy," *ACM Transactions on Database Systems* , vol. 23, no. 1, March 1998.

[22] Shashi K. Gadia, Sunil S. Nair, and Yiu-Cheong Poon, "Incoplete Information in relational databases," in *Proceedings of the 18th VLDB Conference*, Vancouver, 1992, pp. 395-406.

[23] Dengfeng Gao, Christian Jensen, Richard T. Snodgrass, and Michael D. Soo, "Join operations in temporal databases," *The VLDB Journal*, vol. 14, no. 1, pp. 2-29, 2005.

[24] Valentin Goranko, Angelo Montanari, and Guido Sciavicco, "A Road Map on Interval Temporal Logics and Duration Calculi," *Journal of Applied Nonclassical Logics*, vol. 14, no. 1-2, 2004.

[25] Heidi Gregensen and Christian S. Jensen, "Conceptual Modeling of Time-Varying Information," September 1998.

[26] Antony Griffiths and Babis Theodoulidis, "SQL+ i: Adding Temporal Indeterminacy to the Database Language SQL," , 1996.

[27] Guang Ping He, "An experimentally testable proof of the discreteness of time," *Quantum Physics*, November 2009.

[28] Christian S. Jensen. (2000) Introduction to Temporal Database Research. http://people.cs.aau.dk/~csj/Thesis/.

[29] Christian S. Jensen and Richard T. Snodgrass, "Semantics of Time-Varying Information," *Information Systems*, vol. 21, no. 4, June 1996.

[30] Christian S. Jensen and Richard T. Snodgrass, "Temporal Data Management," *IEEE Transactions On Knowledge And Data Engineering*, vol. 11, no. 1, pp. 36-44, Jan/Feb

1999.

[31] Christian S Jensen and Richard T. Snodgrass, "Temporal Database Entries for the Springer Encyclopedia of Database Systems," Technical 2008.

[32] Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo, "The TSQL2 Data Model," in *The TSQL2 Temporal Query Language*., 1995, pp. 153-238.

[33] Tom Johnston and Randal Weis, *Managing Time In Relational Databases*. Burlington: Elsevier, 2010.

[34] Manolis Koubarakis, "Database Models for Infinite and Indefinite Temporal Information," *Information Systems*, vol. 19, no. 2, pp. 141-173, 1994.

[35] Manolis Koubarakis, "Databases and Temporal Constraints: Semantics and Complexity," , 1995.

[36] Manolis Koubarakis, "Temporal CSPs," in *Handbook of Constraint Programming*.: Elsevier, 2006.

[37] Adrei Krokhin, Peter Jeavons, and Peter Jonsson, "Reasoning About Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra," *Journal of the ACM*, vol. 50, no. 5, pp. 591-640, 2003.

[38] Peter Ladkin, "Models of Axioms for Time Intervals," in *AAAI'87 Proceedings of the sixth National conference on Artificial intelligence*, vol. 1, Seattle, Washington, 1987, pp. 234-239.

[39] Peter Ladkin, "Primitives and Units for Time Specification," in *Fifth National Conference on Artificial Intelligence AAAI-86*, vol. 1, Philadelphia, 1986, pp. 353-359.

[40] Peter Ladkin, "Time Representation: A Taxonomy of Interval Relations," 1986.

[41] Bruce Leban, David D. McDonald, and David R. Forster, "A representation for collections of temporal intervals," in *In Proccedings AAAI-86 Fifth National Conference on Artificial Intelligence*, Philadelphia, 1986, pp. 367-371.

[42] Nadeem Mahmood, Aqil Burney, and Ahsan Kamran, "A Logical Temporal Relational Algebra," *International Journal of Computer Science isuues*, vol. 7, no. 1, January 2010.

[43] Edwin L. McKenzie Jr. and Richard T. Snodgrass, "Evaluation of relational algebras incorporating the time dimension in databases," *ACM Comput. Surv.*, vol. 23, no. 4, pp. 501-543, Dec. 1991.

[44] Mohamed Mkaouar, Rafik Bouaziz, and Mohamed Moalla, "Querying and manipulating temporal databases," *Internationa Journal of Database Management Systems*, February 2011.

[45] John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis, "Telos: Representing Knowledge About Information Systems," *ACM Transactions on Information Systems*, vol. 8, no. 4, October 1990.

[46] Dimitris Plexousakis, "Integrity Constraints and Rule Maintenance in Temporal Deductive Knowledge Bases," in *In Proceedings of the 19th International Conference on Very Large Data Bases*, Dublin, 1993, pp. 146-157.

[47] Jia-Dong Ren, Hui-Li Peng, and Xiao-Jian Zhang, "ITRM: A Temporal Data Model Supporting Indeterminacy," *International Journal of Computer Science and Network Security*, vol. 6, September 2006.

[48] Steven Schockaert, Martine De Cock, and Etienne E. Kerre, "Imprecise Temporal Interval Relations," in *Fuzzy Logic and Applications, 6th International Workshop*, Crema, 2005, pp. 108-113.

[49] Richard T. Snodgrass, *Developing Time-Oriented Database Applications in SQL*. San Fransisco, U.S.A.: Morgan Kaufmann, 2000.

[50] Michael D. Soo, Christian S. Jensen, and Richard T. Snodgrass, "An Algebra for TSQL2," in *The TSQL2 Temporal Query Language*., 1995.

[51] Andreas Steiner, "A Generalisation Approach to Temporal Data Models and their Implementations," Swiss Federal Institute of Technology, Zurich, PhD Thesis 1998.

[52] Paolo Terenziani and Luca Anselma, "A Knowledge Server for Reasoning about Temporal Constraints Between Classes and Instances of Events," *International Journal of Intelligent Systems*, vol. 19, no. 10, pp. 919-947, Oct 2004.

[53] David Toman and Jan Chomicki, "Temporal databases," in *Handbook of Temporal Reasoning in Artificial Intelligence*.: Elsevier, 2005, vol. 1, ch. 14, pp. 429 - 467.

[54] Vassilis J. Tsotras and X. Sean Wang, "Temporal Databases," in *Wiley Encyclopedia of Electrical and Electronics Engineering*.: John Wiley & Sons, Inc., 2001, pp. 628-641.

[55] Alexander Tuzhilin and James Clifford, "A Temporal Relational Algebra as a Basis for Temporal Relational Completeness (October 1990).," in *16th International Conference on Very Large Data Bases*, Brisbane, Oct. 1990, pp. 13-23.

[56] Octavian Udrea, Zoran Majkic, and V. S. Subrahmanian, "Aggregates in generalized temporally indeterminate databases," , 2007.

[57] J. van Benthem,. Dordrecht, Holland: Kluwer Academic, 1991, pp. 14-18.

[58] Yde Venema, "Temporal Logic," in *The Blackwell Guide to Philosophical Logic*, L Goble, Ed. Malden, USA: Blackwell Publishers, 2001, pp. 203 - 223.

[59] Marc Vilain, "A system for reasoning about time," *AAAI*, pp. 197-201, 1982.

[60] Marc Vilain, Henry Kautz, and Peter van Beek, "Constraint Propagation Algorithms for Temporal Reasoning: A revised Report," in *AAAI-86 Proceedings*, 1986.

[61] Burghard von Karger, "Temporal Algebra," in *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, Roland Carl Backhouse, Roy L. Crole, and Jeremy Gibbon, Eds. Oxford: Springer, 2002.

[62] Yihong Yuan and Yu Liu, "Modeling temporal uncertainty of events: a descriptive perspective," in *Sixth international conference on Geographic Information Science*, Zurich , 2010.

Appendix A

Optimizations

For the case that non-convex intervals are ordered we provide optimizations for algorithms.

The union is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \rightarrow NC\mathcal{I}^*$.

Algorithm 2($\cup^{\mathcal{CI}-NC\mathcal{I}}$: Convex over non-convex union). If I is the empty interval then $N_\cup = N$ else an iteration (or recursion) will be needed over the members of the non-convex interval. Inputs are the convex interval I and the non-convex interval $N = \{I_1, \dots, I_i, \dots, I_n\}$ and output is $N_\cup$. We have three cases:

Case 1: if I is before a member I_i of the N, we insert I and then we I_i in the output $N_\cup$ and also insert all remaining members of N since all remaining members of N are after I and will not merge.

Case 2: if I is after a member I_i of the N, we insert I_i in the output $N_\cup$ and then we continue with the next member N .

Case 3: if $I$and I_i are mergeable a merge should take place that results I_m. We then continue without insertion in the $N_\cup$ on the next member I_{i+1} of N and in the place of I we will use the merged I_m convex interval.

Algorithm 3: ($\cup^{NC\mathcal{I}}$) We assume as input the non-convex intervals $N_1 = \{I_{11}, \dots, I_{1i}, \dots, I_{1n}\}$ with count $|N_1| = n$ and $N_2 = \{I_{21}, \dots, I_{2j}, \dots, I_{2m}\}$ with count $|N_2| = m$. Output will be the non-convex interval $N_\cup$. Two iterations must be performed for each member of the one interval over each member of the other. For each member of the first interval I_{1i} the *The union* is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \rightarrow NC\mathcal{I}^*$.

Algorithm 2 will be used over N_2. The output of the *The union* is a function with signature $\mathcal{CI} \times NC\mathcal{I}^* \rightarrow NC\mathcal{I}^*$.

Algorithm 2 will be used with I_{1i+1} on the next iteration. The algorithm terminates when every interval of N_1 is used against intervals of N_2.

1. Initialize $N_\cup^{NC\mathcal{I}} = \{\}$
2. Repeat over each $I_1 = I_i \in N_1$ and
    a. Repeat over each $I_2 = I_j \in N_2$
        i. If I_1 before I_2 insert I_1 in $N_\cup^{NC\mathcal{I}}$ continue with $I_1 = I_{i+1}$ and $I_2 = I_j$
        ii. If I_1 after I_2 insert I_2 in $N_\cup^{NC\mathcal{I}}$ continue with $I_1 = I_i$ and $I_2 = I_{j+1}$
        iii. If I_1 and I_2 mergeable merge as I_m and continue with $I_1 = I_m$ and $I_2 = I_{j+1}$
    b. Insert I_1 in $N_\cup^{NC\mathcal{I}}$ continue with $I_1 = I_{i+1}$

Algorithm 4 $\cap^{\mathcal{CI}-NC\mathcal{I}}$Intersecting a convex interval I and a non-convex interval $N = \{I_1, \dots, I_i, \dots, I_n\}$ will result a non-convex interval $N_\cap$. An iteration (or recursion) will be needed over the members of the non-convex interval. On each iteration we test Iagainst the current member I_i of N. We have two cases:

Case 1: I or N are the empty interval then return the empty non convex interval.

Case 2: I and I_i are intersecting. We execute the intersection as described in *Definition 20* and will insert the resulting convex interval in the output $N_\cap$ since it contains all common timepoints. We continue with the convex intersection of the next member I_{i+1} of $N$and the convex interval I.

Case 3: I is before I_i they are not intersecting. We terminate the iteration since all following members will be after I and will not intersect with I.

Algorithm 6: Optimization of $(\overline{N})$ for the ordered case. The complement of a non-convex interval $N_1 = \{I_{11,} \dots, I_{1i}, \dots, I_{1n}\}$ with count $|N_1| = n$ and with ordered members. We calculate the complement with the convex complement function for the first member. We have two cases:

Case 1: The complement of the first interval is a single convex interval and is the empty element the non-convex interval had the total infinite convex interval as member and by construction it does not contain any more members otherwise they would be merged.

Case 2: The complement of the first interval is a single convex interval and is the total infinite interval. The non-convex interval is the empty.

Case 3: The complement of the first interval is a single convex interval and is the left infinite interval. The non-convex interval contains has as member the right infinite convex interval.

Case 4: The complement of the first interval is a single convex interval and is the right infinite interval. The non-convex interval contains has as member the left infinite convex interval.

Case 5: The complement of the first interval has two convex intervals.

For cases 1, 2 and 3 we terminate iteration since there should be no more members in the convex interval that would not have been merged, by construction of the NCI.

For cases 4 and 5 we change the highest endpoint of the right part of the complement with the previous timepoint of the next member and also create the new right complement with lowest endpoint the next timepoint of the highest endpoint of the next member and highest endpoint the $+\infty$ and continue with this right complement to the next member until all members are taken in account.

