SEMANTIC INTEGRATION OF TIME ONTOLOGIES

by

Darren Ong

A thesis submitted in conformity with the requirements for the degree of Masters of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Copyright © 2011 by Darren Ong
Abstract

Semantic Integration of Time Ontologies

Darren Ong

Masters of Applied Science

Graduate Department of Mechanical and Industrial Engineering

University of Toronto

2011

Here we consider the verification and semantic integration for the set of first-order time ontologies by Allen-Hayes, Ladkin, and van Benthem that axiomatize time as points, intervals, or a combination of both within an ontology repository environment. Semantic integration of the set of time ontologies is explored via the notion of theory interpretations using an automated reasoner as part of the methodology. We use the notion of representation theorems for verification by characterizing the models of the ontology up to isomorphism and proving that they are equivalent to the intended structures for the ontology. Provided is a complete account of the meta-theoretic relationships between ontologies along with corrections to their axioms, translation definitions, proof of representation theorems, and a discussion of various issues such as class-quantified interpretations, the impact of namespacing support for Common Logic, and ontology repository support for semantic integration as related to the time ontologies examined.
Acknowledgements

It is difficult to overstate my gratitude to my M.A.Sc supervisor, Professor Michael Grüninger. His passion, enthusiasm, and efforts in explaining concepts clearly helped me progress smoothly in the new and sometimes challenging fields of ontologies and logic. His encouragement, patience, and sound advice throughout my thesis writing period has been a great source of inspiration. Thank you for making my time as a graduate student unintimidating, stimulating and productive.

Thank you to the members on my thesis committee, Professor Mark Fox, Professor Tamir El-Diraby, and my supervisor Professor Michael Grüninger.

I would like to thank all my colleagues in the Semantic Technologies Lab at the University of Toronto for providing a fun, motivating environment to work, learn, and grow as a graduate student.

Thank you to Sule Alp, my supervisor during my employment at Environment Canada, for encouraging me to pursue my goal of post-graduate studies. You were a great role model and I learned many things under your guidance, from technical skills and knowledge to just being a good hard-working person.

Most importantly, I would like to thank my parents for always believing in me and supporting my goals. Thank you for raising me to think for myself, to not be afraid of striving for what is true to my heart, and to have the confidence to see every situation through to the end.
## Contents

1 Introduction 1
   1.1 Ontologies of Time ........................................... 2
   1.2 Related Work for Time Ontologies .......................... 4
   1.3 The Big Picture ............................................... 6

2 Methodology 8
   2.1 First Order and Common Logic Representation ............. 8
   2.2 The Common Logic Ontology Repository (COLORE) ........... 9
      2.2.1 Hierarchy Structure of Ontologies in COLORE .......... 10
   2.3 Relationships between Hierarchies ......................... 11
      2.3.1 Interpretability ........................................ 11
      2.3.2 Translation Definitions ................................ 12
      2.3.3 Proving Relationships between Theories ............... 12
   2.4 Verification of Ontologies ................................... 13
      2.4.1 Representation Theorems ................................ 14
   2.5 Related Work .................................................. 15

3 Verification of Linear Point 18
   3.1 Theory of Linear Point ($T_{linear\_point}$) ................... 18
   3.2 Infinite Linear Ordering ($T_{infinite\_linear\_ordering}$) ....... 20
   3.3 Interpretations with Class-Restricted Quantification ....... 21
   3.4 Translating Between $T_{linear\_point}$ and $T_{infinite\_linear\_ordering}$ ... 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  Notation</td>
<td>81</td>
</tr>
<tr>
<td>B  Glossary</td>
<td>82</td>
</tr>
<tr>
<td>C  Common Logic Theories</td>
<td>83</td>
</tr>
<tr>
<td>D  Prover9 Proof Output</td>
<td>122</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

There exists over 40 different ontologies of time in the literature, with further examination revealing that each ontology represents time in only one of three ways; as points, as intervals, or as both points and intervals. However, the relationships between these ontologies of time are not well understood. Therefore, the questions regarding how each of these independent theories are related must be answered in order to completely characterize the domain of time. Are all of these theories equivalent in some manner? What are the similarities and differences in ontological commitments made by each theory?

By thoroughly examining these similarities and differences between theories of time, we aim to gain a better understanding of the domain of time as a whole. Through identifying the similarities shared by all theories we can define the minimal set of defining characteristics (i.e. axioms) for time, and identifying their differences will define the various extensions to the minimal theory of time. The task of semantically integrating these ontologies of time will be carried out within the context of an ontology repository that supports ontology verification and integration through notions of interpretation, representation theorems, and nonconservative extensions.

With the growing number of new ontologies presented, it is increasingly important for ontology repositories to support the verification and semantic integration of stored ontologies. Evaluating the correctness of stored ontologies is critical in maintaining the credibility of the repository and the integrity of the proven relationships between ontologies. In order to promote ontology reuse, it is important that users of the repository are confident that the models of the axioms are exactly those intended by the ontology. Hand in hand with ontology verification is the task of discovering semantic integration
possibilities between ontologies stored in a repository. Semantic integration analysis between stored ontologies allows users to understand the unique ontological commitments made by the different axiomatizations. Through this, we can determine what information can be shared accurately, through the preservation of semantics during translation, between two systems using slightly different ontologies.

Since the domain of time is mature enough, we can focus on our goals of ontology evaluation and semantic integration without encroaching on issues more related to ontological design. Furthermore, time ontologies are diverse enough in the way they choose to axiomatize time (i.e. as points, intervals, or both) to allow us to explore the meta-theoretic relationships found between ontologies across a wide range in complexity. The intuitive nature of time is also a benefit as it allows the reader to easily interpret our results even during more complex cases such as the mapping between models of time points to those of time intervals. Beyond the results for semantic mapping procedures and evaluation of theories, we’ve also been able to discover new results related to time ontologies. The work done on the semantically integrating time ontologies covering the complete spectrum of representation as points, as intervals, and as both points and intervals, also leads to the larger question on whether we have a full characterization of time with the set of current time ontologies. That is, is there reason for new time ontologies or is the current collection enough to cover every possible extension of time?

1.1 Ontologies of Time

Surveying the various first-order ontologies of time from [14] and [22], we notice that there exist a few different perspectives on modelling time such as a set of points, a set of intervals, both points and intervals where points are first-class objects (i.e. point-continuum), and both points and intervals where intervals are first-class objects (i.e. glass-continuum). Often, these different perspectives of time present very different answers to the same question. An example from [14] considers the problem of dividing an interval into two equal halves and asks the question “which half contains the point of division?” From the perspective of the point-continuum, where intervals are defined by the set of points contained within them, the answer is that it is impossible to divide an interval into equal halves as the point of division must be contained in one of the two halves. This leads to the classification of intervals as either open or closed depending on whether the interval contains the point of division. If we look through the
perspective of the glass-continuum, then the two halves are indeed equal with the question regarding the location of the point of division becoming irrelevant since points are not part of the physical continuum. The question of which perspective is the correct one is further blurred when we take into account a few real-world examples as there seemingly many that support either one. Again, let us consider two examples from [14]. First, the example of turning off a light in a room seems to support the perspective of the glass-continuum as there are two distinct intervals of interest (i.e. when the light is on and when the light is off), with their meeting point inconsequential (does it make sense to consider the state of the light at the point between being on and off?). The second example is that of a ball being tossed into the air that seems to fall in line with the point-continuum, as there intuitively exists a point at which the ball’s upward velocity changes from positive to negative that we want to quantify as the ball being motionless. In addition to these two perspectives of time, we can describe another where the relationship between points and intervals is based on the idea that an interval represents the ambiguity relating to the exact location of a point [14]. Therefore, the size of an interval from this perspective is an expression about the degree of confidence we have about the point’s location. This leads to a different set of relations used in the axiomatization of this theory to describe the relationships between intervals. While the various perspectives are presented as different axiomatizations in [14] and [22], the verification of their models and the formal relationships between them are largely left open. Therefore, in gathering the set of theories that represent these different perspectives on modelling time, we set the stage for the exploration of their meta-theoretic relationships, both between ontologies that share the same perspective and between ontologies across the different perspectives. Here we briefly introduce each of the time ontologies of interest, arranged according to three categories; point theories, interval theories, and hybrid-time theories. These ontologies and their axioms are introduced and discussed in greater detail as they become relevant to the focus of each of the upcoming chapters.

**Linear-Point** - A point-theory of time as infinite linearly ordered time points.

**Interval-Meeting** - An interval-theory of time as intervals related by the points at which they meet.

**Approximate-Point** - An interval-theory of time as the approximate locations of points.

**Periods** - An interval-theory of time as intervals related by inclusion and precedence.
**Endpoints** - A hybrid-time theory through the perspective of the glass-continuum.

**Vector-Continuum** A hybrid-time theory representing time through forward and backward intervals.

**Point-Continuum** - A hybrid-time theory through the perspective of the point-continuum.

With all of these ontologies of time having been published for over two decades, little has been done in regards to formally characterizing the models of their axioms to verify that they are equivalent to the set of intended models. Furthermore, with each of these theories having a different take on how to represent time, the question of how each of their models are related has yet to be explored in detail. In this thesis, we focus on these two issues using the notions of interpretability and representation theorems to formally prove relationships between these theories to gain a better understanding of how these ontologies of time are semantically integrated.

### 1.2 Related Work for Time Ontologies

The primary source of the first-order time ontologies found in our work comes from [14]. This tech-report presents a collection of ontologies that represent time from many different perspectives such as points, intervals, both points and intervals, ticks on a clock, and duration. However, our work focuses only on the most prominent intuitions of time so only theories of points, intervals, and both points and intervals are considered. The axioms for each theory and every defined extension of them are provided in Common Logic syntax throughout the report. While each of the ontologies have a section dedicated to the discussion of their models, no formal work in characterizing them is done. Thus, our work seeks to verify the claims regarding the intended models of the axiomatizations as well as those regarding the relationships between the theories presented in a formal manner using automated reasoners.

In [22], van Benthem provides us with the axiomatizations for a set of time theories named Periods. These theories were presented as a mode natural alternative to the view of time as points. The goal was to construct a period theory of time that complimented the existing classical theories. What is given is a set of theories ranging from what is defined as the minimal theory of periods, to the two major period structures of $T_{INT(Q)}$ and $T_{INT(Z)}$. The first of the two major structures presents a continuous perspective consisting of all open intervals of rational numbers, and the second a more discrete example
of all open intervals between the integers. The axioms for the period theories consider more complex relations for intervals such as greatest lower-bound and least upper-bound between pairs of intervals, the convexity of intervals, in addition to others that present different extensions to the theories of time collected from [14]. Another important contribution of this work is the proofs provided that verify the theories of $T_{INT}^\mathbb{Q}$ and $T_{INT}^\mathbb{Z}$ axiomatize exactly the set of intended models.

In [21], van Benthem and Pearce provide a mathematical characterization of interpretation between theories helpful in defining the necessary translation definitions between theories of different languages. The first part of the paper characterizes the case of simple interpretability and shows the need and method of defining a fixed unary formula by which the quantifiers of all translated formulas must be relativized. We used this to define a method for translating between a theory with class-quantified relations and one without in the context of our procedure. The second half of the paper discusses the notion of general interpretability to cover the examples of reduction between first-order theories. This characterization of interpretation where sets of objects in one theory are mapped to single objects in another was needed when defining a set of translation definitions to prove interpretability between theories of time as intervals and theories of time as points.

In [1], the axiomatization of an interval theory of time is presented where intervals are related by the points at which they meet. This theory extends the one provided in [14] and therefore relevant in constructing a more complete hierarchy of theories for that particular time ontology. Doing so gives us a better coverage of the existing extensions of time theories to work with when we integrate the different ontologies. Furthermore, this paper provides a discussion on the mapping techniques between theories of time as points to those of time as intervals that is useful for our own investigation of those relationships between ontologies in the repository. For this, an embedded type of mapping is presented through the definition of a theory that incorporates both points and intervals along with the set of relations and functions that correlate the two classes of objects, and a set-theoretic definition of points from intervals is given.

Ladkin in [17] extends upon the work done on the theory presented in [1] by characterizing the models of that theory along with a few of his own extensions. Ladkin uses a definable equivalence relation on pairs of intervals to introduce the concept of points, thereby allowing him to prove decidability for the theory presented in [1]. However, of more interest to us was the work done to extend the theory from
Chapter 1. Introduction

[1] to show that the rational intervals, $\text{INT}(\mathbb{Q})$, is the only countable model up to isomorphism for that extension. Thus, enabling the comparison of the theory in [1] with the one specified by van Benthem in [22].

1.3 The Big Picture

The overarching goal of this work is to gain what we can call a complete understanding of the models of time. With all the different representations of time and all the ontologies that exist, can we have a complete characterization of all the possible models of time? In order to answer this, we must first determine how each of the existing theories of time are related. That is, determining if they axiomatize equivalent models of time, and if not, what the differences and similarities of their models are. We want to know whether it is possible to extend one ontology of time to incorporate the different ontological stances taken by each of the other representations. Secondly, we must be able to verify that the intended models for time are indeed only those that are captured by the axiomatizations provided by each ontology. So for each theory of time provided, we would like to be able to characterize their models with well-understood classes of mathematical structures. Using representation theorems we attempt to prove that every intended structure is a model of the ontology, and that every model of the ontology is elementary equivalent to the set of intended structures.

Chapter 2 introduces the repository environment that serves as the setting for our work. We describe the structure for ontologies required by the repository and the meta-theoretic relationships supported. The mathematical notions of interpretation and representation theorems are presented along with our procedure for proving such relationships between theories using the aid of an automated reasoner.

Our work with time ontologies begins in Chapter 3 by proving the representation theorems for the theory of time represented as points, $T_{\text{linear-point}}$. In doing so, we demonstrate the viability of our methodology and address the issue that arises when interpreting between a theory with class-quantified relations and one without.

In Chapter 4, we move to the set of time ontologies that represent time as intervals. Since there are three such ontologies, we focus on defining the relationships between these ontologies in order to understand the similarities and differences between their models. We look at the possibilities of extending
a particular hierarchy with new theories that axiomatize the similarities with another hierarchy. Furthermore, we discuss some of the common issues with using an automated theorem prover and present a few general techniques that improve the effectiveness and efficiency of finding the required proofs.

In Chapter 5, we examine the final category of time ontologies, what we call hybrid-time ontologies, that conceptualize time using both points and intervals as well as the relationships between them. With three different hybrid-time theories present, we begin by exploring the relationships that exist between them in order to understand the differences between their models and then proceed to verify each of the hybrid-time ontologies via representation theorems. During the course of this chapter, we encounter an issue with the lack of namespaces support by the Common Logic syntax, as some of these theories share overlapping lexicon with differing semantics. We look at the situations where namespaces becomes a problem in the context of an ontology repository and discuss the repercussions of using the current Common Logic syntax.

In Chapter 6, we focus on the relationships between theories across the major categories of time representation. That is, the relationships between the models for theories of time as points to theories of time as intervals as well as the hybrid-time theories. We look at the complex set of translation definitions required to map between such models and demonstrate how we are able to utilize previously proven relationships to define new ones. We show that as more relationships are proven (i.e. the stored hierarchies become better integrated), the work required to complete the network of relationships becomes substantially less since we can leverage past proofs and the transitive property of the meta-theoretic relationships between theories. We then present an extension to the current repository architecture to support modules that store the work associated with establishing the meta-theoretic relationships between theories in the repository.