The time complexity of this algorithm is $\mathcal{O}(|N|\ )$.

Appendix B

A java implementation

The implementation in Java consists of three packages:

1. The *times package* that provides an interface for the timepoints and their functionality a time instant of the discrete timeline should provide. In this package an implementation for the special characters is provided and a reference to a numeric timepoint.
2. The *convex intervals package* that provides the CIntervals interface for the convex time periods. Implementations for the general unsorted convex time period but also for the indeterminate and determinate convex time periods as extensions of the general convex case are provided. Dependencies to the times package exist since the end points of the intervals are of types that implement the Time interface of the times package.
3. The *non-convex intervals package* that provides an interface for the non-convex time periods. Implementations for the non-convex time period the sorted non-convex time periods as extensions of the general non-convex case are provided. Dependencies to the *times package* exist since the end points of the convex intervals that are members of a non-convex interval are of types that implement the Time interface of the times package. Also dependencies to the convex intervals package exist since the members of a non-convex interval are convex intervals that implement the CIntervals interface .

The Times package:

The Time Interface

```
public interface Time {
/**
 * Calculates the previous Time Instant related to this Time instant
 *  in the succession of instants over the Time Line.
 * @return the previous Time Instant
 */
public Time previous();

/**
 * Calculates the next Time Instant related to this Time instant in
 * the succession of instants over the Time Line.
 * @return the next Time Instant
 */
public Time next();

/**
 *  Calculates the earliest Time Instant among this Time instant and
 *  Time instant p. In the presence of nil it returns nil
 * @param p a time instant that is compared to this time instant.
 * @return The earliest Time Instant or nil in the presence of nil.
 */
public Time minp(Time p);
```

```
/**
 *  Calculates the earliest valid Time Instant among this Time instant and Time instant p.
 *  In the presence of nil it returns a comparable time instant, if both are nil it returns nil.
 * @param p a time instant that is compared to this time instant.
 * @return The earliest Time Instant or nil in the comparison of nil's.
 */
public Time minvp(Time p);

/**
 *  Calculates the latest Time Instant among this Time instant and Time instant p.
 *  In the presence of nil it returns nil
 * @param p a time instant that is compared to this time instant.
 * @return The latest Time Instant or nil in the presence of nil.
 */
public Time maxp(Time p);

/**
 *  Calculates the latest valid Time Instant among this Time instant and Time instant p.
 *  In the presence of nil it returns a comparable time instant, if both are nil it returns nil.
 * @param p a time instant that is compared to this time instant.
 * @return The latest Time Instant or nil in the comparison of nil's.
 */
public Time maxvp(Time p);

/**
 * Examines if this Time instant is earlier than the Time Instant p.
 * @param p a time instant that is compared to this time instant.
 * @return true if this time instant is earlier than the time instant given in the parameter  else false.
 */
public boolean before(Time p);

/**
 * Examines if this Time instant is later than  the Time Instant p.
 * @param p a time instant that is compared to this time instant.
 * @return true if this time instant is later than the time instant given in the parameter  else false.
 */
public boolean after(Time p);

/**
 * Examines if the Time Instant p is the same with this Time instant.
 * @param p a time instant that is compared to this time instant.
 * @return true if this time instant is equal to the time instant given in the parameter else false.
 */
public boolean equals(Time p);

/**
 * Examines if this Time instant is earlier than or equal to the Time Instant p.
 * @param p a time instant that is compared to this time instant.
 * @return true if this time instant is earlier or equal than the time instant given in the parameter
else false.
 */
public boolean before_equals(Time p);

/**
 * Examines if this Time instant is later than or equal to the Time Instant p.
 * @param p a time instant that is compared to this time instant.
```

```
 * @return true if this time instant is later than or equal the time instant given in the parameter else
false.
 */
public boolean after_equals(Time p);

/**
 * Assigns an integer with the value of the duration of this Time Instant in the finest granularity.
 * @return an integer expressing the duration of this instant in terms of the finest granularity.
 */
public int duration();

/**
 * Calculates an integer with the value of the duration between this Time Instant and
 * the time instant p given in the parameter, in the finest granularity.
 * @param p a time instant.
 * @return an integer expressing the duration of the time gap between this
 * and the time instant given in the parameter.
 */
public int difference(Time p);

/**
 * Returns this time instant.
 * @return this time instant.
 */
public Time timepoint();
/**
 * Assigns a String representation of this time instant
 * @return a String representing this time instant.
 */
public abstract String repString();
}
```

The positive infinite timepoint implementation

```
/**
 * Implements the special character time point positive infinite.
 */
public class Pinf implements Time {

/**
 * Calculates the previous Time Instant related to the positive infinite timepoint
 * @return positive infinite
 */
public Time previous() {
        return this;
}

/**
 * Calculates the next Time Instant related to the positive infinite timepoint
 * @return positive infinite
 */
public Time next() {
        return this;
}

/**
 * Calculates the earliest Time Instant among positive infinite and Time instant p.
 * @param p a time instant that is compared positive infinite.
```

```
 * @return p.
 */
public Time minp(Time p) {
	return p;
}

/**
 * Calculates the earliest valid Time Instant among positive infinite and Time instant p.
 * @param p a time instant that is compared positive infinite.
 * @return positive infinite if p is nil else p.
 */
public Time minvp(Time p) {
	if(p instanceof Nil) return this;
		return p;
}

/**
 *  Calculates the latest Time Instant among the p Time instant and positive infinite timepoint.
 * @param p a time instant that is compared to positive infinite.
 * @return positive infinite or if p is Nil return Nil.
 */
public Time maxp(Time p) {
	if(p instanceof Nil) return p;
	return this;
}

/**
 *  Calculates the latest Time Instant among the p Time instant and positive infinite timepoint.
 * @param p a time instant that is compared to this time instant.
 * @return positive infinite.
 */
public Time maxvp(Time p) {
	return this;
}

/**
 * Examines if positive infinite is earlier than the Time Instant p.
 * @param p a time instant that is compared to positive infinite time instant.
 * @return false for every time instant except the positive infinite time instant.
 */
public boolean before(Time p) {
	if(p instanceof Pinf) return true;
	return false;
}

/**
 * Examines if positive infinite is later than the Time Instant p .
 * @param p a time instant that is compared to positive infinite time instant.
 * @return true.
 */
public boolean after(Time p) {
	return true;
}

/**
 * Examines if positive infinite is equal to  the Time Instant p.
 * @param p a time instant that is compared to positive infinite time instant.
```

```
 * @return true if p is positive infinite else false.
 */
public boolean equals(Time p) {
	if(p instanceof Pinf) return true;
	return false;
}

/**
 * Examines if positive infinite is earlier than or equal the Time Instant p.
 * @param p a time instant that is compared to positive infinite time instant.
 * @return true if p is the positive infinite time instant else false.
 */
public boolean before_equals(Time p) {
	return (this.before(p)||this.equals(p));
}

/**
 * Examines if the positive infinite is later than or equal to the Time Instant p.
 * @param p a time instant that is compared to positive infinite time instant.
 * @return true.
 */
public boolean after_equals(Time p) {
	return true;
}

/**
 * Assigns an -1 as a representation of the duration of positive infinite timepoint.
 * @return -1.
 */
public int duration() {
	return -1; // Should have returned INF
}
/**
 * The duration between positive infinite and the time instant p given in the parameter
 * is infinite and returns -1.
 * @param p a time instant
 * @return -1
 */
public int difference(Time p) {
	return -1;
}
public Time timepoint() {
	return this;
}
/**
 * Assigns a String representation of positive infinite timepoint
 * @return the String "+inf".
 */
public String repString() {
	String s ="+inf";
	return s;
}
}
```

The negative infinite timepoint implementation

```
/**
 * Implements the special character time point negative infinite.
 */
```

```
public class Ninf implements Time{

/**
 * Calculates the previous Time Instant related to the negative infinite timepoint
 * @return negative infinite
 */
public Time previous() {
	return this;
}

/**
 * Calculates the next Time Instant related to the negative infinite timepoint
 * @return negative infinite
 */
public Time next() {
	return this;
}

/**
 * Calculates the earliest Time Instant among the p Time instant and negative infinite timepoint.
 * @param p a time instant that is compared to negative infinite.
 * @return negative infinite or if p is Nil return Nil.
 */
public Time minp(Time p) {
	if(p instanceof Nil) return p;
	return this;
}

/**
 * Calculates the earliest Time Instant among the p Time instant and negative infinite timepoint.
 * @param p a time instant that is compared to negative infinite.
 * @return negative infinite.
 */
public Time minvp(Time p) {
	return this;
}

/**
 *  Calculates the latest Time Instant among the p Time instant and negative infinite timepoint.
 * @param p a time instant that is compared to negative infinite timepoint.
 * @return p.
 */
public Time maxp(Time p) {
	return p;
}

/**
 *  Calculates the latest valid Time Instant among the p Time instant and negative infinite timepoint.
 * @param p a time instant that is compared to negative infinite timepoint.
 * @return negative infinite if p is nil else p.
 */
public Time maxvp(Time p) {
	if(p instanceof Nil) return this;
	return p;
}

/**
```

```
 * Examines if negative infinite is earlier than  the Time Instant p .
 * @param p a time instant that is compared to negative infinite time instant.
 * @return true.
 */
public boolean before(Time p) {
	return true;
}

/**
 * Examines if the negative infinite is later than the Time Instant p.
 * @param p a time instant that is compared to negative infinite time instant.
 * @return true if p is negative infinite else false.
 */
public boolean after(Time p) {
	if(p instanceof Ninf) return true;
	return false;
}

/**
 * Examines if negative infinite is equal to the Time Instant p.
 * @param p a time instant that is compared to negative  infinite time instant.
 * @return true if p is negative infinite else false.
 */
public boolean equals(Time p) {
	if(p instanceof Ninf) return true;
	return false;
}

/**
 * Examines if the negative infinite is earlier than or equal to  the Time Instant p.
 * @param p a time instant that is compared to negative infinite time instant.
 * @return true.
 */
public boolean before_equals(Time p) {
	return (this.before(p)||this.equals(p));
}

/**
 * Examines if the negative infinite is later than or equal to the Time Instant p.
 * @param p a time instant that is compared to negative  infinite time instant.
 * @return true if p is negative infinite else false.
 */
public boolean after_equals(Time p) {
	return (this.after(p)||this.equals(p));
}

/**
 * Assigns an -1 as a representation of the duration of negative infinite timepoint.
 * @return -1.
 */
public int duration() {
	return -1; // representig INF
}

/**
 * The duration between negative infinite and the time instant given in the parameter
 * is infinite and returns -1.
```

```
 * @param p a time instant.
 * @return -1
 */
public int difference(Time p) {
	return -1;
}

public Time timepoint() {
	return this;
}

/**
 * Assigns a String representation of negative infinite timepoint
 * @return the String "-inf".
 */
public String repString() {
	String s ="-inf";
	return s;
}
}
```

The empty timepoint Nil implementation

```
/**
 * Implements the special character time point nil.
 */
public class Nil implements Time {

/**
 * Calculates the previous Time Instant of nil.
 * @return nil.
 */
public Time previous() {
	return new Nil();
}

/**
 * Calculates the next Time Instant of nil.
 * @return nil.
 */
public Time next() {
	return new Nil();
}

/**
 * Calculates the earliest Time Instant among nil and Time instant p.
 * @param p a time instant that is compared nil.
 * @return nil.
 */
public Time minp(Time p) {
	return this;
}

/**
 * Calculates the earliest valid Time Instant among nil and Time instant p.
 * @param p a time instant that is compared nil.
 * @return p.
 */
```

```
public Time minvp(Time p) {
		return p;
}

/**
 * Calculates the latest Time Instant among nil and Time instant p.
 * @param p a time instant that is compared nil.
 * @return nil.
 */
public Time maxp(Time p) {
		return this;
}

/**
 * Calculates the latest valid Time Instant among nil and Time instant p.
 * @param p a time instant that is compared nil.
 * @return p.
 */
public Time maxvp(Time p) {
		return p;
}

/**
 * Examines if nil is earlier than the Time Instant p.
 * @param p a time instant that is compared to nil.
 * @return false.
 */
public boolean before(Time p) {
		return false;
}

/**
 * Examines if nil is later than the Time Instant p.
 * @param p a time instant that is compared to nil.
 * @return false.
 */
public boolean after(Time p) {
		return false;
}

/**
 * Examines if the Time Instant p is equal to nil.
 * @param p a time instant that is compared to nil.
 * @return true if p is nil else false.
 */
public boolean equals(Time p) {
		if(p instanceof Nil) return true;
		return false;
}

/**
 * Examines if nil the is earlier than or equal to Time Instant p.
 * @param p a time instant that is compared to nil.
 * @return true if p is nil else false.
 */
public boolean before_equals(Time p) {
		return (this.before(p)||this.equals(p));
```

```
}

/**
 * Examines if nil the is later than or equal to Time Instant p.
 * @param p a time instant that is compared to nil.
 * @return true if p is nil else false.
 */
public boolean after_equals(Time p) {
	return (this.before(p)||this.equals(p));
}

/**
 * The integer expressing the duration of the nil time point.
 * @return 0.
 */
public int duration() {
	return 0;
}

/**
 * The integer expressing the duration gap of time instant p and the nil time point.
 * @param p a time instant.
 * @return 0.
 */
public int difference(Time p) {
	return 0;
}

public Time timepoint() {
	return this;
}

/**
 * Assigns a String representation of the Nil timepoint
 * @return the String "nil".
 */
public String repString() {
	String s ="nil";
	return s;
}
}
```

The Numeric timepoint implementation

```
public class NumericTimePoint implements Time {
/**
 * The value with reference to the time line of this timepoint
 */
private int value;

public NumericTimePoint(int p){
	this.value=p;
}

public void setValue(int p){
	this.value=p;
}

public int getValue(){
```

```
		return this.value;
	}

	public Time previous() {
		return new NumericTimePoint(this.value-1);
	}

	public Time next() {
		return new NumericTimePoint(this.value+1);
	}

	public Time minp(Time p) {
		if(p instanceof NumericTimePoint){
			NumericTimePoint p1= (NumericTimePoint)p;
			if(this.value < p1.getValue())
				return this;
			else
				return p;
		}
		else if(p instanceof Nil) return p;
		else if(p instanceof Ninf) return p;
		else if(p instanceof Pinf) return this;
		else return null;// Should never reach this point
	}

	public Time minvp(Time p) {
		if(p instanceof NumericTimePoint){
			NumericTimePoint p1= (NumericTimePoint)p;
			if(this.value < p1.getValue())
				return this;
			else
				return p;
		}
		else if(p instanceof Nil) return this;
		else if(p instanceof Ninf) return p;
		else if(p instanceof Pinf) return this;
		else return null;// Should never reach this point
	}

	public Time maxp(Time p) {
		if(p instanceof NumericTimePoint){
			NumericTimePoint p1= (NumericTimePoint)p;
			if(this.value > p1.getValue())
				return this;
			else
				return p;
		}
		else if(p instanceof Nil) return p;
		else if(p instanceof Pinf) return p;
		else if(p instanceof Ninf) return this;
		else return null;// Should never reach this point
	}

	public Time maxvp(Time p) {
		if(p instanceof NumericTimePoint){
			NumericTimePoint p1= (NumericTimePoint)p;
			if(this.value > p1.getValue())
```

```
                        return this;
                else
                        return p;
        }
        else if(p instanceof Nil) return this;
        else if(p instanceof Pinf) return p;
        else if(p instanceof Ninf) return this;
        else return null;// Should never reach this point
}

public boolean before(Time p) {
        if(p instanceof NumericTimePoint){
                NumericTimePoint p1= (NumericTimePoint)p;
                if(this.value<p1.getValue())
                        return true;
                else
                        return false;
        }
        else if(p instanceof Pinf) return true;
        else if(p instanceof Ninf) return false;
        else return false;// Should have returned UNDEFINED if( p instanceof nil)
}

public boolean after(Time p) {
        if(p instanceof NumericTimePoint){
                NumericTimePoint p1= (NumericTimePoint)p;
                if(this.value>p1.getValue())
                        return true;
                else
                        return false;
        }
        else if(p instanceof Pinf) return false;
        else if(p instanceof Ninf) return true;
        else return false;
}

public boolean equals(Time p) {
        if(p instanceof NumericTimePoint)
                if(((NumericTimePoint) p).getValue()==this.value)
                return true;
        return false;
}

public boolean before_equals(Time p) {
        return (this.before(p)||this.equals(p));
}

public boolean after_equals(Time p) {
        return (this.after(p)||this.equals(p));
}

public int duration() {
        return 1;
}

public int difference(Time p) {
        if(p instanceof NumericTimePoint){
```

```
                NumericTimePoint p1= (NumericTimePoint)p;
                if(this.value>p1.getValue())
                        return this.value-p1.getValue();
                else
                        return p1.getValue()-this.value;
        }
        else if(p instanceof Pinf) return -1; // Should never reach
        else if(p instanceof Ninf) return -1; // Should never reach
        else return 0; // Should never reach
}

public Time timepoint() {
        return this;
}

public String repString() {
        String s =""+this.value+"";
        return s;
}
}
```