Finally, we summarize the key findings of each chapter and discuss how they come together in progressing toward our goal of finding a complete characterization of time. Future work is discussed with respect to the remaining obstacles completely integrating the set of time ontologies examined, and applications of this work in a more general context within a repository.
Chapter 2

Methodology

In this chapter, we introduce the repository environment that serves as the context of our work regarding verification and semantic integration of ontologies. We explain the choice of focusing on first-order logic (FOL) ontologies of time axiomatized in Common Logic Interchange Format (CLIF) notation and its relation to the set of chosen meta-theoretic relationships. Next, we introduce hierarchical structure of ontologies stored in the Common Logic Ontology Repository (COLORE) and each of the meta-theoretic relationships utilized by COLORE. We define the notion of interpretability for comparing models of theories with different nonlogical lexicon and the associated translation definitions for translating the axioms of one theory into the language of the other. This leads to our outline of a semi-automated procedure that utilizes an automated reasoner to find the necessary proofs for proving the meta-theoretic relationships between theories. We discuss what it means to verify an ontology and show how it can be done by characterizing the models of its axiomatization up to isomorphism against a class of well-understood mathematical theories. In this way, we can ascertain if the axioms of the ontology capture exactly the set of intended models. Finally, we finish with a literature review covering work done in the field of relating ontology and ontology modules on a meta-theoretic level.

2.1 First Order and Common Logic Representation

To capture the semantics of the relations required to formalize notions of time, a language with the level of expressiveness on the order of FOL is necessary. If we look at the other popular ontology languages
such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL), we can see that they lack the expressiveness needed to axiomatize notions of time. For example, both RDF and OWL are unable to represent notions of convex time intervals or the greatest-lower-bound of overlapping intervals. While there exists an ontology of time for OWL called OWL-Time [10], we would resort to using its FOL representation found in [15] to determine its relationship with the rest of the time ontologies explored here.

Furthermore, the repository environment used to store these ontologies is based on representing axioms using Common Logic (ISO 24707), which is a standardized logical language for the specification of first-order ontologies and knowledge bases. As discussed in [13], this design choice due to the flexibility of CLIF to support the high-level of expressibility of FOL axioms. Since we are working in the context of the COLORE repository our meta-theoretic relationships between ontologies and the procedure to prove them is based on the first-order notions of interpretability and representation. The soundness (i.e. deriving only correct results) and completeness (i.e. ability to derive any logically valid implication) of FOL allows us to guarantee that anything proven using the axioms of a theory holds for all possible models of that theory. Although all theories of time examined here are found in the CLIF format in COLORE, the axioms are written in traditional first-order logic syntax in this thesis for readability.

2.2 The Common Logic Ontology Repository (COLORE)

The COLORE project is an open repository of first-order ontologies, specified using the CLIF syntax, that serves as a testbed for ontology evaluation and integration techniques to support the design, evaluation, and application of ontologies in first-order logic. The relationships identified between modules in COLORE integrate stored ontologies in a manner that allows for functionality that would otherwise be lost if the repository were just loose collections of axioms. In such a repository, users gain the ability to traverse stored ontologies in a more efficient and directed manner by having those sharing a similar domain explicitly linked. Users would be able to explore a single hierarchy composed of all related ontology modules, rather than the separate hierarchies within each ontology. Having the theories ordered by their relative strength within the hierarchy presents to users the theories that exist in each direction
(stronger or weaker) so they can recognize which theory should be explored next in order to find one that more precisely captures their intended models. Since each module of an ontology represents a different set of ontological commitments, having the repository connect all ontologies that share logical similarities would also increase the number of choices presented to the user for ontology design and reuse. Each ontology would now have available to it a set of new extensions through the translation of modules belonging to other ontologies connected in the repository. For example, when two ontologies are connected through the repository, they are able to use translation definitions within the repository to effectually share their modules.

An ontology repository of this nature also serves to facilitate the semantic integration of systems using different ontologies. If the systems are using ontologies that are related in the repository, it would be possible to use that relationship together with the stored translation definitions to understand what information can be accurately shared between them. Therefore, as the repository grows so do the semantic integration possibilities between stored ontologies.

### 2.2.1 Hierarchy Structure of Ontologies in COLORE

All theories within the repository are organized into hierarchies (as proposed in [12]):

**Definition 2.1** A hierarchy \( \mathcal{H} = (\mathcal{H}, <) \) is a partially ordered, finite set of theories \( \mathcal{H} = T_1, \ldots, T_n \) such that

1. \( \mathcal{L}(T_i) = \mathcal{L}(T_j) \), for all \( i, j \);

2. \( T_i \leq T_j \) iff for any \( \sigma \in \mathcal{L}(T_i) \),

\[
T_i \models \sigma \Rightarrow T_j \models \sigma
\]

If \( T_i \) and \( T_j \) are theories in the same hierarchy such that \( T_i < T_j \), then \( T_j \) is a nonconservative extension of \( T_i \). All theories in a particular hierarchy share the same set of nonlogical lexicon, and are ordered by nonconservative extension such that the extensions restrict the set of models of the theory it is extending. With respect to this ordering relation, we say that a theory \( T_i \) is stronger than a theory \( T_j \) if it is a nonconservative extension of \( T_j \) [11].

---

\(^1\)For any theory \( T \), the language of \( T \) is denoted by \( \mathcal{L}(T) \).
2.3 Relationships between Hierarchies

The key benefit of an ontology repository like COLORE comes from the network of meta-theoretic relations defined between the stored theories. These relationships allow us to compare the theories in the repository in a manner that is easier to understand than strictly comparing their axioms by instead looking at their models. This allows us to say whether one theory is stronger, weaker, equivalent to, or inconsistent with another. We can also define new theories that capture the shared models between two theories that have an overlapping set of models. For theories in the same hierarchy (i.e. share the same language), we use the previously defined nonconservative relationship. Another important aspect of a repository like COLORE is the ability to compare different ontologies in a manner similar to the way we look at extensions within a single hierarchy. To do so, we exploit what are known as interpretations. Interpretations allow us to compare theories across different hierarchies by using a set of conservative definitions, called translation definitions, that syntactically translates between the languages of two ontologies while preserving the original semantics of the relations. In order to implement this on a scale as large as a general repository, we define a procedure for discovering these relationships in a semi-automated way utilizing the aid of an automated theorem prover.

2.3.1 Interpretablity

In order to compare ontologies that are axiomatized in languages with disjoint nonlogical lexicon, we need to translate between the lexicon of each ontology while preserving the original semantics of the relations. Here we present in the meta-theoretic relationship between theories in the repository based on relative interpretation. This allows us to take advantage of automated reasoners as we are able to determine relationships between models of different ontologies by working strictly with their axioms (i.e. the soundness and completeness of FOL guarantees that any results derived from the axioms holds for all possible models of the theory).

Consider $T_1$ and $T_2$ to be a pair of axiomatized theories with disjoint nonlogical lexicon. An interpretation is the mapping from $L_1$, the language of $T_1$, to $L_2$, the language of $T_2$, such that the theorems of $T_1$ are preserved. The following definition is adopted from [6]:

**Definition 2.2** An interpretation $\pi$ of a theory $T_1$ with language $L_1$ into a theory $T_2$ with language $L_2$
is a function on the set of parameters of $L_1$ such that

1. $\pi$ assigns to $\forall$ a formula $\pi_\forall$ of $L_2$ in which at most the variable $v_1$ occurs free, such that

\[ T_1 \models (\exists v_1) \pi_\forall \]

2. $\pi$ assigns to each $n$-place relation symbol $P$ a formula $\pi_P$ of $L_2$ in which at most the variables $v_1, ..., v_n$ occur free.

3. For any sentence $\sigma$ in $L_1$,

\[ T_1 \models \sigma \Rightarrow T_2 \models \pi(\sigma) \]

Thus, $\pi$ is an interpretation of $T_1$ in $T_2$ if it preserves the theorems of $T_1$ in $T_2$. If two theories are mutually interpretable, that is $T_1$ is interpretable in $T_2$ and $T_2$ is interpretable in $T_1$, then they are considered definably equivalent.

### 2.3.2 Translation Definitions

If there is an interpretation of $T_1$ in $T_2$, then there exists a set of sentences that axiomatizes the mapping, called a translation definition, in the language of $L_1 \cup L_2$ of the form:

\[ (\forall \overline{x}) p_i(\overline{x}) \equiv \Phi(\overline{x}) \]

where $p_i(\overline{x})$ is a relation symbol in $L_0$ and $\Phi(\overline{x})$ is a formula in $L_1$ whose only free variables are $\overline{x}$.

Translation definitions can be considered to be an axiomatization of the interpretation of $T_1$ into $T_2$. They conservatively extend $T_2$ and a definitionally extend $T_1$.

### 2.3.3 Proving Relationships between Theories

The methodology for proving the relationship between two theories of different hierarchies follows from the steps used to determine definable equivalence of the theories. Suppose that $\Sigma_{12}$ are the translation definitions for $T_1$ into $T_2$, and that $\Sigma_{21}$ are the translation definitions for $T_2$ into $T_1$. The process of verifying that two theories $T_1, T_2$ are definably equivalent can be broken down into three reasoning problems:

1. $T_1 \cup T_2 \cup \Sigma_{12}$ is consistent;
2. \( T_1 \cup \Sigma_{12} \models T_2; \)

3. \( T_1 \cup T_2 \cup \Sigma_{21} \) is consistent;

4. \( T_2 \cup \Sigma_{21} \models T_1. \)

Success of all four reasoning problems proves that the theories \( T_1 \) and \( T_2 \) are are definably equivalent, so that they can be considered to be alternative axiomatizations of the same set of models. If the first or third step fails, so that the axioms of the two theories together with the translation definitions are inconsistent, then the two theories have disjoint sets of models and they are not translatable into one another. If only the second step fails, then \( T_1 \) might be weaker than \( T_2 \). If only the fourth step fails, then \( T_2 \) might be weaker than \( T_1 \). In the last two scenarios, we can only say that one theory is strictly weaker than the other if we are able to find a definably equivalent theory of the stronger theory, in the core hierarchy of the weaker, and show that it non-conservatively extends the weaker theory.

The goal of incorporating automated reasoners is to eventually automate the procedure of adding new theories into a hierarchy and deriving the relationships among theories in the same hierarchy as well as among theories in different hierarchies. Automated theorem provers are used for steps 2 and 4, while model builders are used for steps 1 and 3. This approach is not restricted to a particular theorem prover, but for the present paper, Prover9 and Mace4 [19] were chosen. Prover9 is a first-order logic automated theorem prover that uses resolution to prove that goal sentences are entailed by the background theory (assumptions). Mace4 is a finite-model generator used to find counter-examples of the goal.

### 2.4 Verification of Ontologies

Our method of verifying ontologies applies model-theoretic notions in analyzing ontologies. We characterize the semantics of an ontology via a set of structures which we refer to as the set of intended structures for the ontology. By specifying the set of intended structures with respect to the models of well-understood mathematical theories, we can define the extensions of relations in the intended structure against the properties of those models. This allows us to determine whether the axiomatization of an ontology matches its intended models. If an ontology’s axiomatization contains unintended models, then it is not possible to prove from the axioms of the ontology all of the sentences that are entailed by
the set of intended models. Since systems are considered semantically integrated if their sets of intended models are equivalent, unintended models arising from the axioms prevents the entailment of sentences critical for interoperability. For ontology applications in decision support, verification of an ontology allows us to make the claim that any inferences drawn by a reasoning engine using the axioms of the ontology are actually entailed by the ontology’s intended models.

To verify an ontology, we characterize its models up to isomorphism and determine if those models are elementary equivalent to the set of intended structures. This is formalized by the mathematical notion of representation theorems, where we prove that every intended structure is a model of the ontology and that every model of the ontology is elementary equivalent to some intended structure. To do so we leverage work done by mathematicians in for theories such as orderings, incidence structures, lattices, graphs and algebra to verify ontologies of domains such as time, process and mereotopology.

### 2.4.1 Representation Theorems

The primary challenge for someone attempting to prove representation theorems is to characterize the models of an ontology up to isomorphism.

**Definition 2.3** A class of structures $\mathcal{M}$ can be represented by a class of structures $\mathcal{N}$ iff there is a bijection $\varphi : \mathcal{M} \rightarrow \mathcal{N}$ such that for any $\mathcal{M} \in \mathcal{M}$, $\mathcal{M}$ is definable in $\varphi(\mathcal{M})$ and $\varphi(\mathcal{M})$ is definable in $\mathcal{M}$.

With the following theorem ([11]), we are able to reuse the methodology described for proving definable equivalence to support ontology verification.

**Theorem 2.1** A theory $T_1$ is definably equivalent with a theory $T_2$ iff the class of models Mod($T_1$) can be represented by Mod($T_2$).

Let $\mathcal{M}^{\text{intended}}$ be the class of intended structures for the ontology, and let $T_{\text{onto}}$ be the axiomatization of the ontology. The necessary direction of a representation theorem (i.e. if a structure is intended, then it is a model of the ontology’s axiomatization) can be stated as

$$\mathcal{M} \in \mathcal{M}^{\text{intended}} \Rightarrow \mathcal{M} \in \text{Mod}(T_{\text{onto}})$$
If we suppose that the theory that axiomatizes $\mathcal{M}^{intended}$ is the union of some previously known theories $T_1, ..., T_n$, then by Theorem 2.1 we need to show that $T_{onto}$ interprets $T_1 \cup ... \cup T_n$. If $\Delta$ is the set of translation definitions for this interpretation, then the necessary direction of the representation theorem is equivalent to the following reasoning task:

$$T_{onto} \cup \Delta \models T_1 \cup ... \cup T_n$$  \hspace{1cm} \text{(Rep-1)}

The sufficient direction of a representation theorem (any model of the ontology’s axiomatization is also an intended structure) can be stated as

$$\mathcal{M} \in \text{Mod}(T_{onto}) \Rightarrow \mathcal{M} \in \mathcal{M}^{intended}$$

In this case, we need to show that $T_1 \cup ... \cup T_n$ interprets $T_{onto}$. If $\Pi$ is the set of translation definitions for this interpretation, the sufficient direction of the representation theorem is equivalent to the following reasoning task:

$$T_1 \cup ... \cup T_n \cup \Pi \models T_{onto}$$  \hspace{1cm} \text{(Rep-2)}

## 2.5 Related Work

The Ontolingua Project out of Stanford [9] was one of the first large-scale ontology repositories to exist and brought with it a set of initial considerations for issues related to managing the storage of ontologies from a wide array of domains. However, due to the project’s openness in accepting user-generated ontologies and the lack of strict quality control over submitted content, the major benefit associated with having an ontology repository, that is the ability to reuse stored ontologies, was lost due to a lack of user confidence over the content. This serves to illustrate the importance of COLORE’s focus on verifying stored ontologies. By maintaining the integrity of its ontologies, the meta-theoretic relationships proven between stored ontologies retains much more usefulness.

HETS (Heterogeneous Tool Set) is used to manage libraries of ontologies specified in CASL [18]. These CASL specifications allow users to group parts of an ontology into smaller sub-theories while defining translations, unions, reductions and extensions between them. HETS visualizes these relationships via development graphs by denoting the dependencies between the theories. While the focus of HETS is developing ontologies (libraries) through the combination of smaller theories, we also want
to examine the relationships between similar libraries (sets) of theories. In [16], proving consistency of a large ontology is done by breaking it down into smaller, easier consistency proofs and proofs of conservativity of theory extensions. Relative consistency proofs are used by providing theory interpretations into another theory that is known or assumed to be consistent. Kutz and Mossakowski design the architectural specification that modularizes the DOLCE ontology and the theory interpretations into conservative extensions of those theories to establish the consistency of DOLCE via relative consistency proofs. However, more than consistency, we are interested in determining if sets of models of two differently axiomatized ontologies are equivalent. While it is possible to define CASL specifications for the CLIF theories in COLORE in order to operate within the HETS environment, it is no better suited to managing the meta-theoretic relationships across independent theories in a repository. This is because HETS is more so a tool focused on ontology modularization and integration for use as a single entity rather than a front-end to manage a large repository of varied ontologies.