The ConvexIntervals package:

```
/**
 * Allen's Interval Relationships
 */
public interface Allen {
        public boolean isBefore(CIntervals I);
        public boolean isAfter(CIntervals I);
        public boolean isMeeting(CIntervals I);
        public boolean isMetBy(CIntervals I);
        public boolean isOverlapping(CIntervals I);
        public boolean isOverlappedBy(CIntervals I);
        public boolean isDuring(CIntervals I);
        public boolean isIncluding(CIntervals I);
        public boolean isStarting(CIntervals I);
        public boolean isStartedBy(CIntervals I);
        public boolean isFinishing(CIntervals I);
        public boolean isFinishedBy(CIntervals I);
        public boolean isEqual(CIntervals I);
}

/**
 * Allen's Potential Interval Relationships
 */
public interface PotAllen extends Allen {

        public boolean isPotBefore(CIntervals I);
        public boolean isPotAfter(CIntervals I);
        public boolean isPotMeeting(CIntervals I);
        public boolean isPotMetBy(CIntervals I);
        public boolean isPotOverlapping(CIntervals I);
        public boolean isPotOverlappedBy(CIntervals I);
        public boolean isPotDuring(CIntervals I);
        public boolean isPotIncluding(CIntervals I);
        public boolean isPotStarting(CIntervals I);
        public boolean isPotStartedBy(CIntervals I);
```

```
	public boolean isPotFinishing(CIntervals I);
	public boolean isPotFinishedBy(CIntervals I);
	public boolean isPotEqual(CIntervals I);
}

import java.util.LinkedList;
import times.Time;
public interface CIntervals extends Allen{
	public boolean isAdjacent(CIntervals I);
	public boolean isIntersecting(CIntervals I);
	public boolean isDisjoint(CIntervals I);
	public boolean isMergeable(CIntervals I);
	public boolean isTimepoint();
	public boolean isEmpty();
	public boolean isLeftInfinite();
	public boolean isRightInfinite();
	public boolean isTotalInfinite();
	public boolean isIncluding(Time t);
	public boolean isContaining(CIntervals I);
	public void setStart(Time s);
	public void setEnd(Time e);
	public Time getStart();
	public Time getEnd();
	public int duration();
	public CIntervals mergeWith(CIntervals I);
	public CIntervals intersectWith(CIntervals I);
	public CIntervals highDifference(CIntervals I);
	public CIntervals lowDifference(CIntervals I);
	public LinkedList<CIntervals> union(CIntervals I);
	public LinkedList<CIntervals> intersection(CIntervals I);
	public LinkedList<CIntervals> difference(CIntervals I);
	public LinkedList<CIntervals> complement();
	public String repString();
}


import java.util.LinkedList;
import java.util.ListIterator;
import times.*;
public class ConvexInterval implements Allen, CIntervals{

private Time start;
private Time end;

public ConvexInterval(){}

public ConvexInterval(Time s, Time e){
	if(s instanceof Ninf && e instanceof Ninf){
		this.start=s;
		this.end=new Pinf();
	}
	else if(s instanceof Pinf && e instanceof Pinf){
		this.start=new Ninf();
		this.end=e;
	}
	else if(s instanceof Pinf && e instanceof Ninf){
		this.start=e;
```

```
                this.end=s;
        }
        else if(s instanceof Nil || e instanceof Nil){
                this.start=new Nil();
                this.end=new Nil();
        }
        else{
                this.start=s;
                this.end=e;
        }
}

public ConvexInterval(Time t){
        if(t instanceof Pinf){
                this.start=new Ninf();
                this.end=t;
                }
        else if(t instanceof Ninf){
                this.start=t;
                this.end=new Pinf();
                }
        else
                this.end=this.start=t;
}
/**
 * Constructs an Infinite convex Interval
 * @return An Infinite Convex Interval
 */
public ConvexInterval infinite(){
        return new ConvexInterval(new Pinf());
}

/**
 * Constructs an Empty convex Interval
 * @return An Empty Convex Interval
 */
public ConvexInterval empty(){
        return new ConvexInterval(new Nil());
}

/**
 * Set the starting Time-point of the Interval
 * @param s The Time-Point that will be the starting Time-point.
 */
public void setStart(Time s){
        this.start=s;
}

/**
 * Set the ending Time-point of the Interval
 * @param e The Time-Point that will be the ending Time-point.
 */
public void setEnd(Time e){
        this.end=e;
}
/**
 * Get the starting Time-point of the Interval
```

```
 * @return The starting Time-point of this Interval.
 */
public Time getStart(){
	return this.start;
}
/**
 * Get the ending Time-point of the Interval
 * @return The ending Time-point of this Interval.
 */
public Time getEnd(){
	return this.end;
}
/**
 * Examines the validity of interval in terms of ordering of its time-points
 * @return True for a valid constructed Interval
 */
ublic boolean valid_interval(){
	if(this.end instanceof Nil){
		if(this.start instanceof Nil)
			return true;
		else
			return false;
	}
	else
		if(this.start.before_equals(this.end))
			return true;
		else
			return false;
}

/**
 * Examines if this interval is Empty
 * @return True for an Empty Interval
 */
public boolean isEmpty(){
	if((this.end instanceof Nil) && (this.start instanceof Nil))
		return true;
	else
		return false;
}
/**
 * Examines if this Interval is Left Infinite
 * @return True for a Left Infinite Interval
 */
public boolean isLeftInfinite(){
	if((this.start instanceof Ninf) || this.isTotalInfinite() )
		return true;
	else
		return false;
}
/**
 * Examines if this Interval is Right Infinite
 * @return True for a Right infinite Interval
 */
public boolean isRightInfinite(){
	if((this.end instanceof Pinf) || this.isTotalInfinite())
		return true;
```

```
		else
			return false;
	}
	/**
	 * Examines if this Interval is Totally Infinite
	 * @return True for a Totally infinite Interval
	 */
	public boolean isTotalInfinite(){
		if((this.start instanceof Ninf) && (this.end instanceof Pinf))
			return true;
		else if((this.start instanceof Ninf) && (this.end instanceof Ninf))
			return true;  // Strange implementation
		else if((this.start instanceof Pinf) && (this.end instanceof Pinf))
			return true;  // Strange implementation
		else if((this.start instanceof Pinf) && (this.end instanceof Ninf))
			return true;  // Strange implementation
		else
			return false;
	}

	/**
	 * Examines if this Interval has an Infinite end-point
	 * @return True if this Interval has an  Infinite end-point
	 */
	private boolean isInfinite(){
		return (this.isRightInfinite() || this.isLeftInfinite() || this.isTotalInfinite());
	}
	/**
	 * Examines if this Interval Represents a Time-point
	 * @return True if this Interval is a Time - Point
	 */
	public boolean isTimepoint(){
		if(!this.end.equals(this.start) || this.isTotalInfinite())
			return false;
		else
			return true;
	}
	/**
	 * Examines if this Interval Includes Time-point p
	 * @param p The Time-Point that its inclusion in this Interval is examined.
	 * @return True if this Interval contains Time-Point.
	 */
	public boolean isIncluding(Time p){
		if(p instanceof Nil)
			return false; //THE nil timepoint is nowhere included
		else
			if(p.after_equals(this.start) && p.before_equals(this.end))
				return true;
			else
				return false;
	}
	/**
	 * Examines if this Interval Contains Interval I
	 * @param I The Interval that its inclusion in this Interval is examined.
	 * @return True if this Interval contains Interval I.
	 */
	public boolean isContaining(CIntervals I){
```

```
		if(I.isEmpty())
			return false; //THE empty Interval is nowhere included
		else
			if(I.getStart().after_equals(this.start) && I.getEnd().before_equals(this.end))
				return true;
			else
				return false;
}
/**
 * Examines if interval I given as parameter is disjoint to this interval
 * @param I The interval that its disjointness is to be checked.
 * @return True if I is disjoint to this interval, else false.
 */
public boolean isDisjoint(CIntervals I){
		if(oneInfinite(I))
			return false;
		else
			return this.isBefore(I)||this.isAfter(I);
}
/**
 * Examines if interval I given as parameter is intersecting this interval
 * @param I The interval that its intersection to this Interval is examined.
 * @return True if I is intersecting this interval, else false.
 */
public boolean isIntersecting(CIntervals I){
		if(!(this.isDisjoint(I)||this.isTotalInfinite()||this.isEmpty()||I.isEmpty()))
			return true;
		else
			return false;
}
/**
 * Examines if this interval is adjacent to interval I
 * @param I The interval its adjacency to this Interval is examined.
 * @return True if I is adjacent to this interval, else false.
 */
public boolean isAdjacent(CIntervals I){
		boolean c1 = false;
		if(this.isBefore(I)){
			c1=(this.end.next().equals(I.getStart()));
		}
		else if(this.isAfter(I)){
			c1=this.start.previous().equals(I.getEnd());
		}
		return c1;
}
/**
 * Examines if this interval has common time-points with interval I,
 *  therefore they are Merge-able.
 * @param I The interval its merge-ability to this Interval is examined.
 * @return True if I has common time-points to this interval, else false.
 */
public boolean isMergeable(CIntervals I){
		return (this.isIntersecting(I) || this.isAdjacent(I) || this.isEmpty() || I.isEmpty());
}
/**
 * Constructs a convex Interval that is a coalesced version of this Interval
 * and the Interval I. If the Convex intervals are not merge-able then
```

```
 * the empty interval is returned.
 * @param I The Interval that is to be coalesced with this interval.
 * @return The Coalesced Convex Interval.
 */
public CIntervals mergeWith(CIntervals I){
	CIntervals res;
	if(!this.isMergeable(I))
		res = this.empty();
	else
		res= new ConvexInterval(this.start.minvp(I.getStart()), this.end.maxvp(I.getEnd()));
	return res;
}
/**
 * Constructs a convex Interval that is the intersecting region of this Interval,
 * with the Interval I.  Constructs and returns the empty interval if they do not intersect
 * or if one of them is empty.
 * @param I The convex interval that intersects with this Interval.
 * @return The convex Intersection Interval.
 */
public CIntervals intersectWith(CIntervals I){
	CIntervals res;
	if( bothInfinite(I) )
		res= this.infinite();
	else if(!this.isIntersecting(I))
		res= this.empty();
	else
		res= new ConvexInterval(this.start.maxp(I.getStart()), this.end.minp(I.getEnd()));
	return res;
}
/**
 * Constructs the integer that represents the duration of this convex Interval
 * @return an Integer representing the convex Interval duration.
 * 0 for an empty interval and
 * -1 for infinite.
 */
public int duration(){
	if(this.isEmpty()) return 0;
	else if(this.isInfinite()) return -1;
	else return this.end.difference(start)+1;
}
/**
 * Constructs the Convex Interval Union of this Interval and Interval I
 * as a NonConvexInterval.
 * @param I The convex Interval that Unifies With this Interval
 * @return the Union as a LinkedList of CIntervals.
 */
public LinkedList<CIntervals> union(CIntervals I){
	LinkedList<CIntervals> res=new LinkedList<CIntervals>();
	if(this.isMergeable(I)){
		CIntervals merge=this.mergeWith(I);
		if(!merge.isEmpty())
			res.add(this.mergeWith(I));
	}
	else
		if(this.isBefore(I)){
			res.add(this);
			res.add(I);
```

```
                }
                else{
                        res.add(I);
                        res.add(this);
                }
        return res;
}
/**
 * Constructs the Convex Interval Intersection of this Interval and Interval I
 * as a NonConvexInterval.
 * @param I The convex Interval that Intersects With this Interval
 * @return the Intersection as a NonConvexInterval
 */
public LinkedList<CIntervals> intersection(CIntervals I){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isIntersecting(I)){
                if(bothInfinite(I)){
                        res.add(this.infinite());
                }
                else if(this.isIncluding(I)){
                        res.add(I);
                }
                else if(this.isDuring(I)){
                        res.add(this);
                }
                else{
                        CIntervals low=new
ConvexInterval(this.start.maxp(I.getStart()),this.end.minp(I.getEnd()));
                        res.add(low);
                }
        }
        return res;
}
/**
 * Constructs the Non Convex Interval Complement of this Interval
 * @return the Complement as a NonConvexInterval
 */
public LinkedList<CIntervals> complement(){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isTotalInfinite()){
                res.add(this.empty());
        }
        else if(this.isLeftInfinite()){
                res.add(new ConvexInterval(this.end.next(),new Pinf()));
        }
        else if(this.isRightInfinite()){
                res.add(new ConvexInterval(new Ninf(),this.start.previous()));
        }
        else{
                res.add(new ConvexInterval(new Ninf(),this.start.previous()));
                res.add(new ConvexInterval(this.end.next(),new Pinf()));
        }
        return res;
}
public CIntervals lowDifference(CIntervals I){
        CIntervals low;
        if(this.isDisjoint(I) || I.isEmpty()){
```

```
                low= this;
        }
        else if(this.isOverlapping(I)||this.isMeeting(I)|| this.isFinishedBy(I)){
                low=new ConvexInterval(this.start,I.getStart().previous());
                }
        else if(this.isIncluding(I)){
                low=new ConvexInterval(this.start,I.getStart().previous());
                }
        else
                low=empty();
        return low;
}
public CIntervals highDifference(CIntervals I){
        CIntervals high;
        if(this.isDisjoint(I) || I.isEmpty()){
                high= this;
        }
        else if(this.isOverlappedBy(I)||this.isMetBy(I)|| this.isStartedBy(I)){
                high=new ConvexInterval(I.getEnd().next(),this.end);
                }
        else if(this.isIncluding(I)){
                high=new ConvexInterval(I.getEnd().next(),this.end);
                }
        else
                high=empty();
        return high;
}
/**
 * Constructs the Non Convex Interval Difference of this convex Interval minus
 * the convex Interval I.
 * @param I The convex Interval that is subtracted from this Interval
 * @return the Difference as a NonConvexInterval
 */
public LinkedList<CIntervals> difference(CIntervals I){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isDisjoint(I) || I.isEmpty()){
                res.add(this);
        }
        else{
                CIntervals low, high;
                if(I.isTotalInfinite() || bothInfinite(I) || this.isDuring(I)|| this.isStarting(I)||
this.isFinishing(I)|| this.isEqual(I)){
                        return res;
                        }
                else if(this.isOverlapping(I)||this.isMeeting(I)|| this.isFinishedBy(I)){
                                low=new ConvexInterval(this.start,I.getStart().previous());
                                res.add(low);
                                }
                else if(this.isIncluding(I)){
                                low=new ConvexInterval(this.start,I.getStart().previous());
                                high=new ConvexInterval(I.getEnd().next(),this.end);
                                res.add(low);
                                res.add(high);
                                }
                        else {
                        high=new ConvexInterval(I.getEnd().next(),this.end);
                        res.add(high);
```

```
                                }
        }
        return res;
}
/**
 * Constructs a string representation of this Interval
 * in the form of < starting end-point, ending end-point>.
 * @returns a pictorial string representation.
 */
public String repString(){
        String s="<"+this.start.repString()+","+this.end.repString()+">";
        return s;
}

public String repString(LinkedList<CIntervals> I){
        String s;
        if(I.isEmpty()){
                s="{ }";
        }
        else{
                ListIterator<CIntervals> it=I.listIterator();
                s="{ ";
                while(it.hasNext()){
                        CIntervals n=it.next();
                        s=s.concat(n.repString());
                }
                s=s.concat(" } ");
        }
        return s;
}

/* ALLEN */
private boolean oneEmpty(CIntervals I){
        return (this.isEmpty() || I.isEmpty());
}
private boolean bothEmpty(CIntervals I){
        return (this.isEmpty() && I.isEmpty());
}
private boolean bothInfinite(CIntervals I){
        return this.isTotalInfinite() && I.isTotalInfinite();
}
private boolean oneInfinite(CIntervals I){
        return this.isTotalInfinite() || I.isTotalInfinite();
}
public boolean isBefore(CIntervals I) {
        if(oneEmpty(I) || oneInfinite(I)) return false;
        boolean c1=this.start.before(I.getStart());
        boolean c2=this.start.before(I.getEnd());
        boolean c3=this.end.before(I.getStart());
        boolean c4=this.end.before(I.getEnd());
        return (c1 && c2 && c3 && c4 );
}
public boolean isAfter(CIntervals I) {
        if(oneEmpty(I) || oneInfinite(I)) return false;
        boolean c1=I.getStart().before(this.getStart());
        boolean c2=I.getStart().before(this.getEnd());
        boolean c3=I.getEnd().before(this.getStart());
```

```
            boolean c4=I.getEnd().before(this.getEnd());
            return (c1 && c2 && c3 && c4 );
    }
    public boolean isMeeting(CIntervals I) {
            if(oneEmpty(I) || bothInfinite(I)) return false;
            boolean c1=this.start.before(I.getStart());
            boolean c2=this.start.before(I.getEnd());
            boolean c3=this.end.equals(I.getStart());
            boolean c4=this.end.before(I.getEnd());
            return (c1 && c2 && c3 && c4 );
    }
    public boolean isMetBy(CIntervals I) {
            if(oneEmpty(I) || bothInfinite(I)) return false;
            boolean c1=I.getStart().before(this.getStart());
            boolean c2=I.getStart().before(this.getEnd());
            boolean c3=I.getEnd().equals(this.getStart());
            boolean c4=I.getEnd().before(this.getEnd());
            return (c1 && c2 && c3 && c4 );
    }
    public boolean isOverlapping(CIntervals I) {
            if(oneEmpty(I) || bothInfinite(I)) return false;
            boolean c1=this.start.before(I.getStart());
            boolean c2=this.start.before(I.getEnd());
            boolean c3=this.end.after(I.getStart());
            boolean c4=this.end.before(I.getEnd());
            return (c1 && c2 && c3 && c4 );
    }
    public boolean isOverlappedBy(CIntervals I) {
            if(oneEmpty(I) || bothInfinite(I)) return false;
            boolean c1=I.getStart().before(this.getStart());
            boolean c2=I.getStart().before(this.getEnd());
            boolean c3=I.getEnd().after(this.getStart());
            boolean c4=I.getEnd().before(this.getEnd());
            return (c1 && c2 && c3 && c4 );
    }
    public boolean isDuring(CIntervals I) {
            if(oneEmpty(I) || bothInfinite(I) ) return false;
            boolean c1=this.start.after(I.getStart());
            boolean c2=this.start.before(I.getEnd());