The use of theory interpretations between first-order axiomatized theories as a means to combine smaller theories has been implemented by the Interactive Mathematical Proof System (IMPS). IMPS is a mathematical theorem prover that utilizes a repository of axiomatized mathematical theories linked to each other through theory interpretations using the little theories approach to mechanize traditional tools of classical mathematical reasoning [7]. In the little theories approach, the theory where the theorem is eventually stated and proved is constructed and through extending smaller theories and combining them through their theory interpretations. The use of theory interpretations by IMPS provide the means to transport a theorem from the theory it was proved in to any other theory linked with an interpretation. The IMPS repository is organized around the relative interpretations available between stored theories. Furthermore, IMPS guarantees the consistency of generated proofs based on the notion of relative consistency between theories. Within IMPS there is a set of theories deemed foundational, meaning they are regarded or known to be consistent. Since all proofs begin with a foundational theory and any theory developed from another is a conservative extension of the original theory, all theories developed are consistent relative to the original foundational theory [8]. Although the use and definitions of theory interpretations and relative consistency in IMPS are specific to the purpose of theorem proving, it nonetheless shows how such relationships can be utilized to relate and combine theories. However, our repository focuses on relating stored theories to facilitate ontology design and understanding with the
Chapter 2. Methodology

The $\varepsilon$-connection language is a formalism that allows the combination of decidable logics in a way that preserves decidability while adding expressiveness [4]. An $\varepsilon$-connection is a set of $\varepsilon$-connected ontologies that each model a different application domain, while the $\varepsilon$-connection itself models the union of all the domains [3]. Link properties combine information from different domains by establishing a connection between $\varepsilon$-connected ontologies. In OWL a link property is a binary relationship between instances, classes, and properties that belong to different $\varepsilon$-connected ontologies. However, link properties in OWL cannot be transitive or symmetric. In the case of using $\varepsilon$-connections to decompose a large ontology into modules (a collection of axioms), each module encapsulates some terms of the original ontology. In [3] the definition of semantic encapsulation is given as a component that preserves a basic set of entailments of a term in an ontology. This leads to the partitioning of a large ontology into a collection of modules that are conservative extensions of one another. The relationship of conservative extensions between modules ensures that each module can be reused independent of the rest while retaining the original semantics of its contained terms. An algorithm for partitioning an OWL ontology into $\varepsilon$-connected ontologies is provided in detail in [3]. For modularization during ontology design, $\varepsilon$-connection is used to integrate existing disjoint OWL ontologies using link properties to describe how elements in the different $\varepsilon$-connected ontologies are related. There are, however, a number of restrictions for using link properties that limit the way modules can be designed and used. For instance, the URI for a class cannot be local (declared as a class in the source ontology) and foreign, a class cannot be declared as a subclass of a class in a foreign ontology, a property cannot be declared as a sub-relation of a foreign property, an individual cannot be declared as an instance of a foreign class, and a pair of individuals cannot instantiate a foreign property. Violating any of these conditions make the $\varepsilon$-connection ontology inconsistent. This results in link properties being incompatible with the owl:imports relation since the owl:imports relation essentially copies the axioms of all imported ontologies into the source ontology. The major limitation of using $\varepsilon$-connections is that the use of such modules as a means of refining an ontology (nonconservative extensions of a theory) becomes impossible. This is insufficient as ontology hierarchies in our repository are built around nonconservative extensions of theories allowing our repository to store extensions of the same theory with slightly different sets of ontological commitments.
Chapter 3

Verification of Linear Point

We begin by looking at the one of the most common representations of time, that is, as timepoints on an infinite line. In this chapter we illustrate our methodology of verifying theories by proving the set of representation theorems for the theory of timepoints. First, we introduce the theory of linear_point found in [14] and pose modifications to the original set of axioms as we discover unintended models allowed by the original axioms. Next, we introduce the class of mathematical structures for infinite_linear_ordering used to verify our time theory T_{linear_point}. As we move towards proving interpretability between the two theories, an interesting problem arises due to the fact that T_{linear_point} imposes class-restricted quantification over its relations while T_{infinite_linear_ordering} does not. In order to overcome this obstacle, we look at the work done in [21] and [6] for similar circumstances and discuss how to implement a solution as part of the translation definitions between the theories. Finally, we verify that the intended models of the theory of linear_point are exactly those that are captured by the axiomatization by proving the required representation theorems.

3.1 Theory of Linear Point (T_{linear_point})

The linear_point theory as presented in [14], is a simple ontology representing timepoints on a line. It contains a binary relation, before, that when used to relate elements of the timepoint class has the properties of transitivity, irreflexivity, and total order. Finally, two axioms state that timepoints extend the timeline infinitely in both directions.
Figure 3.1: \(\text{before}\) relation between \textit{timepoints} for models of \(T_{\text{linear point}}\)

\[
(\forall x, y, z) \text{timepoint}(x) \land \text{timepoint}(y) \land \text{timepoint}(z) \land
\text{before}(x, y) \land \text{before}(y, z) \supset \text{before}(x, z)
\]  
(3.1.1)

\[
(\forall x) \text{timepoint}(x) \supset \neg \text{before}(x, x)
\]  
(3.1.2)

\[
(\forall x, y, z) \text{timepoint}(x) \land \text{timepoint}(y) \supset \text{before}(x, y) \lor \text{before}(y, x) \lor (x = y)
\]  
(3.1.3)

\[
(\forall x) \text{timepoint}(x) \supset (\exists y) \text{timepoint}(y) \land \text{before}(x, y)
\]  
(3.1.4)

\[
(\forall x) \text{timepoint}(x) \supset (\exists y) \text{timepoint}(y) \land \text{before}(y, x)
\]  
(3.1.5)

In the above axiomatization, all the properties of the \(\text{before}\) relation such as transitivity, irreflexivity and total ordering apply only when the arguments of \(\text{before}\) belong to the \textit{timepoint} class. However, the axioms as they are also state that \(\text{before}\) is a relation between any element in the domain and not only \textit{timepoints}. Therefore, there exist models of this axiomatization where \(\text{before}\) is a relation between non-\textit{timepoint} elements. This poses potential problems when one wants to extend the ontology in such a way that more classes are introduced as those classes become valid arguments for the \(\text{before}\) relation.

In order to alleviate this problem, we introduce a sort axiom for the \(\text{before}\) relation restricting its arguments to the \textit{timepoint} class and rewrite the remaining axioms slightly to accommodate this new axiom. Therefore, in this theory, \(\text{before}\) is only a relation between \textit{timepoints}.

\[
(\forall x, y) \text{before}(x, y) \supset \text{timepoint}(x) \land \text{timepoint}(y)
\]  
(3.1.6)

\[
(\forall x, y, z) \text{before}(x, y) \land \text{before}(y, z) \supset \text{before}(x, z)
\]  
(3.1.7)
This new axiomatization allows greater flexibility when combining ontologies as its relations are now defined only for the set of intended elements in the domain. For example, the Process Specification Language (PSL) CORE ontology uses a version of the linear_point axioms that closely resembles this as it has other classes in its domain besides timepoints such as activities, activity_occurrences and objects. Thus, we will now refer to the theory consisting of axioms 3.1.6 to 3.1.11 as $T_{\text{linear point}}$.

### 3.2 Infinite Linear Ordering ($T_{\text{infinite linear ordering}}$)

To verify $T_{\text{linear point}}$, we use the class of mathematical structures $\mathcal{M}_{\text{infinite linear ordering}}$ for the representation theorems. First, we define the structure of its models and then proceed to introducing the axiomatization for those models. It is this set of axioms that play a crucial role in our ability to use an automated-theorem prover as part of our methodology.

**Definition 3.1** A linear ordering is a structure

\[ \mathcal{P} = (P, \leq) \]

such that all elements $P$ are mutually comparable under the relation $\leq$.

The axioms of $T_{\text{infinite linear ordering}}$ is taken from [20] and contains a single relation $\text{leq}$ for any two elements in the domain. The binary relation $\text{leq}$ is transitive, antisymmetric and total. This theory axiomatizes models of infinitely linearly ordered sets of elements. These axioms have been verified by mathematicians such that they capture exactly the class of structures described above. This theory will serve as the foundational theory by which other theories (such as linear_point) are verified against.

\[
(\forall x) \neg \text{before}(x, x) 
\]  
(3.1.8)  

\[
(\forall x, y, z) \text{before}(x, y) \lor \text{before}(y, x) \lor (x = y) 
\]  
(3.1.9)  

\[
(\forall x)(\exists y) \text{timepoint}(y) \land \text{before}(x, y) 
\]  
(3.1.10)  

\[
(\forall x)(\exists y) \text{timepoint}(y) \land \text{before}(y, x) 
\]  
(3.1.11)  

\[
(\forall x) \text{leq}(x, x) 
\]  
(3.2.1)
(∀x, y) leq(x, y) ∧ leq(y, x) ⊃ (x = y) \quad (3.2.2)

(∀x, y, z) leq(x, y) ∧ leq(y, z) ⊃ leq(x, z) \quad (3.2.3)

(∀x, y) lt(x, y) ≡ leq(x, y) ∧ ¬leq(y, x) \quad (3.2.4)

(∀x, y) lt(x, y) ∨ lt(y, x) ∨ (x = y) \quad (3.2.5)

(∀x)(∃y) lt(y, x) \quad (3.2.6)

(∀x)(∃y) lt(x, y) \quad (3.2.7)

### 3.3 Interpretations with Class-Restricted Quantification

Looking at the two theories described above, the theory of linear point contains one unary relation that defines a class of elements in its domain as timepoints that the before relation is valid over, while the theory of infinite linear ordering on the other hand defines no classes of elements, and so its leq relation is valid over the entire domain of elements. Defining translations between the relations of before and leq alone are insufficient in correctly mapping between the models due to the class-restricted nature of before. How then do we correctly compare these two types of theories using interpretations?

Looking at work done on characterizing interpretations between theories in [21] and [6], it is stated that a theory \( T_1 \) is interpretable in a theory \( T_2 \) if there exists, along with the translation \( \pi \), some fixed unary formula \( D(x) \) such that:

1. \( T_2 \models (\exists x)D(x) \)
2. \( T_2 \models (\pi(\varphi))^D \), for all formulas of \( \varphi \) of \( T_1 \), where \( T_1 \models \varphi \)

Where \( D(x) \) is a class of elements in \( T_2 \) for which translated \( T_1 \) sentences are quantified over. Meaning, \( T_2 \) entails relative to some class of elements in its domain \( D(x) \), some translation of \( T_1 \). Therefore, when specifying the translation definitions between theories with and without defined classes of elements we must first explicitly define a class of elements \( D(x) \) in the background theory that the translated axioms of the other theory are valid for. Finally, we must relativize the quantification of the target axiom according to \( D(x) \) when using a reasoning-engine to prove representation theorems.
3.4 Translating Between $T_{\text{linear point}}$ and $T_{\text{infinite linear ordering}}$

For the first set of translation definitions from $T_{\text{infinite linear ordering}}$ to $T_{\text{linear point}}$, we define $D$ as the unary relation called \textit{domain} and set it equivalent to the class of \textit{timepoints}. That is, we are claiming that the axioms of $T_{\text{infinite linear ordering}}$ are valid in the theory of \textit{linear point} over the class of \textit{timepoint} elements.

\textbf{Definition 3.2} Let $\Sigma_{lp\rightarrow ilo}$ be the set of translation definitions from $T_{\text{infinite linear ordering}}$ to $T_{\text{linear point}}$:

\begin{align*}
(\forall x, y) \text{leq}(x, y) &\equiv \text{before}(x, y) \lor (x = y) \\
(\forall x) \text{domain}(x) &\equiv \text{timepoint}(x)
\end{align*}

The set of translated axioms from $T_{\text{infinite linear ordering}}$ after relativization to $D$ are as follows:

\begin{align*}
(\forall x) \text{domain}(x) &\supset \text{leq}(x, x) & \text{(3.4.1)} \\
(\forall x, y) \text{domain}(x) \land \text{domain}(y) &\supset (\text{leq}(x, y) \land \text{leq}(y, x) \supset (x = y)) & \text{(3.4.2)} \\
(\forall x, y, z) \text{domain}(x) \land \text{domain}(y) \land \text{domain}(z) &\supset (\text{leq}(x, y) \land \text{leq}(y, z) \supset \text{leq}(x, z)) & \text{(3.4.3)} \\
(\forall x, y) \text{domain}(x) \land \text{domain}(y) &\supset (\exists y) \text{domain}(y) \land \text{lt}(y, x) \lor (x = y) & \text{(3.4.4)} \\
(\forall x) \text{domain}(x) &\supset (\exists y) \text{domain}(y) \land \text{lt}(y, x) & \text{(3.4.5)} \\
(\forall x) \text{domain}(x) &\supset (\exists y) \text{domain}(y) \land \text{lt}(y, x) & \text{(3.4.6)}
\end{align*}

In the opposite direction of translation, we are translating axioms from a theory with one defined class of elements to a theory without classes. A new unary relation for \textit{domain} to partition the set of elements in the background theory is not necessary as $D$ in this case encapsulates the entire set of elements in $T_{\text{infinite linear ordering}}$ when interpreting axioms of $T_{\text{linear point}}$. Instead, we assert as part of the translation definitions that all elements in the theory $T_{\text{infinite linear ordering}}$ are equivalent to the \textit{timepoint} class of $T_{\text{linear point}}$.

\textbf{Definition 3.3} Let $\Sigma_{ilo\rightarrow lp}$ be the set of translation definitions from $T_{\text{linear point}}$ to $T_{\text{infinite linear ordering}}$:

\begin{align*}
(\forall x, y) \text{before}(x, y) &\equiv \text{leq}(x, y) \land \neg\text{leq}(y, x) \\
(\forall x) \text{timepoint}(x) &\equiv (x = x)
\end{align*}
3.5 Representation Theorem for $T_{\text{linear point}}$

The intended models of $\text{linear point}$ are those of an infinite timeline of $\text{timepoints}$ where any two $\text{timepoints}$ are comparable via the $\text{before}$ relation. If we say that the intended semantics of the temporal relation $\text{before}$ is equivalent to the mathematical ordering relation $<$, we can verify this by determining if the axioms of $\text{linear point}$ can be represented by the class of mathematical structures of $\text{infinite linear ordering}$.

To verify $T_{\text{linear point}}$, we first prove the reasoning tasks that instantiate $(\text{Rep} - 1)$ and $(\text{Rep} - 2)$:

**Theorem 3.1** $T_{\text{linear point}}$ is definably equivalent to $T_{\text{infinite linear ordering}}$

**Proof:** Using Prover9, we have shown that

$$T_{\text{linear point}} \cup \Sigma_{lp,ilo} \models T_{\text{infinite linear ordering}}$$

and

$$T_{\text{infinite linear ordering}} \cup \Sigma_{ilo,lp} \models T_{\text{linear point}}$$

$\square$

The last step of verifying $T_{\text{linear point}}$ is defining the class of intended models:

**Definition 3.4** $\mathcal{M}_{\text{linear point}}$ is the following class of structures: $\mathcal{M} \in \mathcal{M}_{\text{linear point}}$ iff

1. $\mathcal{M} \models \mathcal{P}$, where $\mathcal{P} = \langle P, \leq \rangle$ is a linear ordering
2. $\langle t \rangle \in \text{timepoint}$ iff $t \in P$;
3. $\langle t_1, t_2 \rangle \in \text{before}$ iff $t_1 < t_2$.

We can now state the Representation Theorem for $T_{\text{linear point}}$:

**Theorem 3.2** $\mathcal{M} \in \mathcal{M}_{\text{linear point}}$ iff $\mathcal{M} \in \text{Mod}(T_{\text{linear point}})$.

**Proof:** This follows from Theorem 2.1 and Theorem 3.1, together with the fact that $T_{\text{infinite linear ordering}}$ axiomatizes the class of infinite linear orderings. $\square$
3.6 Summary

We have verified that the models of the time theory $T_{\text{linear,point}}$ is represented by the class of structures of \textit{infinite linear ordering} using the methodology outlined in the previous chapter. During the process, we have made a set of necessary changes to the original axiomatization found in [14] such that the revised axioms capture only the set of intended models of the ontology, and discussed the issue of defining translation definitions between theories with and without class-quantified relations. The changes made to the axioms also make the theory more suitable for reuse as we have restricted the relations of the theory to apply only over the class of objects that they are intended to relate. To overcome the issue of interpreting between a theory with class-quantified relations and one without, we discussed the use of a unary \textit{domain} relation incorporated into the translation definitions used to relativize the original queries (i.e. axioms). Incorporating this step into our procedure allowed for the continued use of automated reasoners in finding the necessary proofs.
Chapter 4

Integrating Time Interval Ontologies

Representing time as intervals allows for flexibility when defining the set of relations between those intervals. This leads to the existence of various axiomatizations of time as intervals. Since intervals can be different sizes (as opposed to points), it is possible to use relations such as inclusion or overlap in addition to precedence. Alternatively, one could choose to model intervals by the points at which they meet. This leads to there being different hierarchies of theories that all represent time as intervals. Therefore, the key objective in this chapter is to explore and determine the relationships both among theories in the same hierarchy and across the different hierarchies. The set of three time interval theories investigated in this chapter come from [14] and [22], and while these axiomatizations were published almost two decades ago, the relationships among them remain unclear. We continue to use our methodology for identifying the relationships between theories in an attempt to determine what the different ontological commitments made by each set of theories are, whether there exist a common set of models between them, how we can axiomatize those models, and also how we can extend a theory from one hierarchy to be equivalent to one from another. By doing so we aim to provide a rich network of proven relationships between the hierarchies of time interval theories in order to better integrate them by understand their similarities and differences.

First, we introduce the three different hierarchies and each of the contained theories as we establish the relationships between theories within the same hierarchy. Next, we proceed to exploring the relationships shared between theories across the hierarchies as we introduce the different sets of translation definitions. Throughout the process of proving relationships, we also elaborate on the finer details,
challenges and workarounds when using an automated reasoner as part of the methodology.

4.1 Theories of Time Intervals

In this section, we introduce the three hierarchies of time interval ontologies that are the focus of this chapter –

- the hierarchy $H_{Periods}$, whose theories were introduced in [22],
- the hierarchy $H_{Approximate\text{-}\text{Point}}$ presented in [14],
- the hierarchy $H_{Interval\text{-}\text{Meeting}}$, which has been explored in [1], [14], and [17].

With respect to relationships between these time ontologies, Ladkin [17] fully characterizes the models of another of Hayes’ time interval theories, $T_{interval\text{-}\text{meeting}}$, that uses the meets relation. In doing so he is able to extend the theory to one that is equivalent to $T_{INT}(Q)$. However, the process is manual and only proof sketches are provided in comparison to the semi-automated procedure described in this paper that utilizes an automated theorem prover to provide detailed machine proofs. Although we may later chose to leverage this proof as a form of verification, we will focus on discovering new relationships between these ontologies using our methodology.

4.1.1 Theories of the Hierarchy $H_{Periods}$

Here we introduce the four first-order theories in the hierarchy $H_{Periods}$ (see Figure 4.1) whose axioms were provided by van Benthem in [22]. We begin by describing the weakest theory of $T_{periods}$ and then work our way up the hierarchy by looking at each of its extensions.

Minimal Theory of Periods ($T_{periods}$)

This theory constitutes the minimal set of conditions that must be met by any period structure [22] and has two relations (precedence and inclusion) and two conservative definitions (glb and overlaps) as its non-logical lexicon. A total of eight axioms are contained within this theory. Transitivity and irreflexivity axioms for the precedence relation make it a strict partial order, and transitivity, reflexivity, and
anti-symmetry axioms for the inclusion relation make it a partial order. While the axioms of mono-
tonicity enforce correct interplay between the precedence and inclusion relations. Van Benthem further
includes in this minimal theory, an axiom that guarantees the existence of greatest lower bounds (\textit{glb})
between overlapping intervals. No classes of elements are specified; hence, every element in the domain
is considered a \textit{timeinterval}. Also, the axioms of this theory are satisfied by both infinite and finite
models.

\[
(\forall x, y, z) \text{precedence}(x, y) \land \text{precedence}(y, z) \supset \text{precedence}(x, z) \tag{4.1.1.1}
\]

\[
(\forall x) \neg \text{precedence}(x, x) \tag{4.1.1.2}
\]

\[
(\forall x, y, z) \text{inclusion}(x, y) \land \text{inclusion}(y, z) \supset \text{inclusion}(x, z) \tag{4.1.1.3}
\]

\[
(\forall x) \text{inclusion}(x, x) \tag{4.1.1.4}
\]

\[
(\forall x, y) \text{inclusion}(x, y) \land \text{inclusion}(y, x) \supset (x = y) \tag{4.1.1.5}
\]

\[
(\forall x, y) \text{overlaps}(x, y) \equiv (\exists z) \text{inclusion}(z, x) \land \text{inclusion}(z, y) \tag{4.1.1.6}
\]

\[
(\forall x, y, z) \text{glb}(x, y, z) \equiv \text{inclusion}(z, x) \land \text{inclusion}(z, y) \land
\]

\[
(\forall u) (\text{inclusion}(u, x) \land \text{inclusion}(u, y) \supset \text{inclusion}(u, z)) \tag{4.1.1.7}
\]

\[
(\forall x, y) \text{overlaps}(x, y) \supset (\exists z) \text{glb}(x, y, z) \tag{4.1.1.8}
\]

\[
(\forall x, y, z) \text{precedence}(x, y) \land \text{inclusion}(z, x) \supset \text{precedence}(z, y) \tag{4.1.1.9}
\]
(∀x, y, z) precedence(x, y) ∧ inclusion(z, y) ⊃ precedence(x, z) \hspace{1cm} (4.1.1.10)

Figure 4.2: Interval \(k\) exists as the greatest lower-bound between overlapping intervals \(i\) and \(j\) enforced by Axiom 4.1.1.8.

**Theory of Mixed Periods (\(T_{\text{mixed\_periods}}\))**

The theory of mixed periods extends \(T_{\text{periods}}\) with eleven more axioms that force an infinite linear ordering of intervals at both ends of the timeline, the joining and interaction of neighbouring intervals, that intervals are uninterrupted stretches, the relationship between overlap and inclusion of intervals, and the existence of a larger interval that covers any pair of intervals. The lexicon is expanded with the addition of two new conservative definitions (\(\text{lub} \) and \(\text{underlaps}\)), but no new relations. This theory refines the models of time intervals from \(T_{\text{periods}}\) in order to capture further intuitions about time, but without enforcing density or discreteness of intervals.

(∀x)(∃y) precedence(x, y) \hspace{1cm} (4.1.1.11)

(∀x)(∃y) precedence(y, x) \hspace{1cm} (4.1.1.12)

(∀x, y) precedence(x, y) ⊃

(∃w) precedence(x, w) ∧ ¬((∃z)precedence(x, z) ∧ precedence(z, w)) \hspace{1cm} (4.1.1.13)

(∀x, y) precedence(y, x) ⊃

(∃w) precedence(w, x) ∧ ¬((∃z)precedence(w, z) ∧ precedence(z, x)) \hspace{1cm} (4.1.1.14)

(∀x, y, z) lub(x, y, z) ≡ inclusion(x, z) ∧ inclusion(y, z) ∧

(∀u) (inclusion(x, u) ∧ inclusion(y, u) ⊃ inclusion(z, u)) \hspace{1cm} (4.1.1.15)
(∀x, y) underlaps(x, y) ≡ (∃z) inclusion(x, z) ∧ inclusion(y, z) \hspace{1cm} (4.1.1.16)

(∀x, y) underlaps(x, y) ⊃ (∃z) lub(x, y, z) \hspace{1cm} (4.1.1.17)

(∀x, y) ¬inclusion(x, y) ⊃ (∃z) inclusion(z, x) ∧ ¬overlaps(z, y) \hspace{1cm} (4.1.1.18)

(∀x, y)(∃z) inclusion(x, z) ∧ inclusion(y, z) \hspace{1cm} (4.1.1.19)

(∀x, y, z) precedence(x, y) ∧ precedence(z, y) ⊃ (∃w) glb(x, z, w) ∧ precedence(w, y) \hspace{1cm} (4.1.1.20)

(∀x, y, z) precedence(y, x) ∧ precedence(y, z) ⊃ (∃w) glb(x, z, w) ∧ precedence(y, w) \hspace{1cm} (4.1.1.21)

(∀x, y) precedence(x, y) ∨ precedence(y, x) ∨ overlaps(x, y) \hspace{1cm} (4.1.1.22)

(∀x, y, z) precedence(x, y) ∧ precedence(y, z) ⊃ (∀u) (inclusion(x, u) ∧ inclusion(z, u) ⊃ inclusion(y, u)) \hspace{1cm} (4.1.1.23)

**Theory of Intervals Over Rational Numbers (T_{INT(Q)})**

The theory T_{INT(Q)} is the first-order axiomatization of rational intervals where the only countable model (up to isomorphism) is characterized by the set of intervals that exist between the rational numbers [22]. This theory extends the T_{mixed,pertods} by adding two axioms – one for density, that states that any interval can be divided into two smaller intervals, and a second axiom enforcing Allen’s temporal relations that characterize all the possible orientations of any two intervals.

The axiom enforcing density of intervals is:

(∀x)(∃y_1, y_2) precedence(y_1, y_2) ∧ lub(y_1, y_2, x) \hspace{1cm} (4.1.1.24)

and the axiom characterizing the orientation of any two intervals is:

(∀x, y) overlaps(x, y) ⊃ (x = y) \hspace{1cm} ∨

(inclusion(x, y) ∧ (∃z_1)(precedence(x, z_1) ∧ lub(x, z_1, y) ∨ (∃z_2)(precedence(z_2, x) ∧ lub(z_2, x, y))) ∨

(∃z_3, z_4, z_5)(precedence(z_3, x) ∧ precedence(x, z_4) ∧ lub(z_3, x, z_5) ∧ lub(z_5, z_4, y))) \hspace{1cm} ∨

(inclusion(y, x) ∧ (∃u_1)(precedence(y, u_1) ∧ lub(y, u_1, x) ∨ (∃u_2)(precedence(u_2, y) ∧ lub(u_2, y, x))) ∨

(∃u_3, u_4, u_3)(precedence(u_3, y) ∧ precedence(y, u_4) ∧ lub(u_3, y, u_3) ∧ lub(u_3, u_4, x))) \hspace{1cm} ∨
Theory of Intervals Over Integers ($T_{\text{INT}}(\mathbb{Z})$)

The theory of $T_{\text{INT}}(\mathbb{Z})$, on the other hand, is the first-order axiomatization of discrete intervals whose models are the sets of intervals over the integers. The axiomatization provided by van Benthem includes the second-order principle of well-foundedness. However, he conjectures that this theory is elementary equivalent to one in which the principle of well-foundedness is replaced by an axiom for atomicity of intervals (ATOM). This modified theory constitutes the theory of discrete periods $T_{\text{INT}}(\mathbb{Z})$. The theories $T_{\text{INT}}(\mathbb{Q})$ and $T_{\text{INT}}(\mathbb{Z})$ both extend $T_{\text{mixed, periods}}$, but are inconsistent with one another as one takes on the ontological commitment of density for its models of intervals and the other the commitment of discreteness.

The axiom for the atomicity of intervals:

$$(\forall x)(\exists y) \text{inclusion}(y, x) \land (\forall z) (\text{inclusion}(z, y) \supset (z = y))$$

\[ (4.1.1.26) \]

4.1.2 Theories of the Hierarchy $\mathbb{H}_{\text{Approximate-Point}}$

The ontologies in the hierarchy $\mathbb{H}_{\text{Approximate-Point}}$ (see Figure 4.3) represent intervals as approximations of points. As presented in [14], the hierarchy consists of three theories; once again, we begin by examining the axioms of the weakest theory and then move on to its extensions.

Theory of Approximate-Point ($T_{\text{ap}}$)

Hayes’ first-order theory $T_{\text{ap}}$ consists of two relations, where $\text{precedes}$ is the relation between sufficiently distinct intervals, and where $\text{finer}$ is the relation of sub-intervals. The $\text{precedes}$ relation is a strict partial ordering, while the $\text{finer}$ relation is a partial ordering. There also exists the $\text{ncdf}$ (not clearly distinguishable from) relation that is a conservative definition relating two intervals that share a common
interval (i.e. overlapping intervals). Transitivity and irreflexivity axioms exist for the precedes relation, while transitivity, reflexivity, and anti-symmetry axioms exist for the finer relation.

\[(\forall x, y, z) \text{finer}(x, y) \land \text{finer}(y, z) \supset \text{finer}(x, z)\]  
(4.1.2.1)

\[(\forall x) \text{finer}(x, x)\]  
(4.1.2.2)

\[(\forall x, y) \text{finer}(x, y) \land \text{finer}(y, x) \supset (x = y)\]  
(4.1.2.3)

\[(\forall x, y) \text{finer}(x, y) \supset \lnot \text{precedes}(x, y)\]  
(4.1.2.4)

\[(\forall x, y)(\exists z) \text{finer}(x, z) \land \text{finer}(y, z)\]  
(4.1.2.5)

\[(\forall x, y, z) \text{precedes}(x, y) \land \text{precedes}(y, z) \supset \text{precedes}(x, z)\]  
(4.1.2.6)

\[(\forall x) \lnot \text{precedes}(x, x)\]  
(4.1.2.7)

\[(\forall x)(\exists y) \text{precedes}(x, y)\]  
(4.1.2.8)

\[(\forall x)(\exists y) \text{precedes}(y, x)\]  
(4.1.2.9)

\[(\forall x, y) \text{ncdf}(x, y) \equiv (\exists z) \text{finer}(z, x) \land \text{finer}(z, y)\]  
(4.1.2.10)
\[(\forall x, y) \text{ precedes}(x, y) \lor \text{ precedes}(y, x) \lor \text{ ncdf}(x, y)\]  
\[(\forall x, y, z) \text{ finer}(x, y) \land \text{ precedes}(y, z) \supset \text{ precedes}(x, z)\]

**Theory of Dense Approximate-Point** \((T_{\text{dense, ap}})\)

Since the theory \(T_{\text{ap}}\) enforces neither density or discreteness for elements in its domain (time intervals), this theory \(T_{\text{dense, ap}}\) exists as an extension of \(T_{\text{ap}}\) that adds an axiom to enforce density of time intervals. This axiom states that \text{finer} intervals always exist.

\[(\forall x)(\exists y) \text{ finer}(y, x) \land \lnot \text{ finer}(x, y)\]

**Theory of Discrete Approximate-Point** \((T_{\text{discrete, ap}})\)

In this extension of \(T_{\text{ap}}\), two conservative definitions are added – one for the \text{meets} relation, that states that an interval \text{meets} another if there does not exist an interval between them, and the other for the \text{moment} class of elements that defines them as the smallest interval (an interval is a \text{moment} iff it contains no smaller intervals). The axiom for discreteness then stipulates that every \text{moment} \text{meets} another \text{moment}, both in its past and in its future.

\[(\forall x, y) \text{ meets}(x, y) \equiv \text{ precedes}(x, y) \land \lnot((\exists z)\text{ precedes}(x, z) \land \text{ precedes}(z, y))\]

\[(\forall x) \text{ moment}(x) \equiv \lnot((\exists y)\text{ finer}(y, x) \land \lnot(x = y))\]

\[(\forall x)(\exists y) \text{ meets}(x, y) \land \text{ moment}(y)\]

\[(\forall x)(\exists y) \text{ meets}(y, x) \land \text{ moment}(y)\]

**4.1.3 Theories of the Hierarchy** \(\mathbb{H}_{\text{Interval-Meeting}}\)

The ontologies in the hierarchy \(\mathbb{H}_{\text{Interval-Meeting}}\) (see Figure 4.5) represent intervals by the points at which they meet. These theories are collected from [14], [1], and [17]. We begin by examining the axioms of the weakest theory and then move on to its extensions.
Figure 4.5: Hierarchy of Interval-Meeting theories from [14], [1], and [17]. Dashed-lines denote non-conservative extensions.