            boolean c3=this.end.after(I.getStart());
            boolean c4=this.end.before(I.getEnd());
            return (c1 && c2 && c3 && c4 );
    }
    public boolean isIncluding(CIntervals I) {
            if(oneEmpty(I) || bothInfinite(I) ) return false;
            boolean c1=I.getStart().after(this.getStart());
            boolean c2=I.getStart().before(this.getEnd());
            boolean c3=I.getEnd().after(this.getStart());
            boolean c4=I.getEnd().before(this.getEnd());
            return (c1 && c2 && c3 && c4 );
    }
    public boolean isStarting(CIntervals I) {
            if(oneEmpty(I) || bothInfinite(I)) return false;
            boolean c1=this.start.equals(I.getStart());
            boolean c2=this.start.before(I.getEnd());
            boolean c3=this.end.after(I.getStart());
            boolean c4=this.end.before(I.getEnd());
```

```
		return (c1 && c2 && c3 && c4 );
	}
	public boolean isStartedBy(CIntervals I) {
		if(oneEmpty(I) || bothInfinite(I)) return false;
		boolean c1=I.getStart().equals(this.getStart());
		boolean c2=I.getStart().before(this.getEnd());
		boolean c3=I.getEnd().after(this.getStart());
		boolean c4=I.getEnd().before(this.getEnd());
		return (c1 && c2 && c3 && c4 );
	}
	public boolean isFinishing(CIntervals I) {
		if(oneEmpty(I) || bothInfinite(I)) return false;
		boolean c1=this.start.after(I.getStart());
		boolean c2=this.start.before(I.getEnd());
		boolean c3=this.end.after(I.getStart());
		boolean c4=this.end.equals(I.getEnd());
		return (c1 && c2 && c3 && c4 );
	}
	public boolean isFinishedBy(CIntervals I) {
		if(oneEmpty(I) || bothInfinite(I)) return false;
		boolean c1=I.getStart().after(this.getStart());
		boolean c2=I.getStart().before(this.getEnd());
		boolean c3=I.getEnd().after(this.getStart());
		boolean c4=I.getEnd().equals(this.getEnd());
		return (c1 && c2 && c3 && c4 );
	}
	public boolean isEqual(CIntervals I) {
		if(oneEmpty(I)) return false;
		if(bothEmpty(I)) return true;
		boolean c1=I.getStart().equals(this.getStart());
		boolean c2=I.getStart().before(this.getEnd());
		boolean c3=I.getEnd().after(this.getStart());
		boolean c4=I.getEnd().equals(this.getEnd());
		return (c1 && c2 && c3 && c4 );
	}
	}

	import java.util.LinkedList;
	import times.*;
	public class DeterminateInterval extends ConvexInterval implements Allen, CIntervals, PotAllen{

	public DeterminateInterval(Time s, Time e){
		super(s,e);
	}

	public DeterminateInterval(Time t){
		super(t);
	}

	public DeterminateInterval(CIntervals I){
		super(I.getStart(),I.getEnd());
	}

	public DeterminateInterval infinite(){
		return new DeterminateInterval(new Pinf());
	}
```

```
public DeterminateInterval empty(){
	return new DeterminateInterval(new Nil());
}

public boolean isMergeable(CIntervals I){
	boolean res=false;
	if((I instanceof IndeterminateInterval) && this.isAdjacent(I))
		res=false;
	else if(this.isIntersecting(I)||  this.isAdjacent(I) || this.isEmpty() || I.isEmpty())
		res=true;
	return res;
}

public CIntervals mergeWith(DeterminateInterval I){
	return new DeterminateInterval(super.mergeWith(I));
}

public CIntervals intersectWith(CIntervals I){
	if(I instanceof DeterminateInterval)
		return new DeterminateInterval(super.intersectWith(I));
	else
		return I.intersectWith(this);
}

public LinkedList<CIntervals> union(CIntervals I){
	LinkedList<CIntervals> res;
	if(!this.isMergeable(I)){
		res=this.NonMergeableUnion(I);
	}
	else{
		res=new LinkedList<CIntervals>();
		if(this.isEmpty()){
			if(I.isEmpty())
				return res;
			else
				res.add(I);
		}
		else if(I.isEmpty()){
			res.add(this);
		}
		else if(I instanceof DeterminateInterval)
			res.add(this.mergeWith(I));
		else {
			CIntervals ld=I.lowDifference(this);
			CIntervals hd=I.highDifference(this);
			if(ld.isEmpty() && hd.isEmpty()){
				res.add(I);
				}
			else if( hd.isEmpty()){
				res.add(ld);
				res.add(this);
				}
			else{
				res.add(I);
				res.add(this);
			}
		}
```

```
		}
		return res;
}

private LinkedList<CIntervals> NonMergeableUnion(CIntervals I){
		LinkedList<CIntervals> res=new LinkedList<CIntervals>();
		if(this.isBefore(I)){
			res.add(this);
			res.add(I);
			}
		else{
			res.add(I);
			res.add(this);
			}
		return res;
}

public LinkedList<CIntervals> intersection(CIntervals I){
		LinkedList<CIntervals> res=new LinkedList<CIntervals>();
		CIntervals ci;
		if(I instanceof DeterminateInterval)
			ci=this.intersectWith(I);
		else
			ci=I.intersectWith(this);
		if(ci.isEmpty())
			return res;
		else
			res.add(ci);
		return res;
}

public CIntervals lowDifference(CIntervals I){
		return new DeterminateInterval(super.lowDifference(I));
}

public CIntervals highDifference(CIntervals I){
		return new DeterminateInterval(super.highDifference(I));
}

public LinkedList<CIntervals> difference(CIntervals I){
		LinkedList<CIntervals> res=new LinkedList<CIntervals>();
		if(this.isDisjoint(I) || I.isEmpty()){
			res.add(this);
			}
		else{
			CIntervals low, mid, high;
			low=this.lowDifference(I);
			high=this.highDifference(I);
			if(!low.isEmpty())
				res.add(low);
			if(I instanceof IndeterminateInterval){
				mid=this.lowDifference(I);
				if(!mid.isEmpty())
					res.add(mid);
			}
			if(!high.isEmpty())
				res.add(high);
```

```
                }
        return res;
}
/**
 * Constructs the Non Convex Interval Complement of this Interval
 * @return the Complement as a NonConvexInterval
 */
public LinkedList<CIntervals> complement(){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isTotalInfinite()){
                return res;
        }
        else if(this.isLeftInfinite()){
                res.add(new DeterminateInterval(this.getEnd().next(),new Pinf()));
        }
        else if(this.isRightInfinite()){
                res.add(new DeterminateInterval(new Ninf(),this.getStart().previous()));
        }
        else{
                res.add(new DeterminateInterval(new Ninf(),this.getStart().previous()));
                res.add(new DeterminateInterval(this.getEnd().next(),new Pinf()));
        }
        return res;
}

public String repString(){
        String s=" Det"+super.repString();;
        return s;
}
// POTENTIAL

public boolean isPotBefore(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval){
                e1 = this.getEnd().before(I.getEnd());
                }
        else{
                e1 = this.isBefore(I);
                }
        return e1;
}

public boolean isPotAfter(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval){
                e1 = this.getStart().after(I.getStart());
                }
        else{
                e1 = this.isAfter(I);
                }
        return e1;
}

public boolean isPotMeeting(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval) {
```

```
                    e1 = this.getEnd().before_equals(I.getEnd()) &&
this.getEnd().after_equals(I.getStart()) ;
                    }
          else{
                    e1 = this.isMeeting(I);
          }
          return e1;
}

public boolean isPotMetBy(CIntervals I) {
          boolean e1;
          if(I instanceof IndeterminateInterval) {
                    e1 = this.getStart().before_equals(I.getEnd()) &&
this.getStart().after_equals(I.getStart()) ;
                    }
          else{
                    e1 = this.isMetBy(I);
                    }
          return e1;
}

public boolean isPotOverlapping(CIntervals I) {
          boolean e1;
          if(I instanceof IndeterminateInterval) {
                    e1 = this.getEnd().before(I.getEnd()) && this.getEnd().after(I.getStart()) ;
          }
          else{
                    e1 = this.isOverlapping(I);
          }
          return e1;
}

public boolean isPotOverlappedBy(CIntervals I) {
          boolean e1;
          if(I instanceof IndeterminateInterval) {
                    e1 = this.getStart().before(I.getEnd()) && this.getStart().after(I.getStart()) ;
          }
          else{
                    e1 = this.isOverlappedBy(I);
          }
          return e1;
}

public boolean isPotDuring(CIntervals I) {
          boolean e1;
          if(I instanceof IndeterminateInterval) {
                    e1 = this.getStart().after(I.getStart()) && this.getEnd().before(I.getEnd());
          }
          else{
                    e1 = this.isOverlappedBy(I);
          }
          return e1;
}

public boolean isPotIncluding(CIntervals I) {
          boolean e1;
          if(I instanceof IndeterminateInterval) {
```

```
                e1 = this.getStart().before(I.getEnd()) && this.getEnd().after(I.getStart());
                }
        else{
                e1 = this.isIncluding(I);
        }
        return e1;
}

public boolean isPotStarting(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval) {
                e1 = this.getStart().after_equals(I.getStart()) && this.getEnd().before(I.getEnd());
                }
        else{
                e1 = this.isStarting(I);
                }
        return e1;
}

public boolean isPotStartedBy(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval) {
                e1 = this.getStart().after_equals(I.getStart()) && this.getStart().before(I.getEnd());
                }
        else{
                e1 = this.isStartedBy(I);
                }
        return e1;
}

public boolean isPotFinishing(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval) {
                e1 = this.getStart().after(I.getStart()) && this.getEnd().before_equals(I.getEnd());
        }
        else{
                e1 = this.isFinishing(I);
        }
        return e1;
}

public boolean isPotFinishedBy(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval) {
                e1 = this.getEnd().after(I.getStart()) && this.getEnd().before_equals(I.getEnd());
                }
        else{
                e1 = this.isFinishedBy(I);
        }
        return e1;
}

public boolean isPotEqual(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval) {
                e1 = this.getStart().after_equals(I.getStart()) &&
this.getEnd().before_equals(I.getEnd());
```

```
                }
        else{
                e1 = this.isEqual(I);
        }
        return e1;
}
}

import java.util.LinkedList;
import times.*;
public class IndeterminateInterval extends ConvexInterval implements Allen, CIntervals{

public IndeterminateInterval(Time s, Time e){
        super(s,e);
}

public IndeterminateInterval(Time t){
        super(t);
}

public IndeterminateInterval(CIntervals I){
        super(I.getStart(),I.getEnd());
}

public IndeterminateInterval infinite(){
        return new IndeterminateInterval(new Pinf());
}

public IndeterminateInterval empty(){
        return new IndeterminateInterval(new Nil());
}

public CIntervals mergeWith(IndeterminateInterval I){
                return new IndeterminateInterval(super.mergeWith(I));
}

public CIntervals intersectWith(CIntervals I){
        return new IndeterminateInterval(super.intersectWith(I));
}

public boolean isMergeable(CIntervals I){
        boolean res=false;
        if((I instanceof DeterminateInterval) && this.isAdjacent(I))
                res=false;
        else if(this.isIntersecting(I)||  this.isAdjacent(I) || this.isEmpty() || I.isEmpty())
                res=true;
        return res;
}

public LinkedList<CIntervals> union(CIntervals I){
        LinkedList<CIntervals> res;
        if(!this.isMergeable(I)){
                res=this.NonMergeableUnion(I);
        }
        else{
                res=new LinkedList<CIntervals>();
                if(this.isEmpty()){
```

```
                        if(I.isEmpty())
                                return res;
                        else
                                res.add(I);
                }
                else if(I.isEmpty()){
                        res.add(this);
                }
                else if(I instanceof IndeterminateInterval)
                        res.add(this.mergeWith(I));
                else{
                        CIntervals ld=this.lowDifference(I);
                        CIntervals hd=this.highDifference(I);
                        if(ld.isEmpty() && hd.isEmpty()){
                                res.add(I);
                                }
                        else if( hd.isEmpty()){
                                res.add(ld);
                                res.add(I);
                                }
                        else{
                                res.add(I);
                                res.add(hd);
                        }
                }
        }
        return res;
}

private LinkedList<CIntervals> NonMergeableUnion(CIntervals I){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isBefore(I)){
                res.add(this);
                res.add(I);
                }
        else{
                res.add(I);
                res.add(this);
                }
        return res;
}

public LinkedList<CIntervals> intersection(CIntervals I){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isIntersecting(I)){
                if(this.isTotalInfinite() && I.isTotalInfinite()){
                        res.add(this.infinite());
                }
                else if(this.isIncluding(I)){
                        res.add(new IndeterminateInterval(I.getStart(),I.getEnd()));
                }
                else if(this.isDuring(I)){
                        res.add(this);
                }
                else{
                        CIntervals low=new
IndeterminateInterval(this.getStart().maxvp(I.getStart()),this.getEnd().minvp(I.getEnd()));
```

```
                        res.add(low);
                }
        }
        return res;
}

public CIntervals lowDifference(CIntervals I){
        return new IndeterminateInterval(super.lowDifference(I));
}

public CIntervals highDifference(CIntervals I){
        return new IndeterminateInterval(super.highDifference(I));
}

public LinkedList<CIntervals> difference(CIntervals I){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isEmpty())
                return res;
        else if(this.isDisjoint(I) || I.isEmpty()){
                res.add(this);
        }
        else{
                CIntervals low=this.lowDifference(I);
                CIntervals high=this.highDifference(I);
                if(!low.isEmpty())
                        res.add(low);
                if(!high.isEmpty())
                        res.add(high);
        }
        return res;
}

/**
 * Constructs the Non Convex Interval Complement of this Interval
 * @return the Complement as a NonConvexInterval
 */
public LinkedList<CIntervals> complement(){
        LinkedList<CIntervals> res=new LinkedList<CIntervals>();
        if(this.isTotalInfinite()){
                res.add(this);
        }
        else if(this.isLeftInfinite()){
                res.add(this);
                res.add(new DeterminateInterval(this.getEnd().next(),new Pinf()));
        }
        else if(this.isRightInfinite()){
                res.add(new DeterminateInterval(new Ninf(),this.getStart().previous()));
                res.add(this);
        }
        else{
                res.add(new DeterminateInterval(new Ninf(),this.getStart().previous()));
                res.add(this);
                res.add(new DeterminateInterval(this.getEnd().next(),new Pinf()));
        }
        return res;
}
```

```
public String repString(){
	String s=" Ind"+super.repString();
	return s;
}
// POTENTIAL

public boolean isPotBefore(CIntervals I) {
	boolean e1;
	if(I instanceof IndeterminateInterval)
		e1 = this.getStart().before(I.getEnd());
	else if( I instanceof DeterminateInterval )
		e1 = this.getStart().before(I.getStart());
	else
		e1 = this.isBefore(I);
	return e1;
}

public boolean isPotAfter(CIntervals I) {
	boolean e1;
	if(I instanceof IndeterminateInterval)
		e1 = this.getStart().after(I.getEnd());
	else if( I instanceof DeterminateInterval )
		e1 = this.getEnd().after(I.getEnd());
	else
		e1 = this.isAfter(I);
	return e1;
}

public boolean isPotMeeting(CIntervals I) {
	boolean e1;
	if(I instanceof IndeterminateInterval)
		e1 = this.getEnd().after_equals(I.getStart()) && this.getStart().before(I.getEnd());
	else if( I instanceof DeterminateInterval )
		e1 = this.getStart().before_equals(I.getStart()) &&
this.getEnd().after_equals(I.getStart());
	else
		e1 = this.isMeeting(I);
	return e1;
}

public boolean isPotMetBy(CIntervals I) {
	boolean e1;
	if(I instanceof IndeterminateInterval)
		e1 = this.getEnd().after(I.getStart()) && this.getStart().before_equals(I.getEnd());
	else if( I instanceof DeterminateInterval )
		e1 = this.getStart().before_equals(I.getEnd()) &&
this.getEnd().after_equals(I.getEnd());
	else
		e1 = this.isMetBy(I);
	return e1;
}

public boolean isPotOverlapping(CIntervals I) {
	boolean e1;
	if(I instanceof IndeterminateInterval)
		e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
	else if((I instanceof DeterminateInterval))
```

```
                e1 = this.getStart().before(I.getStart()) && this.getEnd().after(I.getStart());
        else
                e1 = this.isOverlapping(I);
        return e1;
}

public boolean isPotOverlappedBy(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before(I.getEnd()) && this.getEnd().after(I.getEnd());
        else
                e1 = this.isOverlappedBy(I);
        return e1;
}

public boolean isPotDuring(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before(I.getEnd()) && this.getEnd().after(I.getStart());
        else
                e1 = this.isDuring(I);
        return e1;
}

public boolean isPotIncluding(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before(I.getStart()) && this.getEnd().after(I.getEnd());
        else
                e1 = this.isIncluding(I);
        return e1;
}

public boolean isPotStarting(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before_equals(I.getStart()) && this.getEnd().after(I.getStart());
        else
                e1 = this.isStarting(I);
        return e1;
}

public boolean isPotStartedBy(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before_equals(I.getStart()) && this.getEnd().after(I.getStart());
        else
```

```
                e1 = this.isStartedBy(I);
        return e1;
}

public boolean isPotFinishing(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before(I.getEnd()) && this.getEnd().after_equals(I.getEnd());
        else
                e1 = this.isFinishing(I);
        return e1;
}

public boolean isPotFinishedBy(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before(I.getStart()) && this.getEnd().after_equals(I.getEnd());
        else
                e1 = this.isFinishedBy(I);
        return e1;
}

public boolean isPotEqual(CIntervals I) {
        boolean e1;
        if(I instanceof IndeterminateInterval)
                e1 = this.getEnd().after(I.getStart()) && this.getStart().before(I.getEnd());
        else if( I instanceof DeterminateInterval )
                e1 = this.getStart().before_equals(I.getStart()) &&
this.getEnd().after_equals(I.getEnd());
        else
                e1 = this.isEqual(I);
        return e1;
}
}
```