**Theory of Interval-Meeting (T_{im})**

The theory of *interval meeting* axiomatizes models of *timeintervals* that meet each other at points. The lexicon consists of a unary relation *timeinterval* that defines a class of elements in the domain, one binary *meets* relation over that class of elements, four conservative definitions using the *meets* relation (*starts*, *during*, *finishes*, *moment*), and a *plus* function. The axioms enforce an infinite linear ordering between *timeintervals* extending in both directions of the timeline, the uniqueness of points at which *timeintervals* meet, and the existence of larger *timeintervals* that result from a union of two adjacent ones. The conservative definitions of *starts*, *finishes*, and *during* specify the orientation of two overlapping intervals, while *moment* defines a subclass of atomic *timeintervals*.

\[
(\forall i, j) \text{meets}(i, j) \supset \text{timeinterval}(i) \land \text{timeinterval}(j) \tag{4.1.3.1}
\]

\[
(\forall i, j, k, m) \text{meets}(i, k) \land \text{meets}(j, k) \land \text{meets}(i, m) \supset \text{meets}(j, m) \tag{4.1.3.2}
\]
(∀i, j, k, l) meets(i, j) ∧ meets(k, l) ⊃

meets(i, l) ∨ (∃n) (meets(i, n) ∧ meets(n, l)) ∨ (meets(k, n) ∧ meets(n, j))  

(∀i, j) meets(i, j) ⊃ ¬meets(j, i)  

(∀i, j, k, m) meets(i, j) ∧ meets(j, k) ∧ meets(k, m) ⊃

meets(i, plus(j, k)) ∧ meets(plus(j, k), m)  

(∀i, j, k) moment(i) ≡ (i ≠ plus(j, k))  

(∀i)(∃j, k) meets(j, i) ∧ meets(i, k)  

(∀i, j) starts(i, j) ≡ (∃k, m, n) meets(i, m) ∧ meets(m, n) ∧ meets(k, j) ∧ meets(j, n)  

(∀i, j) during(i, j) ≡

(∃k, m, n, o) meets(m, i) ∧ meets(i, n) ∧ meetgs(n, o) ∧ meets(k, j) ∧ meets(j, o)  

(∀i, j) finishes(i, j) ≡ (∃k, m, n) meets(m, i) ∧ meets(i, n) ∧ meets(j, k) ∧ meetgs(j, n)  

The original set of axioms provided in [14] includes axiom with a plus function that is responsible for creating an inconsistency. That axiom entails that all time intervals in the domain meet each other, causing an inconsistency with axiom 4.1.3.4 that prevents such cycles from existing. Instead we will replace axiom 4.1.3.5 with an equivalent existential form from [17], thereby removing the plus function. With this, the definitional axiom for moment is rewritten to accommodate the removal of plus.

The new axioms replacing 4.1.3.5 and 4.1.3.6 are:

(∀i, j, k, m) meets(i, j) ∧ meets(j, k) ∧ meets(k, m) ⊃ (∃n) meets(i, n) ∧ meets(n, m)  

(∀i, j, k, m) moment(n) ≡ meets(i, n) ∧ meets(n, m) ∧

¬(∃k) (meets(i, j) ∧ meets(j, k) ∧ meets(k, m))
The Theory of Allen-Hayes ($T_{\text{allen\_hayes}}$)

This theory from [1] extends the theory of $T_{\text{interval\_meeting}}$ with the axiom:

$$(\forall p q r s) \text{meets}(p, q) \land \text{meets}(q, s) \land \text{meets}(p, r) \land \text{meets}(r, s) \supset (q = r) \quad (4.1.3.13)$$

**Lemma 4.1** $T_{\text{allen\_hayes}}$ is a nonconservative extension of $T_{\text{interval\_meeting}}$.

**Proof:** Let $T_{\text{finite\_interval\_meeting}}$ be the subtheory of $T_{\text{interval\_meeting}}$ without the axioms that force the existence of infinite sets of intervals. Using Mace, one can construct a model $M$ of $T_{\text{finite\_interval\_meeting}}$ that falsifies the additional axiom. This model can be extended to construct a model of $T_{\text{interval\_meeting}}$ that falsifies this axiom. □

This additional axiom of this theory removes the possibility of duplicate intervals in satisfying models of the theory.

The Theory of Ladkin-INT($\mathbb{Q}$) ($T_{\text{ladkin\_intq}}$)

This theory characterized by Ladkin in [17] nonconservatively extends the theory of $T_{\text{allen\_hayes}}$ such that it axiomatizes $Th(INT(\mathbb{Q}))$ (i.e. the theory of rational intervals where the models are characterized by the set of intervals that exist between the rational numbers). This theory includes a conservative definition of $\text{equiv}$ that is a definable relation with four interval arguments to define an equivalence relation on pairs of intervals. As points can be thought of as a definable equivalence relation on pairs of intervals, $\text{equiv}$ is effectively an equivalence relation over points. The relation $\text{pointless}$ is an ordering relation over these points, and axiom 4.1.3.16 ensures that these points are densely ordered.

$$(\forall p, q, r, s) \text{equiv}(p, q, r, s) \equiv \text{meets}(p, q) \land \text{meets}(r, s) \land \text{meets}(p, s) \quad (4.1.3.14)$$

$$(\forall p, q, r, s) \text{pointless}(p, q, r, s) \equiv (\exists u, v, w)\text{equiv}(p, q, u, v) \land \text{equiv}(r, s, v, w) \quad (4.1.3.15)$$

$$(\forall p, q, r, s) \text{pointless}(p, q, r, s) \supset (\exists x, y)\text{pointless}(p, q, x, y) \land \text{pointless}(x, y, r, s) \quad (4.1.3.16)$$
4.2 Relationships between \( \mathbb{H}_{\text{Approximate-Point}} \) and \( \mathbb{H}_{\text{Periods}} \)

First we explore the relationship shared between theories across the hierarchies of \( \mathbb{H}_{\text{Approximate-Point}} \) and \( \mathbb{H}_{\text{Periods}} \). We identify the translation definitions between their root theories and proceed to determine the relationship of definable interpretation between theories across hierarchies and ultimately find if there exists those that are definably equivalent as a way to draw similarities between the hierarchies (see Figure 4.7).

![Figure 4.7: Relationships between theories of \( \mathbb{H}_{\text{Approximate-Point}} \) and \( \mathbb{H}_{\text{Periods}} \). Dashed-lines denote nonconservative extensions, double-arrowed solid-lines denote definable equivalence, clouds define the boundaries of each hierarchy, bolded theories are new additions to the hierarchy.]

With the addition of new extensions for the \( \mathbb{H}_{\text{Approximate-Point}} \) found through the translation of \( \mathbb{H}_{\text{Periods}} \) theories, we can better understand the relationships between the original theories. For instance, if we look at both the original hierarchies (see Figure 4.7) we notice that for each there exists dense and discrete interval theories (i.e \( T_{\text{INT}(\mathbb{Z})} \) and \( T_{\text{INT}(\mathbb{Q})} \) in \( \mathbb{H}_{\text{Periods}} \), \( T_{\text{dense\_ap}} \) and \( T_{\text{discrete\_ap}} \) in \( \mathbb{H}_{\text{Approximate-Point}} \)). However, the exact differences between them is initially unclear. Although we
know $T_{INT(Q)}$ is stronger than $T_{dense, ap}$, we want to figure out exactly what their differences in ontological commitments are. By identifying the new extension, $T_{ap,interval}$, in $\mathbb{H}_{Approximate-Point}$ and realizing that $T_{INT(Q)}$ is equivalent to $T_{ap,interval}$ extended with the density axiom of the original theory $T_{dense, ap}$, we have isolated the difference in ontological commitments between the original theories of dense intervals, $T_{INT(Q)}$ and $T_{dense, ap}$, to the set of axioms used to extend $T_{ap}$ to $T_{ap,interval}$. Thus, now not only do we know that $T_{INT(Q)}$ is a stronger theory than $T_{dense, ap}$, but we also know the set of ontological commitments that make up their difference. By examining the remaining relationships found between theories of these two hierarchies we can identify similar results between the rest of the original theories.

### 4.2.1 Translating between $\mathbb{H}_{Approximate-Point}$ and $\mathbb{H}_{Periods}$

The translation definitions, $\Sigma_{p, ap}$, between $\mathbb{H}_{Approximate-Point}$ and $\mathbb{H}_{Periods}$ are straight-forward as they use the same set of relations to axiomatize time intervals. In more complex cases, there would exist a different set of definitions for each direction of translation between core hierarchies, but the one-to-one mapping of relations here makes $\Sigma_{p, ap} \equiv \Sigma_{ap, p}$. These translation definitions are required to relate the two ontologies by finding the similarities, differences, and relative strengths of each of their modules.

When translating between ontologies, relations that have conservative definitions in the ontology do not require explicit translation definitions of their own since their definitional axioms can be reused in conjunction with the existing translation definitions. However, the simple translation definition between the conservative definitions of overlaps from $\mathbb{H}_{Periods}$ and $ncdf$ from $\mathbb{H}_{Approximate-Point}$ are added to the translation definitions in this case to increase the efficiency of the automated theorem prover.

**Definition 4.1** The translation definitions $\Sigma_{p, ap}$ for the interpretation of theories in $\mathbb{H}_{Approximate-Point}$ to theories in $\mathbb{H}_{Periods}$ is the set of sentences:

- $(\forall x, y) precedence(x, y) \equiv precedes(x, y)$
- $(\forall x, y) inclusion(x, y) \equiv finer(x, y)$
- $(\forall x, y) overlaps(x, y) \equiv ncdf(x, y)$

The translation definitions $\Sigma_{ap, p}$ for the interpretation of theories in $\mathbb{H}_{Periods}$ to theories in $\mathbb{H}_{Approximate-Point}$ is equivalent to $\Sigma_{p, ap}$. 
4.2.2 Relationship between $T_{periods}$ and $T_{ap}$

The first relationship to be determined between these two hierarchies will be between their weakest theories, $T_{periods}$ and $T_{ap}$.

**Lemma 4.2** $T_{periods} \cup \Sigma_{p,ap} \nvdash T_{ap}$

**Proof:** Using Mace, one can construct models of $T_{periods} \cup \Sigma_{p,ap}$ that falsify the $T_{ap}$ axioms (4.1.2.5), (4.1.2.8), (4.1.2.9), (4.1.2.11). □

**Lemma 4.3** $T_{ap} \cup \Sigma_{ap,p} \nvdash T_{periods}$

**Proof:** Let $T_{ap}^{finite}$ be the subtheory of $T_{ap}$ without the axioms that force the existence of infinite sets of intervals.

Using Mace, one can construct a model $\mathcal{M}$ of $T_{ap}^{finite} \cup \Sigma_{ap,p}$ that falsifies the axiom (4.1.1.8) of $T_{periods}$. This model can be extended to construct a model of $T_{ap} \cup \Sigma_{ap,p}$ that falsifies the axiom (4.1.1.8). □

We can use these results to identify the subtheory $T_{ap,root}^{1}$ of $T_{ap}$ that is definably equivalent to the subtheory $T_{periods,root}^{2}$ of $T_{periods}$.

**Theorem 4.1** $T_{ap,root}$ is definably equivalent to $T_{periods,root}$.

These new root theories axiomatize a subset of models shared by both ontologies.

4.2.3 Relationship between $T_{mixed,periods}$ and $T_{ap}$

Intuitively, the above results show that $T_{periods}$ is not strong enough to definably interpret $T_{ap}$, so we move to $T_{mixed,periods}$, which is the next (stronger) theory within the hierarchy $\mathbb{H}_{Periods}$. We use Prover9 to show:

**Lemma 4.4** $T_{mixed,periods} \cup \Sigma_{p,ap} \models T_{ap}$

Since $T_{mixed,periods}$ is an extension of $T_{periods}$, we have

---

1containing axioms: (4.1.2.1), (4.1.2.2), (4.1.2.3), (4.1.2.4), (4.1.2.6), (4.1.2.7), (4.1.2.12).

2containing axioms: (4.1.1.1), (4.1.1.2), (4.1.1.3), (4.1.1.4), (4.1.1.5), (4.1.1.9), (4.1.1.10).
Lemma 4.5 \( \mathcal{T}_{ap} \cup \Sigma_{ap,p} \not\models T_{\text{mixed,periods}} \)

Nevertheless, we can specify an extension \( \mathcal{T}_{ap,\text{interval}} \) of \( \mathcal{T}_{ap} \) such that

\[
\mathcal{T}_{ap,\text{interval}} \cup \Sigma_{ap,p} \models T_{\text{mixed,periods}}
\]

\[
T_{\text{mixed,periods}} \cup \Sigma_{p,ap} \models \mathcal{T}_{ap,\text{interval}}
\]

This is done by taking the axioms of \( T_{\text{mixed,periods}} \) not entailed by \( \mathcal{T}_{ap} \), translating them into the language of \( \mathcal{T}_{ap} \) via the translation definitions \( \Sigma_{ap,p} \), and extending \( \mathcal{T}_{ap} \) with this set of axioms. Therefore, we have created a new theory in the hierarchy \( \mathbb{H}_{\text{Approximate-Point}} \) such that

Theorem 4.2 \( \mathcal{T}_{ap,\text{interval}} \) is definably equivalent to \( T_{\text{mixed,periods}} \)

4.2.4 Relationship between \( T_{\text{INT}(Q)} \) and \( T_{\text{dense,ap}} \)

In comparing the two dense extensions of \( \mathbb{H}_{\text{Periods}} \) and \( \mathbb{H}_{\text{Approximate-Point}} \) the proofs become less trivial as the axioms enforcing density of time intervals differ between the two ontologies.

Lemma 4.6 \( T_{\text{INT}(Q)} \cup \Sigma_{p,ap} \models T_{\text{dense,ap}} \)

Proof: The density axiom (4.1.2.13) of \( T_{\text{dense,ap}} \) is proven using the subset of axioms of \( T_{\text{INT}(Q)} \):
(4.1.1.2), (4.1.1.10), (4.1.1.15), (4.1.1.24). \( \square \)

Lemma 4.7 \( T_{\text{dense,ap}} \cup \Sigma_{ap,p} \not\models T_{\text{INT}(Q)} \)

Proof: Using Mace, one can construct a model \( \mathcal{M} \) of \( T_{\text{finite}} \cup \Sigma_{ap,p} \) that falsifies Axiom (4.1.1.23) enforcing the convexity of intervals of \( T_{\text{INT}(Q)} \). This model can be extended to construct a model of \( T_{\text{ap,dense}} \cup \Sigma_{ap,p} \) that falsifies the axiom. \( \square \)

In a similar fashion as Theorem 4.2, we can extend \( T_{\text{dense,ap}} \) and specify a new theory in the hierarchy \( \mathbb{H}_{\text{Approximate-Point}} \) which is definably equivalent to \( T_{\text{INT}(Q)} \):

Theorem 4.3 \( T_{\text{ap,rational}} = T_{\text{dense,ap}} \cup T_{\text{ap, interval}} \) is definably equivalent to \( T_{\text{INT}(Q)} \)
Proof: Prover9 can be used to show that:

\[ T_{dense, ap} \cup \Sigma_{ap, p} \models Axiom(4.1.1.24) \]

so that we have:

\[ T_{ap, rational} \cup \Sigma_{ap, p} \models T_{INT(Q)} \]

\[ \square \]

4.2.5 Relationship between \( T_{INT(Z)} \) and \( T_{discrete, ap} \)

Finally we look at the relationship between the remaining two theories of either hierarchy, both of which are theories of intervals with discrete models.

Theorem 4.4 \( T_{INT(Z)} \cup \Sigma_{p, ap} \not\models T_{discrete, ap} \)

Proof: Let \( T_{INT(Z)}^{finite} \) be the subtheory of \( T_{INT(Z)} \) without the axioms that force the existence of infinite sets of intervals.

Using Mace, one can construct a model \( M \) of \( T_{INT(Z)}^{finite} \cup \Sigma_{ap, p} \) that falsifies Axiom (4.1.2.16) of \( T_{discrete, ap} \). This model can be extended to construct a model of \( T_{INT(Z)} \cup \Sigma_{ap, p} \) that falsifies the axiom. \( \square \)

Lemma 4.8 \( T_{discrete, ap} \cup \Sigma_{ap, p} \not\models T_{INT(Z)} \)

Proof: Using Mace, one can construct a model \( M \) of \( T_{ap}^{finite} \cup \Sigma_{ap, p} \) that falsifies the axiom used in the proof of Lemma 4.7, which is also entailed by \( T_{INT(Q)} \). This model can be extended to construct a model of \( T_{ap, discrete} \cup \Sigma_{ap, p} \) that falsifies the axiom. \( \square \)

Since the set of axioms not entailed by \( T_{discrete, ap} \) are the same ones used to extend \( T_{ap} \) to \( T_{ap, interval} \), we can specify a new theory, \( T_{ap, integer} \), in the same manner as Theorem 4.3 such that:

Theorem 4.5 \( T_{ap, integer} = T_{discrete, ap} \cup T_{ap, interval} \) is definably equivalent to \( T_{INT(Z)} \)

Proof: Prover9 can be used to show that:

\[ T_{discrete, ap} \cup \Sigma_{ap, p} \models Axiom(4.1.1.26) \]
so that we have:

$$T_{ap\text{.integer}} \cup \Sigma_{ap\text{.p}} \models T_{INT(\mathbb{Z})}$$


### 4.3 Relationships between $\mathbb{H}_{Interval-Meeting}$ and $\mathbb{H}_{Approximate-Point}$

Finally, we consider the case of integrating the hierarchy $\mathbb{H}_{Interval-Meeting}$. To do so we identify the translation definitions between $T_{interval\text{.meeting}}$ and the theories in the hierarchy $\mathbb{H}_{Approximate-Point}$ as we attempt to find definably equivalent theories (see Figure 4.8). Since the relationships between theories in the hierarchies $\mathbb{H}_{Approximate-Point}$ and $\mathbb{H}_{Periods}$ have already been established, we can determine the relationship between theories in $\mathbb{H}_{Interval-Meeting}$ and $\mathbb{H}_{Approximate-Point}$ by composition of the translation definitions.

![Figure 4.8: Relationships between theories of $\mathbb{H}_{Interval-Meeting}$ and $\mathbb{H}_{Approximate-Point}$. Dashed-lines denote nonconservative extensions, single-arrowed solid-lines denote definable interpretation, clouds define the boundaries of each hierarchy, bolded theories are new additions to the hierarchy.](image)

4.3.1 Translating between $\mathbb{H}_{Approximate-Point}$ and $\mathbb{H}_{Interval-Meeting}$

Translation of the lexicon between these two hierarchies is not one-to-one like those between theories in the hierarchies $\mathbb{H}_{Approximate-Point}$ and $\mathbb{H}_{Periods}$, so more complex translation definitions are required.
Definition 4.2 The translation definitions $\Sigma_{\text{ap,im}}$ for the interpretation of theories in $\mathbb{H}_{\text{Interval-Meeting}}$ by theories in $\mathbb{H}_{\text{Approximate-Point}}$ is the set of sentences:

$$(\forall x, y) \text{meets}(x, y) \equiv \text{precedes}(x, y) \land \neg((\exists z) \text{precedes}(x, z) \land \text{precedes}(z, y))$$

Note that starts, finishes, and during are conservative definitions of theories in $\mathbb{H}_{\text{Interval-Meeting}}$ and, therefore, do not require translation definitions to theories in $\mathbb{H}_{\text{Approximate-Point}}$.

Definition 4.3 The translation definitions $\Sigma_{\text{im,ap}}$ for the interpretation of theories in $\mathbb{H}_{\text{Approximate-Point}}$ by theories in $\mathbb{H}_{\text{Interval-Meeting}}$ is the set of sentences:

$$(\forall x, y) \text{precedes}(x, y) \equiv \text{meets}(x, y) \lor ((\exists z) \text{meets}(x, z) \land \text{meets}(z, y))$$

$$(\forall x, y) \text{finer}(x, y) \equiv \text{starts}(x, y) \lor \text{during}(x, y) \lor \text{finishes}(x, y) \lor (x = y)$$

### 4.3.2 Relationship between $T_{\text{interval-meeting}}$ and $T_{\text{ap}}$

Interestingly, these two theories are not definably equivalent:

**Lemma 4.9** $T_{\text{interval-meeting}} \cup \Sigma_{\text{im,ap}} \not\models T_{\text{ap}}$

**Proof:** Mace constructed models of $T_{\text{finite}}^{\text{interval-meeting}} \cup \Sigma_{\text{im,ap}}$ that falsified Axioms (4.1.2.5) and (4.1.2.11) of $T_{\text{ap}}$. This model can be extended to construct a model of $T_{\text{interval-meeting}} \cup \Sigma_{\text{im,ap}}$ that falsifies the axiom. □

The next lemma illustrates the challenges related to the feasibility of our approach that uses an automated theorem prover in the discovery of the relationships between theories.

**Lemma 4.10** $T_{\text{interval-meeting}} \cup \Sigma_{\text{im,ap}} \models T_{\text{ap,root}}$

The theorem prover easily found proofs for the $T_{\text{ap}}$ axioms for infinite intervals in both directions (4.1.2.8 and 4.1.2.9), reflexivity of the finer relation (4.1.2.2), and irreflexivity of the precedes relation (4.1.2.7); however, getting the theorem prover to provide proofs for the seven remaining axioms of $T_{\text{ap}}$ required user assistance in the form of lemmas. For example, in proving the entailment of the transitivity axiom of precedes (4.1.2.6), the user had to manually translate the axiom into the lexicon of
the background theory (in this case $T_{interval\_meeting}$) using the translation definitions ($\Sigma_{im\_op}$) for the theorem prover to find the proof.

The translated axiom:

\[(\forall x, y, z) \ (meets(x, y) \lor ((\exists u)meets(x, u) \land meets(u, y))) \land \]
\[(meets(y, z) \lor ((\exists w)meets(y, w) \land meets(w, z))) \supset \]
\[(meets(x, z) \lor ((\exists p)meets(x, p) \land meets(p, z)))]

The next set of axioms that required the use of lemmas were axioms that contained the relation finer. For the theorem prover to find a proof for the entailment of the anti-symmetry axiom of finer, the need to select only a subset of axioms for use in the background theory, in addition to the use of lemmas were required. The lemmas required were axioms that represented each of the possible cases of the translation of finer to the lexicon of $T_{interval\_meeting}$. For example, when attempting to use the automated theorem prover to prove the axiom stating the transitive property of finer, we must consider all permutations for the different translation possibilities of finer into the language of $T_{interval\_meeting}$ as lemmas. Since the translation of finer to the language of $T_{interval\_meeting}$ is a disjunction of four relations, we will have a lemma corresponding to each of the four relations. Additionally, these general lemmas are required:

\[(\forall i, j) \ starts(i, j) \supset \lnot starts(i, j)\]
\[(\forall i, j) \ during(i, j) \supset \lnot during(j, i)\]
\[(\forall i, j) \ finishes(i, j) \supset \lnot finishes(j, i)\]

To obtain entailment proofs for axioms (4.1.2.1), (4.1.2.12), and (4.1.2.4), similar lemmas that considered each case for the translation of finer individually (nine lemmas for 4.1.2.1, four lemmas for 4.1.2.12, and four lemmas for 4.1.2.4), and the careful selection of axioms used for the background theory were once again required.

Selecting the appropriate subset of axioms of the background theory for the theorem prover to use was significant as this was often the difference between the theorem prover timing out, and a proof being found. In all of the cases we found that definitional axioms and the axiom for infinite intervals (4.1.3.7)
were prime candidates for removal from the background theory when they were not required for the proof (e.g. removing the definitional axiom of \textit{starts} when that relation was not needed in the proof).

**Theorem 4.6** \( T_{\text{ap}} \cup \Sigma_{\text{ap,im}} \nvdash T_{\text{interval,meeting}} \)

**Proof:** Mace constructs a model that falsifies Axiom (4.1.3.11) of \( T_{\text{interval,meeting}} \). \( \square \)

Thus, it seems that the major difference between the theories of \( T_{\text{ap}} \) and \( T_{\text{interval,meeting}} \) is that the first allows for models where gaps can exist between time intervals, whereas the second does not.

We can also define a weaker theory than \( T_{\text{interval,meeting}} \) in \( \mathbb{H}_{\text{Interval-Meeting}} \) such that for the new theory \( T_{\text{meets,root}} \):

**Theorem 4.7** \( T_{\text{ap}} \cup \Sigma_{\text{ap,im}} \models T_{\text{meets,root}} \)

**Proof:** In order to prove Axiom (4.1.3.2) is entailed, the following lemma was needed

\[
(\forall i, j, k, m) \text{meets}(i, j) \land \text{meets}(i, k) \land \text{meets}(l, j) \supset \text{precedes}(l, k)
\]

\( \square \)

By examining the relationship between \( T_{\text{ap}} \) and \( T_{\text{meets,root}} \), we have identified a new ontology \( T_{\text{m,exist}} \) in the hierarchy \( \mathbb{H}_{\text{Approximate-Point}} \) such that \( T_{\text{m,exist}} \) is a nonconservative extension of \( T_{\text{ap}} \) that incorporates the ontological commitment made by \( T_{\text{interval,meeting}} \) of no gaps existing between intervals on the timeline.

The new theory \( T_{\text{m,exist}} \) adds the axiom stating if two intervals do not meet, then there must exist some interval that occupies the gap between them:

\[
(\forall i, j) \neg(\text{precedes}(x, y) \land \neg((\exists z) \text{precedes}(x, z) \land \text{precedes}(z, y)))
\]

\[ \supset \]

\[
((\exists x) \text{precedes}(i, x) \land \neg((\exists z_1) \text{precedes}(i, z_1) \land \text{precedes}(z_1, x)) \land \text{precedes}(x, j) \land \neg((\exists z_2) \text{precedes}(x, z_2) \land \text{precedes}(z_2, j)))
\]

\[ \lor \]

\[
((\exists y) \text{precedes}(j, y) \land \neg((\exists u_1) \text{precedes}(j, u_1) \land \text{precedes}(u_1, j)) \land \text{precedes}(y, i) \land \neg((\exists u_2) \text{precedes}(y, u_2) \land \text{precedes}(u_2, i)))
\]

\( (4.3.2.1) \)
4.4 Semantic Integration between Hierarchies

The technique utilized throughout this chapter centered on using the already available semantic mappings between the two hierarchies to create images of theories from one hierarchy in the other. That is, if a definably equivalent theory is not found in the other hierarchy, then one can be created by extending the weaker theory in the other hierarchy with the axioms found in the difference using the available translation definitions to translate between the languages. This new theory would be definably equivalent to the stronger theory in the current hierarchy, but in the language of the other.

For example, the theory $T_{\text{mixed periods}}$ was found to be stronger than the theory $T_{\text{ap}}$, and so no theory in the hierarchy $H_{\text{Approximate Point}}$ was found to be definably equivalent to $T_{\text{mixed periods}}$. However, we took the axioms of $T_{\text{mixed periods}}$ that $T_{\text{ap}}$ could not entail and translated them into the language of the $H_{\text{Approximate Point}}$ using the set of translation definitions $\Sigma_{\text{ap,p}}$, to extend $T_{\text{ap}}$ with those axioms in order to create the new theory $T_{\text{ap interval}}$ that is definably equivalent to $T_{\text{mixed periods}}$. This new theory is the image of $T_{\text{Mixed Periods}}$ in $H_{\text{Approximate Point}}$. Since we’ve previously proven that $T_{\text{mixed periods}}$ is stronger than $T_{\text{ap}}$ and have shown that $T_{\text{mixed periods}}$ is definably equivalent to $T_{\text{ap interval}}$, then the relationship between $T_{\text{ap}}$ and $T_{\text{ap interval}}$ is one of nonconservative extension with $T_{\text{ap interval}}$ being the stronger theory.

By doing this we are now better able to understand the relationship between original theories of $T_{\text{mixed periods}}$ and $T_{\text{ap}}$. In this case, we have now isolated the set of axioms between these two theories that constitute their differences in ontological commitments. Identifying these sets of axioms that make up the similarities and differences between two theories in opposite hierarchies is an important step for semantic integration applications as it allows the ontology user to explicitly recognize the extent to which two ontologies can accurately share information.