The Non Convex Intervals package:

```
import java.util.LinkedList;
import convexIntervals.CIntervals;
import times.Time;
public interface NCIntervals{
        public boolean addConvex(CIntervals I);
        public boolean add(CIntervals I);
        public LinkedList<CIntervals> getNCIInterval();
        public void setNCIInterval(LinkedList<CIntervals> nc);
        public boolean isEmpty();
        public boolean isIncluding(Time t);
        public boolean isContaining(CIntervals I);
        public CIntervals latest();
        public CIntervals earliest();
        public int duration();
```

```
	public int count();
	public CIntervals diameter();
	public NCIntervals union(CIntervals CI);
	public NCIntervals union(NCIntervals NCI);
	public NCIntervals intersection(CIntervals CI);
	public NCIntervals intersection(NCIntervals NCI);
	public NCIntervals complement();
	public NCIntervals difference(CIntervals CI);
	public NCIntervals difference(NCIntervals NCI);
	public String repString();
}

import java.util.LinkedList;
import java.util.ListIterator;
import convexIntervals.CIntervals;
import convexIntervals.ConvexInterval;
import convexIntervals.DeterminateInterval;
import convexIntervals.IndeterminateInterval;
import times.Nil;
import times.Time;
public class NonConvexInterval implements NCIntervals{
/**
 *
 */

private LinkedList<CIntervals> nc;

/**
 * Create a new empty non convex interval
 */
public NonConvexInterval(){
	nc=new LinkedList<CIntervals>();
}

/**
 * Create a new non-convex interval out of a convex one.
 * @param The convex interval that should be converted to non-convex
 */
public NonConvexInterval(CIntervals I){
	nc=new LinkedList<CIntervals>();
	if(!I.isEmpty())
		nc.add(I);
}

/**
 * Create a new non-convex interval out of LinkedList of CIntervals.
 * @param The LinkedList of CIntervals that should be converted to a non-convex Interval
 */
public NonConvexInterval(LinkedList<CIntervals> nc){
	this.nc=nc;
}

/**
 * Returns the NCI Intervals LinkedList.
 * @return the linked list of the Intervals
 */
public LinkedList<CIntervals> getNCIInterval(){
```

```
		return nc;
}

/**
 * Sets the non-convex NCI LinkedList to the one provided by the parameter.
 * @param nc a LinkedList of CIntervals.
 */
public void setNCIInterval(LinkedList<CIntervals> nc){
		this.nc=nc;
}

public boolean isEmpty(){
		return this.nc.isEmpty();
}

public boolean isIncluding(Time t) {
		ListIterator<CIntervals> it=nc.listIterator();
		while(it.hasNext()){
			if(it.next().isIncluding(t))
				return true;
		}
		return false;
}

public boolean isContaining(CIntervals I) {
		ListIterator<CIntervals> it=nc.listIterator();
		while(it.hasNext()){
			CIntervals ci=it.next();
			if(ci.isContaining(I))
				return true;
		}
		return false;
}

public CIntervals earliest(){
		return this.nc.getFirst();
}

public CIntervals latest(){
		return this.nc.getLast();
}

public CIntervals diameter(){
		CIntervals I;
		if(this.isEmpty())
			I=new ConvexInterval(new Nil());
		else
			I=new ConvexInterval( earliest().getStart(),latest().getEnd());
		return I;
}

public int duration(){
		int dur=0;
		ListIterator<CIntervals> it=nc.listIterator();
		while(it.hasNext() && dur >-1){
			int d=it.next().duration();
			if(d==-1)
```

```
                        dur=-1;
                else
                        dur+=d;
        }
        return dur;
}

/**
 * Counts the members of the Non-Convex Interval
 * @return the number that represents the membership count of convex intervals
 */
public int count() {
        return this.nc.size();
}

/**
 * In this NonConvex Interval the convex Interval I is added at the end  of the List
 * @param I the Convex Interval that is added to this Non-Convex Interval
 * @return True on succeeding to add the Interval
 */
public boolean add(CIntervals I){
        if(I.isEmpty())
                return false;
        else
                return this.nc.add(I);
        }

/**
 * Adding a new convex interval in the set of non-convex intervals
 * and preserves the ordering of the list. Operates a merging of intervals if necessary.
 * @param The convex interval that should be added in the non-convex interval I
 * @return true if the addition was successful
 */
public boolean addConvex(CIntervals I){
        if(I.isEmpty())
                return false;
        if(this.nc.isEmpty())
                return nc.add(I);
        else{
                ListIterator<CIntervals> it=nc.listIterator();
                while(it.hasNext()){
                        CIntervals n=it.next();
                        if(I.isBefore(n)){
                                break;
                        }
                        else if(I.isMergeable(n)){
                                I=I.mergeWith(n);
                                it.remove();
                        }
                }
                it.add(I);
                return true;
        }
}

// UNIONS //
/**
```

```
 * Performs the set union of a convex interval in the non-convex interval
 * preserves the ordering and merges mergeable members.
 * @param The convex interval I that will be unified in the non convex interval.
 * @return the Union as a non convex Interval
 */
public NCIntervals union(CIntervals I){
	NonConvexInterval res;
	res= new NonConvexInterval(this.union(I, this.getNCIInterval()));
	return res;
}

/**
 * Returns a new LinkedList of CIntervals (ordered) that is the union of the nci LinkedList of CIntervals
 * with the CInterval I.
 * @param I The interval operand of the union.
 * @param nci the LinkedList of CIntervals operand of the union
 * @return
 */
protected LinkedList<CIntervals> union(CIntervals I, LinkedList<CIntervals> nci){
	LinkedList<CIntervals> res=new LinkedList<CIntervals>();

	if(I.isEmpty())
		res=nci;
	else if(nci.isEmpty())
		res.add(I);
	else{
		ListIterator<CIntervals> it=nci.listIterator();
		while(it.hasNext()){
			CIntervals n=it.next();
			if(I.isBefore(n)){
				res.add(I);
				I=n;
			}
			else if(I.isMergeable(n)){
				I=I.mergeWith(n);
			}
			else if(I.isAfter(n)){
				res.add(n);
			}
		}
		res.add(I);
	}
	return res;
}

/**
 * Implements the union of this non-convex Interval with the non-convex Interval given as parameter
 * and returns the unified result.
 * @param N The non convex interval that will be unified with this non-convex Interval.
 * @return The resulted Union of the Intervals.
 */
public NCIntervals union(NCIntervals N){
	return new NonConvexInterval(union(this.getNCIInterval(),N.getNCIInterval()));
}

/**
 * Implements the union of LinkedList of CIntervals N1 with the LinkedList of CIntervals N2
```

```
 * and returns the unified result as LinkedList of CIntervals.
 * @param N1 a LinkedList of CIntervals that will be unified.
 * @param N2 a LinkedList of CIntervals.
 * @return The resulted Union of the CIntervals as an ordered LinkedList.
 */
protected LinkedList<CIntervals> union(LinkedList<CIntervals> N1, LinkedList<CIntervals> N2){
	LinkedList<CIntervals> res;
	if(N1.isEmpty()){
		res=N2;
	}
	else if(N2.isEmpty()){
		res=N1;
	}
	else{
		res=N2;
		ListIterator<CIntervals> it=N1.listIterator();
		while(it.hasNext()){
			CIntervals n=it.next();
			res = union(n, res);
		}
	}
	return res;
}

/**
 * Implements the unification of the non-convex Interval N given as parameter, in this non-convex
Interval.
 * @param N The non convex interval that will be unified in this non-convex Interval.
 */
public void unifyIn(NCIntervals N){
	if(!N.isEmpty()){
		ListIterator<CIntervals> it=N.getNCIInterval().listIterator();
		while(it.hasNext()){
			CIntervals n=it.next();
			this.addConvex(n);
		}
	}
}

// INTERSECTIONS //
/**
 * Performs an Intersection of the non-convex interval with the convex interval I that is given for
parameter
 * @param I The convex interval that is to intersect with this non-convex interval.
 * @return The Non-Convex Intersection.
 */
public NCIntervals intersection(CIntervals I){
	return new NonConvexInterval(intersection(I, this.nc));
}

/**
 *  Performs an Intersection of the non-convex interval nci with the convex interval I that is given for
parameter
 * @param I The convex interval operand of the Intersection
 * @param nci The non-convex interval operand of the Intersection
 * @return The Non-Convex Intersection of parameters.
 */
```

```
protected LinkedList<CIntervals> intersection(CIntervals I, LinkedList<CIntervals> nci){
	LinkedList<CIntervals> res=new LinkedList<CIntervals>();
	if(nci.isEmpty() || I.isEmpty())
		return res;
	else{
		ListIterator<CIntervals> it=nci.listIterator();
		while(it.hasNext()){
			CIntervals n=it.next();
			if(I.isIntersecting(n)){
				CIntervals inter=n.intersectWith(I);
				res.add(inter);
			}
			else if(I.isBefore(n))
				break;
			}
		}
	return res;
}

/**
 * Implements the Intersection of this non-convex Interval with the non-convex Interval given as
parameter.
 * @param N The non convex interval that will be unified with this non-convex Interval.
 * @return The resulted Union of the Intervals.
 */
public NCIntervals intersection(NCIntervals N){
	return new NonConvexInterval(intersection(this.nc,N.getNCIInterval()));
}

/**
 * Implements the Intersection of LinkedList of CIntervals N1 with the LinkedList of CIntervals N2
 * and returns the intersection result as LinkedList of CIntervals.
 * @param N1 a LinkedList of CIntervals.
 * @param N2 a LinkedList of CIntervals.
 * @return The resulted Intersection of the CIntervals as an ordered LinkedList.
 */
protected LinkedList<CIntervals> intersection(LinkedList<CIntervals> N1, LinkedList<CIntervals> N2){
	LinkedList<CIntervals> res=new LinkedList<CIntervals>();
	if(N1.isEmpty() || N2.isEmpty())
		return res;
	else{
		ListIterator<CIntervals> it=N1.listIterator();
		while(it.hasNext()){
			CIntervals In=it.next();
			res=this.union(intersection(In, N2),res);
		}
	}
	return res;
}

// COMPLEMENTS //

/**
 * Constructs the complement of the non convex Interval
 * @return the non-convex Interval that is the complement of this non-convex Interval
 */
public NCIntervals complement() {
```

```
		NCIntervals res = new NonConvexInterval(this.complement(this.getNCIInterval()));
		return res;
	}

	/**
	 * Constructs the complement of the LinkedList of CIntervals
	 * @param I a LinkedList of CIntervals.
	 * @return the complement as a LinkedList of CIntervals
	 */
	protected LinkedList<CIntervals> complement(LinkedList<CIntervals> I) {
		LinkedList<CIntervals> res=new LinkedList<CIntervals>();
		if(I.isEmpty()){
			res.add(new ConvexInterval().infinite());
		}
		else{
			ListIterator<CIntervals> it=I.listIterator();
			CIntervals In=it.next();
			res=union(In.complement(),res);
			while(it.hasNext()){
				In=it.next();
				res=intersection(In.complement(), res);
			}
		}
		return res;
	}

	// DIFFERENCES //
	/**
	 * Subtract from this Non-Convex Interval the convex interval CI
	 * @param CI The convex interval that is subtracted
	 * @return The non-convex difference
	 */
	public NCIntervals difference(CIntervals CI) {
		return new NonConvexInterval(difference(this.nc,CI));
	}

	/**
	 * Subtract from LinkedList of CIntervals the CInterval CI
	 * @param CI The CIntervals interval that is subtracted
	 * @param NCI a LinkedList of CIntervals.
	 * @return The non-convex difference of NCI by CI
	 */
	protected LinkedList<CIntervals> difference(LinkedList<CIntervals> NCI, CIntervals CI) {
		LinkedList<CIntervals> res;
		if(CI.isEmpty() || NCI.isEmpty()){
			res=NCI;
		}
		else{
			res=new LinkedList<CIntervals>();
			ListIterator<CIntervals> it=NCI.listIterator();
			while(it.hasNext()){
				CIntervals In=it.next();
				res=this.union(In.difference(CI), res);
			}
		}
		return res;
	}
```

```
/**
 * Subtract as a set difference from the Convex Interval CI the non Convex Interval NCI
 * @param CI the convex interval as first operand of the difference
 * @param NCI the non-convex interval as second operand of the difference
 * @return the non-convex interval of the difference
 */
protected LinkedList<CIntervals> difference(CIntervals CI, LinkedList<CIntervals> NCI) {
	LinkedList<CIntervals> res=new LinkedList<CIntervals>();
	if(CI.isEmpty()){
		return res;
	}
	else if( NCI.isEmpty()){
		res.add(CI);
	}
	else{
		ListIterator<CIntervals> it=NCI.listIterator();
		CIntervals In=it.next();
		if(CI.isBefore(In)){
			res.add(CI);
		}
		else{
			res=union(CI.difference(In), res);
			while(it.hasNext()){
				In=it.next();
				if(CI.isBefore(In))
					return res;
				else if(CI.isAfter(In))
					continue;
				else
					res=intersection(CI.difference(In),res);
			}
		}
	}
	return res;
}

/**
 * Subtract from this Non-Convex Interval the non-convex interval NCI
 * @param NCI The non-convex interval that is subtracted
 * @return The non-convex difference
 */
public NCIntervals difference(NCIntervals NCI) {
	return new NonConvexInterval(this.difference(nc, NCI.getNCIInterval()));
}

/**
 * Subtract from Non-Convex Interval NCI1 the non-convex interval NCI2
 * @param NCI1 The non-convex interval as first operand
 * @param NCI2 The non-convex interval that is subtracted
 * @return The non-convex difference
 */
protected LinkedList<CIntervals> difference(LinkedList<CIntervals> NCI1, LinkedList<CIntervals> NCI2)
{
	LinkedList<CIntervals> res;
	if(NCI1.isEmpty() || NCI2.isEmpty()){
```

```
                res=NCI1;
        }
        else{
                res=new LinkedList<CIntervals>();
                ListIterator<CIntervals> it1=NCI1.listIterator();
                ListIterator<CIntervals> it2=NCI2.listIterator();
                CIntervals CI1=it1.next();
                CIntervals CI2=it2.next();
                boolean next=true;
                do{
                        if(CI1.isBefore(CI2)){
                                res.add(CI1);
                                if(it1.hasNext())
                                        CI1=it1.next();
                                else
                                        next=false;
                        }else if(CI1.isAfter(CI2)){
                                if(it2.hasNext()){
                                        CI2=it2.next();
                                }
                                else{
                                        while(it1.hasNext())
                                                res.add(it1.next());
                                        next=false;
                                }
                        }
                        else{
                                res=union(CI1.lowDifference(CI2), res);
                                if((CI1 instanceof DeterminateInterval) && (CI2 instanceof
IndeterminateInterval) )
                                        res=union(CI2, res);
                                CI1=CI1.highDifference(CI2);
                                if(CI1.isEmpty()){
                                        if(it1.hasNext())
                                                CI1=it1.next();
                                        else
                                                next=false;
                                }
                                else{
                                        if(it2.hasNext())
                                                CI2=it2.next();
                                        else{
                                                res.add(CI1);
                                                next=false;
                                        }
                                }
                        }
                }while(next);
        }
        return res;
}

/**
 * Constructs a string representation of this Non-Convex Interval
 * in the form of {CI1,...,CIN}. Where CI is the string representation of Convex Intervals.
 * @returns a pictorial string representation.
 */
```

```
public String repString(){
	String s;
	if(this.nc.isEmpty()){
		s="{ }";
	}
	else{
		ListIterator<CIntervals> it=nc.listIterator();
		s="{ ";
		while(it.hasNext()){
			CIntervals n=it.next();
			s=s.concat(n.repString());
		}
		s=s.concat(" } ");
	}
	return s;
}

public String repString(LinkedList<CIntervals> I){
	String s;
	if(I.isEmpty()){
		s="{ }";
	}
	else{
		ListIterator<CIntervals> it=I.listIterator();
		s="{ ";
		while(it.hasNext()){
			CIntervals n=it.next();
			s=s.concat(n.repString());
		}
		s=s.concat(" } ");
	}
	return s;
}

}
import java.util.Iterator;
import java.util.LinkedList;
import java.util.ListIterator;
import times.Nil;
import convexIntervals.CIntervals;
import convexIntervals.ConvexInterval;
import convexIntervals.DeterminateInterval;
import convexIntervals.IndeterminateInterval;

public class SortedNonConvexInterval extends NonConvexInterval {

public SortedNonConvexInterval(){
	super();
}

public SortedNonConvexInterval(NCIntervals N){
	this.setNCIInterval(N.getNCIInterval());
}

public SortedNonConvexInterval(LinkedList<CIntervals> nc){
	this.setNCIInterval(nc);
}
```

```
public SortedNonConvexInterval(CIntervals I){
	super(I);
}

public CIntervals diameter(){
	CIntervals I;
	if(this.isEmpty())
		I=new IndeterminateInterval(new Nil());
	else{
		I=new ConvexInterval( earliest().getStart(),latest().getEnd());
	}
	return I;
}

public CIntervals diameterDet(){
	CIntervals I;
	if(this.isEmpty() || earliestDet()==null)
		I=new IndeterminateInterval(new Nil());
	else{
		I=new ConvexInterval( earliestDet().getStart(),latestDet().getEnd());
	}
	return I;
}

public CIntervals diameterInd(){
	CIntervals I;
	if(this.isEmpty() || earliestInd()==null)
		I=new IndeterminateInterval(new Nil());
	else{
		I=new ConvexInterval( earliestInd().getStart(),latestInd().getEnd());
	}
	return I;
}

public int durationInd(){
	int d=0;
	ListIterator<CIntervals> it=this.getNCIInterval().listIterator();
	while(it.hasNext()){
		CIntervals CI=it.next();
		if(CI instanceof IndeterminateInterval){
			int di=CI.duration();
			if(di<0)
				return -1;
			d+=di;
			}
		}
	return d;
}

public int durationDet(){
	int d=0;
	ListIterator<CIntervals> it=this.getNCIInterval().listIterator();
	while(it.hasNext()){
		CIntervals CI=it.next();
		if(CI instanceof DeterminateInterval){
			int di=CI.duration();
```

```
                        if(di<0)
                                return -1;
                        d+=di;
                        }
                }
        return d;
}


public CIntervals latestInd(){
        CIntervals res=null;
        Iterator<CIntervals> it=this.getNCIInterval().descendingIterator();
        while(it.hasNext()){
                CIntervals CI=it.next();
                if(CI instanceof IndeterminateInterval){
                        res=CI;
                        break;
                }
        }
        return res;
}

public CIntervals earliestInd(){
        CIntervals res=null;
        ListIterator<CIntervals> it=this.getNCIInterval().listIterator();
        while(it.hasNext()){
                CIntervals CI=it.next();
                if(CI instanceof IndeterminateInterval){
                        res=CI;
                        break;
                }
        }
        return res;
}

public CIntervals latestDet(){
        CIntervals res=null;
        Iterator<CIntervals> it=this.getNCIInterval().descendingIterator();
        while(it.hasNext()){
                CIntervals CI=it.next();
                if(CI instanceof DeterminateInterval){
                        res=CI;
                        break;
                }
        }
        return res;
}

public CIntervals earliestDet(){
        CIntervals res=null;
        ListIterator<CIntervals> it=this.getNCIInterval().listIterator();
        while(it.hasNext()){
                CIntervals CI=it.next();
                if(CI instanceof DeterminateInterval){
                        res=CI;
                        break;
                }
```

```
		}
		return res;
}

// UNIONS //
/**
 * Performs the set union of a sorted convex interval in the sorted non-convex interval
 * preserves the ordering and merges mergeable members.
 * @param The sorted convex interval I that will be unified in the non convex interval.
 * @return the Union as a non convex Interval with sorted members
 */
public NCIntervals union(CIntervals I){
		SortedNonConvexInterval res;
		res= new SortedNonConvexInterval(this.union(I, this.getNCIInterval()));
		return res;
}

/**
 * Implements the union of this sorted non-convex Interval with the sorted non-convex Interval given
as parameter
 * and returns the unified result.
 * @param N the sorted non convex interval that will be unified with this non-convex Interval.
 * @return The resulted Union of the Intervals as a sorted non-convex Interval.
 */
public NCIntervals union(NCIntervals N){
		return new SortedNonConvexInterval(union(this.getNCIInterval(),N.getNCIInterval()));
}

// INTERSECTIONS //
/**
 * Performs an Intersection of the non-convex interval with the convex interval I that is given for
parameter
 * @param I The convex interval that is to intersect with this non-convex interval.
 * @return The Non-Convex Intersection.
 */
public NCIntervals intersection(CIntervals I){
		return new SortedNonConvexInterval(intersection(I, this.getNCIInterval()));
}

/**
 * Implements the Intersection of this sorted non-convex Interval with the sorted non-convex Interval
given as parameter.
 * @param N The non convex interval that will be unified with this non-convex Interval.
 * @return The resulted Union of the Intervals as a sorted non convex Interval.
 */
public NCIntervals intersection(NCIntervals N){
		return new SortedNonConvexInterval(intersection(this.getNCIInterval(),N.getNCIInterval()));
}

// COMPLEMENTS //
/**
 * Constructs the complement of the non convex Interval
 * @return the non-convex Interval that is the complement of this non-convex Interval
 */
public NCIntervals complement() {
		NCIntervals res = new SortedNonConvexInterval(this.complement(this.getNCIInterval()));
		return res;
```

```
}

// DIFFERENCES //
/**
 * Subtract from this sorted Non-Convex Interval the sorted convex interval CI
 * @param CI The sorted convex interval that is subtracted
 * @return The sorted non-convex difference
 */
public NCIntervals difference(CIntervals CI) {
	return new SortedNonConvexInterval(difference(this.getNCIInterval(),CI));
}

/**
 * Subtract from this Sorted Non-Convex Interval the sorted Non-convex interval NCI
 * @param NCI The non-convex interval that is subtracted
 * @return The non-convex difference
 */
public NCIntervals difference(NCIntervals NCI) {
	return new SortedNonConvexInterval(this.difference(this.getNCIInterval(),
NCI.getNCIInterval()));
}


/**
 * Extracts the a LinkedList of Determinate CIntervals from this sorted Non-Convex Interval
 * @return The full determinate part of this Sorted Non-Convex Interval as a LinkedList
 */
protected LinkedList<CIntervals> determinateList(){
	LinkedList<CIntervals> res=new LinkedList<CIntervals>();
	ListIterator<CIntervals> it=this.getNCIInterval().listIterator();
	while(it.hasNext()){
		CIntervals CI=it.next();
		if(CI instanceof DeterminateInterval){
			res.add(CI);
		}
	}
	return res;
}

/**
 * Extracts the a LinkedList of Indeterminate CIntervals from this sorted Non-Convex Interval
 * @return the full indeterminate part of this Sorted Non-Convex Interval as a LinkedList
 */
protected LinkedList<CIntervals> indeterminateList(){
	LinkedList<CIntervals> res=new LinkedList<CIntervals>();
	ListIterator<CIntervals> it=this.getNCIInterval().listIterator();
	while(it.hasNext()){
		CIntervals CI=it.next();
		if(CI instanceof IndeterminateInterval){
			res.add(CI);
		}
	}
	return res;
}

public NCIntervals determinate() {
	return new SortedNonConvexInterval(this.determinateList());
```