For our work with time ontologies, not only does this technique allow us to better define the relationships between original theories by identifying the set of axioms that comprise their similarities and differences, but by doing so we can begin to determine universally common characteristics of time and also how each of the various extensions are related.
4.5 Summary

In this chapter we managed to establish a rich network of relationships between the various ontologies that represent time as intervals (see Figure 4.9). Doing so has led us to find the different sets of equivalent models shared by the ontologies and the axiomatizations of those models. For instance, we have identified that three theory of dense intervals, one in each of the hierarchies ($T_{INT(Q)}$, $T_{ap\_rational}$, and $T_{ladkin\_intq}$), are equivalent. Through this relationship the differences between the other theories of each hierarchy can be defined. We have provided the set of translation definitions between each of the hierarchies and showed how to use them to extend hierarchies with new theories that are definably equivalent to theories of other hierarchies. Furthermore, we present a few important general techniques used to significantly increase the efficiency and effectiveness of the automated theorem prover in returning proofs. In many cases, these techniques were shown to be the difference between the theorem prover yielding a proof or timing-out. These techniques play a key role in giving us the results needed for the remainder of our work. The relationships identified in this chapter serve as a step forward in understanding all the possible conceptualizations of time by making explicit the similarities and differences between the set of time interval ontologies. Because we were able to to this with all of the hierarchies of time interval ontologies, this shows that these different representations of time can be made definably equivalent. If we can show that all ontologies of time can be made equivalent in this manner, then we can say that no new ontology of time is needed since any of the current ones can be extended to capture the same perspective.
Figure 4.9: Relationships between theories of all Interval Hierarchies in the repository. Dashed-lines denote nonconservative extensions, single-arrowed solid-lines denote definable interpretation, double-arrowed solid-lines denote definable equivalence, clouds define the boundaries of each hierarchy. The relationship between $T_{ladkin\_intq}$ and $T_{INT(Q)}$ is conjectured by Ladkin in [17], but this has not be verified.
Chapter 5

Hybrid-Time Theories

This chapter centres on the verification of hybrid-time ontologies presented in [14] and discovering the relationships that exist between their models. Hybrid-time theories are those that include both time points and time intervals as primitives, and define a set of functions and relations specifying the interactions between them. However, depending on the relations and functions used, they can end up representing time in very different ways. For instance, a theory like $T_{\text{endpoints}}$, that chooses to define points only as the boundary of intervals (i.e. every interval is associated with exactly two points, its beginof and endof) is very different from one like $T_{\text{point continuum}}$, that defines intervals by the set of adjacent points contained within. The third theory examined, $T_{\text{vector continuum}}$ introduces the concept of directionality by allowing backward intervals (where the endof point is before the beginof point in the timeline). With such varied models for each of the hybrid-time theories, it is of interest to investigate their semantic integration possibilities by formalizing the relationships between them.

We begin by introducing each hybrid-time theory and make corrections to their original axioms that were overlooked. Next, we explore the relationships between each theory in a similar manner done for the previous chapter. During the course of investigating these relationships, we encounter an issue regarding the lack of namespaces support in the current CLIF syntax. This proves to be a serious problem when reasoning between theories with overlapping lexicon using a semi-automated procedure that includes an automated reasoner. We look at the situations where namespaces is important for an ontology repository like COLORE and discuss the consequences associated with using the current CLIF syntax.
5.1 Hybrid Theories of Time

Here we introduce each of the three hybrid-time ontologies from [14] that are of interest in this chapter. We present the corrections to their axiomatizations required for their verification via representation theorems.

5.1.1 Theory of Endpoints (\(T_{\text{endpoints}}\))

The endpoints theory combines the language of intervals and points by defining the functions \(\text{beginof}\), \(\text{endof}\), and \(\text{between}\) to relate intervals to points and vice-versa. This theory imports the axioms of \(T_{\text{linear point}}\) that define the binary before relation between timepoints as transitive and irreflexive, and impose the condition that all timepoints are linearly ordered and infinite in both directions. The endpoints theory includes axioms defining the meets-at relation as one between two intervals and the point at which they meet along with conservative definitions for meets, precedes, overlaps, starts, during and finishes. Finally, an axiom that restricts the beginof an interval to always come before its endof and another that states that intervals are between two points if they are properly ordered complete the theory. The first of the final two axioms has the effect of preventing single-point intervals from existing in this theory as an interval that has the same point as its beginof and endof would be inconsistent due to the irreflexivity of the before relation.

The axioms of this theory are:

\[
(\forall i) \text{timeinterval}(i) \supset \text{timepoint}(\text{beginof}(i)) \land \text{timepoint}(\text{endof}(i)) \quad (5.1.1.1)
\]

\[
(\forall x, y) \text{before}(x, y) \equiv (x = \text{beginof}(\text{between}(x, y))) \land (y = \text{endof}(\text{between}(x, y))) \quad (5.1.1.2)
\]

\[
(\forall i, j, x) \text{meetsat}(i, x, j) \supset \text{timeinterval}(i) \land \text{timeinterval}(j) \land \text{timepoint}(x) \quad (5.1.1.3)
\]

\[
(\forall i, j, x) \text{meetsat}(i, x, j) \equiv (x = \text{endof}(i)) \land (x = \text{beginof}(j)) \quad (5.1.1.4)
\]

\[
(\forall i, j) \text{meets}(i, j) \equiv \text{timeinterval}(i) \land \text{timeinterval}(j) \land (\text{endof}(i) = \text{beginof}(j)) \quad (5.1.1.5)
\]

In order to prove the representation theorems, we discovered that the following axioms are missing from the original axioms of endpoints:

\[
(\forall x) \text{timepoint}(x) \lor \text{timeinterval}(x) \quad (5.1.1.6)
\]
These additional axioms serve only to enforce the intended semantics of endpoints as described in [14] by limiting the class of elements to either timepoints or timeintervals with no overlapping elements, and enforcing the correct temporal ordering of the arguments for the between function. These changes were essential to proving consistency of the theory as well as the representation theorems for verification. Thus, we will refer to this extension of the original endpoints theory as \( T_{\text{endpoints}} \).

### 5.1.2 Theory of Vector Continuum (\( T_{\text{vector \_continuum}} \))

The vector \_continuum theory is a one of timepoints and intervals that introduces the notion of orientation of intervals. It also imports theory \( T_{\text{linear \_point}} \) and adds to it axioms that define the meets-at relation and the conservative definitions of meets, precedes, overlaps, starts, during and finishes in the same way as the endpoints theory. Although it has the same three functions (beginof, endof, and between) that transform intervals to points and vice-versa, it differs in its definition of between by allowing the formation of intervals whose endof point is equal to or before its beginof. Thus, unlike the endpoints theory, every interval in vector \_continuum has a reflection in the opposite direction via the back function and intervals oriented in the forward direction (beginof is before endof) are defined by the forward relation. In this theory single-point intervals, known as moments, are defined as intervals whose beginof and endof points are the same.

This axioms of this theory are:

\[
(\forall i) \ \text{timeinterval}(i) \supset \text{timepoint}(\text{beginof}(i)) \land \text{timepoint}(\text{endof}(i)) \quad (5.1.2.1)
\]

\[
(\forall x, y) \ (x = \text{beginof}(\text{between}(x, y))) \land (y = \text{endof}(\text{between}(x, y))) \quad (5.1.2.2)
\]

\[
(\forall i, j, x) \ \text{meetsat}(i, x, j) \supset \text{timeinterval}(i) \land \text{timeinterval}(j) \land \text{timepoint}(x) \quad (5.1.2.3)
\]

\[
(\forall i, j, x) \ \text{meetsat}(i, x, j) \equiv (x = \text{endof}(i)) \land (x = \text{beginof}(j)) \quad (5.1.2.4)
\]

\[
(\forall i, j) \ \text{meets}(i, j) \equiv \text{timeinterval}(i) \land \text{timeinterval}(j) \land (\text{endof}(i) = \text{beginof}(j)) \quad (5.1.2.5)
\]
Similar to endpoints, we discovered that the following axioms are missing from vector_continuum:

\[(\forall x) \text{timepoint}(x) \lor \text{timeinterval}(x)\]  \hspace{1cm} (5.1.2.6)

\[(\forall i, x, y) (\text{beginof}(i) = x) \land \text{endof}(i) = y \supset \text{between}(x, y) = i\]  \hspace{1cm} (5.1.2.7)

Without these axioms, there exist models that falsify the sentence

\[(\forall i) (\text{back} (\text{back}(i)) = i)\]

which Hayes claims to be provable from vector_continuum.

As these axioms are essential in verification, we will refer to this extension of the original vector_continuum theory as \(T_{\text{vector/continuum}}\).

### 5.1.3 Theory of Point Continuum \(T_{\text{point/continuum}}\)

The point − continuum theory combines intervals and points by defining the relation \(\text{in}\) that relates a point to the interval it is contained in. This theory like the others, import the axioms of \(T_{\text{linear/point}}\). All intervals of this theory are oriented in the forward direction and are considered either \(\text{open}\), when the \(\text{beginof}\) and \(\text{endof}\) points are not \(\text{in}\) in the interval, or \(\text{closed}\), when the \(\text{beginof}\) and \(\text{endof}\) points are included \(\text{in}\) in the interval. Therefore, the axioms defining the functions \(\text{beginof}\), \(\text{endof}\), and \(\text{between}\) also make the distinction between open and closed intervals. The axiom for the function \(\text{closure}\) ensures that every \(\text{open}\) interval has a \(\text{closed}\) interval with the same endpoints. The relation \(\text{acoao}\) (also closed or also open) that compares two intervals is essential for the conservative definitions of the temporal relations \(\text{meets}\), \(\text{starts}\) and \(\text{finishes}\). With the distinction between \(\text{closed}\) and \(\text{open}\) intervals, \(\text{open}\) intervals in this theory can only meet and interval that is \(\text{closed}\). Therefore, if two \(\text{open}\) intervals share an endpoint in common (where the \(\text{endof}\) one is equal to the \(\text{beginof}\) the other) these intervals do not \(\text{meet}\) each other, but instead they each meet the same \(\text{closed}\) single-point interval known as a \(\text{moment}\) that resides between them.

\[(\forall x, y) \text{b/\text{before}}(x, y) \equiv \text{before}(x, y) \lor (x = y)\]  \hspace{1cm} (5.1.3.1)
(∀x, y) in(x, y) ⊃ timepoint(x) ∧ timeinterval(y) \hspace{1cm} (5.1.3.2)

(∀i) timeinterval(i) ⊃ (open(i) ∧ ¬closed(i) ∧

(∀p)(in(p, i) ⇔ before(beginof(i), p) ∧ before(p, endof(i)))) ∨

(closed(i) ∧ ¬open(i) ∧ (∀q)(in(q, i) ⇔ (bbefore(beginof(i), q) ∧ bbefore(q, endof(i)))) \hspace{1cm} (5.1.3.3)

(∀p, q) timepoint(p) ∧ timepoint(q) ⊃ before(p, q) ∨ (∃i)(timeinterval(i) ∧ closed(i) ∧ (beginof(i) = p) ∧ (endof(i) = q)) \hspace{1cm} (5.1.3.4)

(∀p, q) timepoint(p) ∧ timepoint(q) ⊃ bbefore(p, q) ⇔ timeinterval(between(p, q)) ∧

(p = beginof(between(p, q))) ∧ (q = endof(between(p, q)))) \hspace{1cm} (5.1.3.5)

(∀i) timeinterval(i) ⊃ timeinterval(closure(i)) ∧ closed(closure(i)) ∧

(beginof(i) = beginof(closure(i))) ∧ (endof(i) = endof(closure(i))) \hspace{1cm} (5.1.3.6)

(∀i, j) acoao(i, j) ⇔

timeinterval(i) ∧ timeinterval(j) ⊃ (open(i) ∧ open(j)) ∨ (closed(i) ∧ closed(j))) \hspace{1cm} (5.1.3.7)

(∀i, j) meets(i, j) ⇔

timeinterval(i) ∧ timeinterval(j) ⊃ (¬acoao(i, j) ∧ (endof(i) = beginof(j)))) \hspace{1cm} (5.1.3.8)

The following axioms were found to be missing from point continuum:

(∀x) timepoint(x) ∨ timeinterval(x) \hspace{1cm} (5.1.3.10)

(∀i) open(i) ⊃ timeinterval(i) \hspace{1cm} (5.1.3.11)

(∀i) closed(i) ⊃ timeinterval(i) \hspace{1cm} (5.1.3.12)

(∀i) timeinterval(i) ⊃

timepoint(beginof(i)) ∧ timepoint(endof(i)) \hspace{1cm} (5.1.3.13)
Once again the corrections here focus on defining the scope of elements in the theory and the scope of elements for some relations missing in the original axioms. That is, the scope of the open and closed relation are limited to time intervals and the relationship between timepoints and time intervals via the beginof and endof functions are correctly specified. We will now refer to the extension of the original point continuum theory with these axioms as $T_{\text{point continuum}}$.

### 5.2 Relationships between Hybrid-Time Theories

Similar to the previous chapter, we are looking to identify the relationship between the different hybrid-time ontologies. These connections give us a better understanding of the theories as a whole as they allow us to compare the similarities and differences of their models (see Figure 5.1). Detail regarding the relationships identified in Figure 5.1 is explained throughout the section as they become relevant.

Figure 5.1: Relationships between hybrid-time theories. Dashed-lines denote nonconservative extensions, solid-lines denote definable interpretation, clouds define the boundaries of each hierarchy, bolded theories are new additions to the hierarchy.
5.2.1 Relationship between \( T_{\text{endpoints}} \) and \( T_{\text{vector\_continuum}} \)

Since the theories of \( T_{\text{endpoints}} \) and \( T_{\text{vector\_continuum}} \) have an equivalent set of primitive nonlogical lexicon that are also semantically equivalent, we can determine their relationship using the notions of satisfiability, extension, and independence. That is, these two theories actually belong to the same hierarchy. In particular, we use the following notion from [12]:

**Definition 5.1** Let \( T_1 \) and \( T_2 \) be theories with the language \( \mathcal{L} \). The similarity of \( T_1 \) and \( T_2 \) is the strongest subtheory of \( T_1 \) and \( T_2 \) so that for all \( \sigma, \phi \in \mathcal{L} \):

if \( T_1 \models \sigma \) and \( T_2 \models \phi \) and \( T \not\models \sigma \) and \( T \not\models \phi \), then either \( \sigma \lor \phi \) is independent of \( T \) or it is a tautology.

Let \( \text{Sim}(\text{endpoint, vector\_continuum}) \) be the theory which is equivalent to \( T_{\text{vector\_continuum}} \) with the axiom:

\[
(\forall p, q) \text{timepoint}(p) \land \text{timepoint}(q) \supset

(\text{beginof}(\text{between}(p, q)) = p)

\land (\text{endof}(\text{between}(p, q)) = q)
\]

replaced by:

\[
(\forall p, q) \text{before}(p, q) \supset

(\text{beginof}(\text{between}(p, q)) = p)

\land (\text{endof}(\text{between}(p, q)) = q)
\]

**Theorem 5.1** \( \text{Sim}(\text{endpoint, vector\_continuum}) \) is the similarity of \( T_{\text{endpoints}} \) and \( T_{\text{vector\_continuum}} \).

**Proof:** Let \( \text{backwards} \) be the sentence

\[
(\forall i_1) \text{timeinterval}(i_1) \supset (\exists i_2) \text{timeinterval}(i_2)
\land (\text{beginof}(i_2) = \text{endof}(i_1)) \land (\text{endof}(i_2) = \text{beginof}(i_1))
\]

Let \( \text{no\_backwards} \) be the sentence

\[
(\forall i_2) \text{timeinterval}(i_1) \supset \neg (\exists i_2) \text{timeinterval}(i_2)
\]
\( \land (beginof(i_2) = endof(i_1)) \land (endof(i_2) = beginof(i_1)) \)

Let \textit{moment} be the sentence

\[
(\forall t) \text{timepoint}(t) \supset (\exists i) \text{timeinterval}(i)
\]

\( \land (beginof(i) = t) \land (endof(i) = t) \)

Let \textit{no\_moment} be the sentence

\[
(\forall t) \text{timepoint}(t) \supset \neg(\exists i) \text{timeinterval}(i)
\]

\( \land (beginof(i) = t) \land (endof(i) = t) \)

We show using Prover9:

\[
T_{\text{endpoints}} \models \text{no\_backwards} \land \text{no\_moment}
\]

\[
T_{\text{vector\_continuum}} \models \text{backward} \land \text{moment}
\]

and using Mace4, we show that if disjunctions of these sentences are not tautologies, then they are independent of \(Sim(\text{endpoints, vector\_continuum})\). \(\square\)

The next corollary is not explicitly stated in [14], but it follows from the propositions used in the proof of Theorem 5.1.