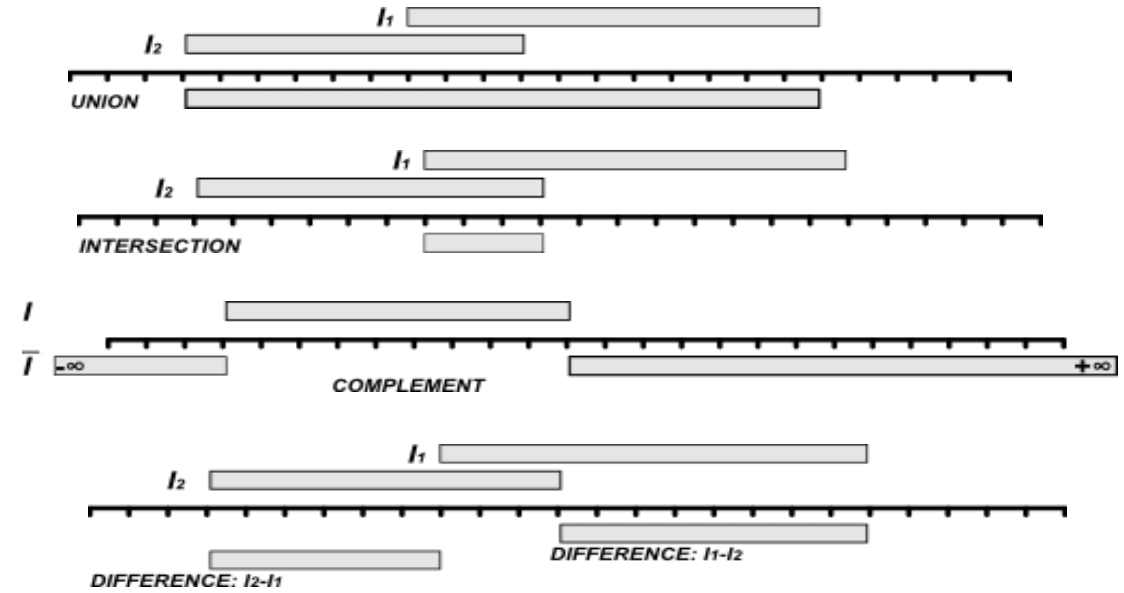
```
}

public NCIntervals indeterminate() {
        return new SortedNonConvexInterval(this.indeterminateList());
}
}
```

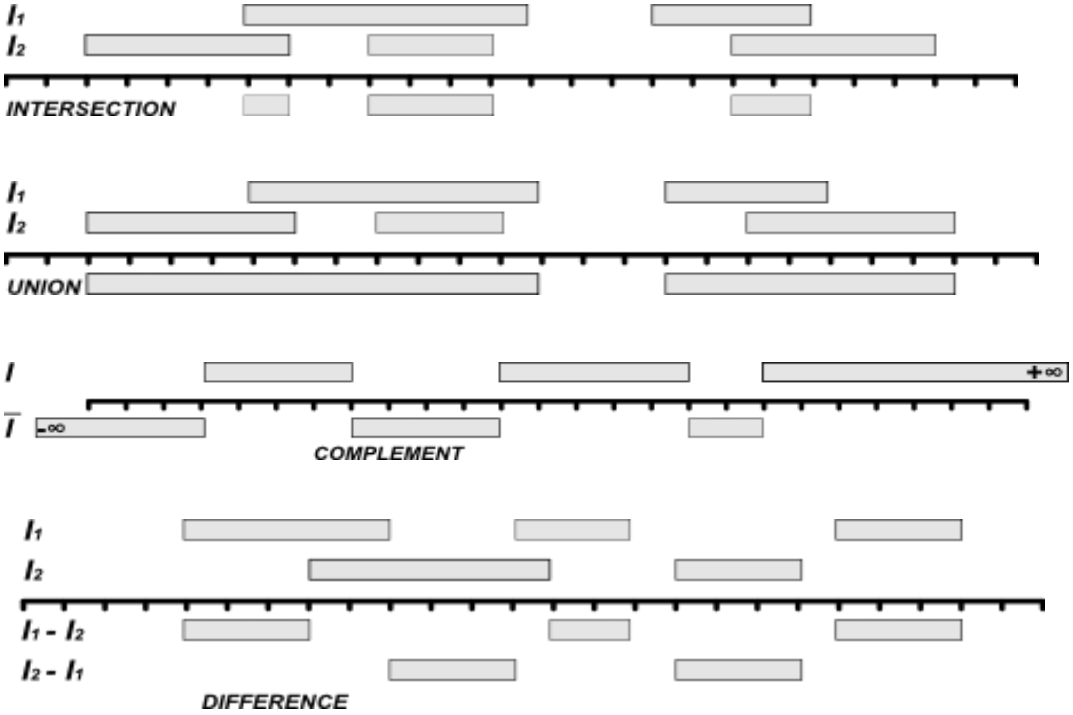
Appendix C

Visual summary

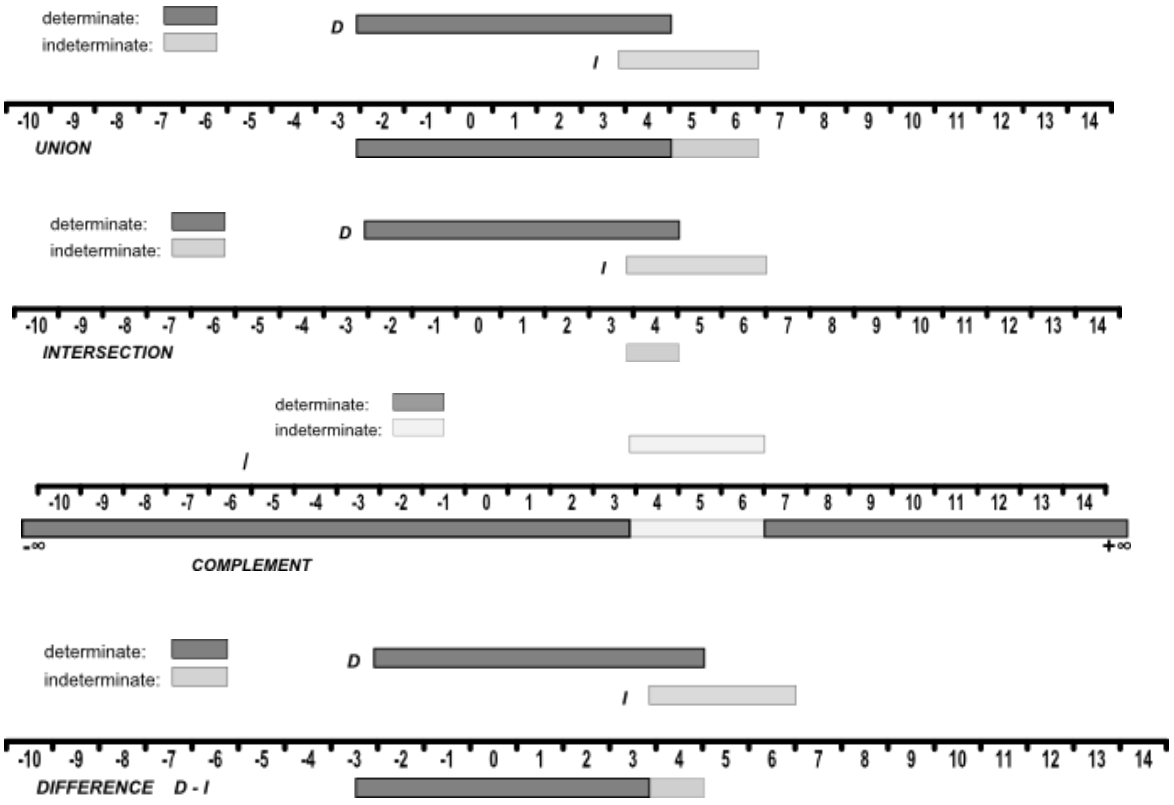
Convex Intervals



Non-Convex Intervals



Sorted Convex Intervals



Sorted Non- Convex Intervals