\textbf{Corollary 1} \(T_{\text{endpoints}}\) and \(T_{\text{vector\_continuum}}\) are mutually inconsistent.

This follows from the differences in ontological commitments made by each theory, as \(T_{\text{endpoints}}\) is intended to have only forward-facing intervals and also not to have single-point intervals (i.e. \textit{moments}), and \(T_{\text{vector\_continuum}}\) commits to including \textit{moments} and backward-facing intervals as shown in Figure 5.1. Thus, by defining the theory \(Sim(\text{endpoints, vector\_continuum})\) that constitutes the similarity between \(T_{\text{endpoints}}\) and \(T_{\text{vector\_continuum}}\) we are able to isolate their exact differences in ontological commitments that render these two original theories mutually inconsistent. Furthermore, each of these differences now specifies a separate extension of the time domain.
5.2.2 Relationship of $T_{\text{point} \cdot \text{continuum}}$ to $T_{\text{endpoints}}$ and $T_{\text{vector} \cdot \text{continuum}}$

We return to the notion of relative interpretation to identify the relationships between $T_{\text{point} \cdot \text{continuum}}$ to $T_{\text{endpoints}}$ and $T_{\text{vector} \cdot \text{continuum}}$ as they no longer share the same language\(^1\).

**Definition 5.2** Let $\Sigma_{\text{pe} \cdot \text{ec}}$ be the following set of translation definitions:

$$
(\forall x) \text{timepoint}^\text{endpoints}(x) \equiv \text{timepoint}^\text{point} \cdot \text{continuum}(x) \\
(\forall x) \text{timeinterval}^\text{endpoints}(x) \equiv \text{open}^\text{point} \cdot \text{continuum}(x) \\
(\forall x, y) (\text{beginof}^\text{endpoints}(x) = y) \equiv (\text{beginof}^\text{point} \cdot \text{continuum}(x) = y) \\
(\forall x, y) (\text{endof}^\text{endpoints}(x) = y) \equiv (\text{endof}^\text{point} \cdot \text{continuum}(x) = y)
$$

**Theorem 5.2** $T_{\text{point} \cdot \text{continuum}}$ definably interprets $T_{\text{endpoints}}$.

**Proof:** Using Prover9, we have shown that

$$
T_{\text{point} \cdot \text{continuum}} \cup \Sigma_{\text{pe} \cdot \text{ec}} \models T_{\text{endpoints}}
$$

\(\Box\)

**Definition 5.3** Let $\Sigma_{\text{pe} \cdot \text{vc}}$ be the following set of translation definitions:

$$
(\forall x) \text{timepoint}^\text{vector} \cdot \text{continuum}(x) \equiv \text{timepoint}^\text{point} \cdot \text{continuum}(x) \\
(\forall x) \text{timeinterval}^\text{vector} \cdot \text{continuum}(x) \equiv \text{closed}^\text{point} \cdot \text{continuum}(x) \\
(\forall x, y) (\text{beginof}^\text{vector} \cdot \text{continuum}(x) = y) \equiv (\text{beginof}^\text{point} \cdot \text{continuum}(x) = y) \\
(\forall x, y) (\text{endof}^\text{vector} \cdot \text{continuum}(x) = y) \equiv (\text{endof}^\text{point} \cdot \text{continuum}(x) = y)
$$

**Theorem 5.3** $T_{\text{point} \cdot \text{continuum}}$ definably interprets $T_{\text{vector} \cdot \text{continuum}}$.

**Proof:** Using Prover9, we have shown that

$$
T_{\text{point} \cdot \text{continuum}} \cup \Sigma_{\text{pe} \cdot \text{vc}} \models T_{\text{vector} \cdot \text{continuum}}
$$

\(\Box\)

---

\(^1\)Since these theories use relations with the same names, we distinguish them by a superscript that denotes the theory in which they are axiomatized.
Notice that intervals in \( T_{endpoints} \) are interpreted as open intervals in \( T_{point\ continuum} \) and intervals in \( T_{vector\ continuum} \) are interpreted as closed intervals. In this sense, the inconsistency between \( T_{endpoints} \) and \( T_{vector\ continuum} \) appears as the disjointness of the classes of open and closed intervals in \( T_{point\ continuum} \). It is important to note that \( T_{point\ continuum} \) is not equivalent to the combination of axioms of \( T_{endpoints} \) and \( T_{vector\ continuum} \), but it is the new language of \( T_{point\ continuum} \) that includes the definition of subclasses of intervals that allows it to interpret mutually inconsistent theories while remaining consistent. Therefore, in Figure 5.1 \( T_{point\ continuum} \) is shown to be residing in a different hierarchy than \( T_{endpoints} \) and \( T_{vector\ continuum} \).

5.3 Representation Theorems for Hybrid-Time Theories

Once again, we are verifying each of the hybrid-time ontologies by proving that the models of the theory’s axiomatization are exactly the set of intended models. To do so, we use representation theorems to characterize the models of each theory up to isomorphism and show that they are elementary equivalent to the set of well-understood mathematical structures. In the case of verifying the hybrid-time theories, we look to the classes of mathematical structures called graphical incidence structures. We introduce the various classes of graphical incidence structures needed for each representation theorem before verifying each of the hybrid-time theories individually.

5.3.1 Graphical Incidence Structures

The basic building blocks for the models of theories in this chapter are based on the notion of incidence structures ([2]).

**Definition 5.4** A k-partite incidence structure is a tuple \( \Pi = (\Omega_1, \ldots, \Omega_k, \text{in}) \), where \( \Omega_1, \ldots, \Omega_k \) are sets with
\[
\Omega_i \cap \Omega_j = \emptyset, i \neq j
\]
and
\[
\text{in} \subseteq \left( \bigcup_{i \neq j} \Omega_i \times \Omega_j \right)
\]
Two elements of \( \Pi \) that are related by \( \text{in} \) are called incident.
We will use special classes of incidence structures to construct the models of each hybrid-time
theory.

**Definition 5.5** An strict graphical incidence structure is a bipartite incidence structure
\[ G = \langle X, Y, \text{in}^G \rangle \]
such that all elements of \( Y \) are incident with exactly two elements of \( X \), and for each pair of points \( p, q \in X \) there exists a unique element in \( Y \) that is incident with both \( p \) and \( q \).

The class of strict graphical incidence structures is axiomatized by \( T_{\text{strict\_graphical}} \).

**Definition 5.6** An strong graphical incidence structure is a bipartite incidence structure
\[ S = \langle X, Y, \text{in}^S \rangle \]
such that all elements of \( Y \) are incident with either one or two elements of \( X \), and for each pair of points \( p, q \in X \) there exists a unique element in \( Y \) that is incident with both \( p \) and \( q \).

The class of strong graphical incidence structures is axiomatized by \( T_{\text{strong\_graphical}} \).

These two classes of incidence structures get their names from graph-theoretic representation theo-
remes of their own.

**Definition 5.7** A graph \( G = (V, E) \) consists of a nonempty set \( V \) of vertices and a set \( E \) of ordered
pairs of vertices called edges.

An edge whose vertices coincide is called a loop. A graph with no loops or multiple edges is a
simple graph.

A complete graph is a graph in which each pair of vertices is adjacent.

**Theorem 5.4** Let \( G = (V, E) \) be a complete simple graph.

A bipartite incidence structure is a strict graphical incidence structure iff it is isomorphic to \( I = (V, E, \in) \), where \( \in \) is the containment relation for vertices in an edge.

**Theorem 5.5** Let \( G = (V, E) \) be a complete graph with loops.

A bipartite incidence structure is a strong graphical incidence structure iff it is isomorphic to \( I = (V, E, \in) \).
These two representation theorems show that there is a one-to-one correspondence between each class of incidence structures and the given class of graphs. This gives us a characterization of the incidence structures up to isomorphism.

A third class of incidence structures requires the notion of the direct product of incidence structures:

**Definition 5.8** Given two incidence structures \( \mathcal{I}_1 = \langle \mathcal{P}_1, \mathcal{L}_1, \text{in}_1 \rangle \) and \( \mathcal{I}_2 = \langle \mathcal{P}_2, \mathcal{L}_2, \text{in}_2 \rangle \) the direct product \( \mathcal{I}_1 \times \mathcal{I}_2 \) is the incidence structure such that

- \( \mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2 \);
- \( \mathcal{L} = (\mathcal{P}_1 \times \mathcal{L}_2) \cup (\mathcal{L}_1 \times \mathcal{P}_2) \);
- the point \((x, y)\) is incident with the line \((x, L) \in \mathcal{P}_1 \times \mathcal{L}_2 \) iff \((y, L) \in \text{in}_2\);
- the point \((x, y)\) is incident with the line \((M, y) \in \mathcal{L}_1 \times \mathcal{P}_2 \) iff \((x, M) \in \text{in}_1\).
Definition 5.9 Let $G = \langle P, G, \text{in}^G \rangle$ be a strict graphical incidence structure. Let $S = \langle P, S, \text{in}^S \rangle$ be a strong graphical incidence structure.

An incidence structure $D = \langle P, X, \text{in}^D \rangle$ is a double complete incidence structure iff

$$D \cong G \times S$$

The class of double complete incidence structures is axiomatized by $T_{\text{double complete}}$.

As above, we can also provide a representation theorem for double complete incidence structures with respect to a class of graphs:

Theorem 5.6 Let $G = (V, E)$ be a complete graph in which $E$ is a symmetric reflexive relation.

A bipartite incidence structure is a double complete incidence structure iff it is isomorphic to $I = (V, E, \in)$.

5.3.2 Representation Theorem for $T_{\text{endpoints}}$

We want to show that every model of $T_{\text{endpoints}}$ can be represented by the classes of mathematical structures of infinite linear ordering and strict graphical incidence structures.

The first step utilizes the sets of translation definitions $\Sigma_{\text{ep,is}}$ and $\Sigma_{\text{is,ep}}$ in the reasoning tasks that prove mutual interpretability between the theories of $T_{\text{endpoints}}$ and $T_{\text{infinite linear ordering}} \cup T_{\text{strict graphical}}$.

Definition 5.10 The translation definitions $\Sigma_{\text{ep,is}}$ for the interpretation of $T_{\text{infinite linear ordering}} \cup T_{\text{strict graphical}}$ in $T_{\text{endpoints}}$ is the set of sentences:

$$(\forall x) \; \text{point}(x) \equiv \text{timepoint}(x)$$

$$(\forall x) \; \text{line}(x) \equiv \text{timeinterval}(x)$$

$$(\forall x, y) \; \text{in}(x, y) \equiv ((\text{beginof}(y) = x) \lor (\text{endof}(y) = x))$$

$$(\forall x, y) \; \text{before}(x, y) \equiv \text{leq}(x, y)$$

Definition 5.11 The translation definitions $\Sigma_{\text{is,ep}}$ for the interpretation of $T_{\text{endpoints}}$ in $T_{\text{infinite linear ordering}} \cup T_{\text{strict graphical}}$ is the set of sentences:

$$(\forall x) \; \text{timepoint}(x) \equiv \text{point}(x)$$
\((\forall x)\) \text{timeinterval}(x) \equiv \text{line}(x)\\
(\forall x, y) (\text{beginof}(y) = x) \equiv ((\text{in}(x, y) \land ((\forall z) \text{in}(z, y) \supset \text{leq}(x, z)))\\
(\forall x, y) (\text{endof}(y) = x) \equiv ((\text{in}(x, y) \land ((\forall z) \text{in}(z, y) \supset \text{leq}(z, x)))\\
(\forall x, y) \text{before}(x, y) \equiv \text{leq}(x, y)

Instantiating (Rep \(-\) 1) and (Rep \(-\) 2) shows:

**Theorem 5.7** \(T_{\text{endpoints}}\) is definably equivalent to

\[ T_{\text{infinite linear ordering}} \cup T_{\text{strict graphical}} \]

**Proof:** Using Prover9, we have shown that

\[ T_{\text{endpoints}} \cup \Sigma_{\text{ep}} \models T_{\text{infinite linear ordering}} \cup T_{\text{strict graphical}} \]

and

\[ T_{\text{infinite linear ordering}} \cup T_{\text{strict graphical}} \cup \Sigma_{\text{is ep}} \models T_{\text{endpoints}} \]

□

The second step in the verification of \(T_{\text{endpoints}}\) requires the definition of the class of intended models:

**Definition 5.12** \(M_{\text{endpoints}}\) is the following class of structures: \(M \in M_{\text{endpoints}}\) iff

1. \(M \cong P \cup G\), where
   
   (a) \(P = (P, \leq)\) is a linear ordering
   
   (b) \(G = (P, G, \text{in}_G)\) is a strict graphical incidence structure.

2. \((t) \in \text{timepoint} \iff t \in P\);

3. \((i) \in \text{timeinterval} \iff i \in G\);

4. \(\text{beginof}(i) = t \iff (t, i) \in \text{in}_G\) and for any \(t' \in P\) such that \((t', i) \in \text{in}_G\), we have \(t \leq t'\).
5. \( \text{endof}(i) = t \) iff \( \langle t, i \rangle \in \text{in}^G \) and for any \( t' \in P \) such that \( \langle t', i \rangle \in \text{in}^G \), we have \( t' \leq t \).

6. \( \text{between}(t_1, t_2) = i \) iff \( \langle t_1, i \rangle, \langle t_2, i \rangle \in \text{in}^G \);

7. \( \langle t_1, t_2 \rangle \in \text{before} \) iff \( t_1 < t_2 \).

The Representation Theorem for \( T_{\text{endpoints}} \) shows that the class of intended structures does characterize the models of \( T_{\text{endpoints}} \) up to isomorphism:

**Theorem 5.8** \( \mathcal{M} \in \mathfrak{M}_{\text{endpoints}} \) iff \( \mathcal{M} \in \text{Mod}(T_{\text{endpoints}}) \).

**Proof:** This follows from Theorem 2.1 and Theorem 5.7, together with the fact that \( T_{\text{strict_graphical}} \) axiomatizes the class of strict graphical incidence structures and \( T_{\text{infinite_linear_ordering}} \) axiomatizes the class of infinite linear orderings. \( \square \)

### 5.3.3 Representation Theorem for \( T_{\text{vector_continuum}} \)

We want to verify \( T_{\text{vector_continuum}} \) using the classes of mathematical structures of infinite linear ordering and double complete incidence structures.

Verifying this theory utilizes the sets of translation definitions \( \Sigma - \text{vc}_{i}d \) and \( \Sigma_{i}d_{\text{vc}} \) in the reasoning tasks that prove mutual interpretability between the theories of \( T_{\text{vector_continuum}} \) and \( T_{\text{infinite_linear_ordering}} \cup T_{\text{double_complete}} \).

**Definition 5.13** The translation definitions \( \Sigma - \text{vc}_{i}d \) for the interpretation of \( T_{\text{infinite_linear_ordering}} \cup T_{\text{double_complete}} \) in \( T_{\text{vector_continuum}} \) is the set of sentences:

\[
(\forall x) \text{point}(x) \equiv \text{timepoint}(x)
\]

\[
(\forall x) \text{line}(x) \equiv \text{timeinterval}(x)
\]

\[
(\forall x, y) \text{in}^g(x, y) \equiv ((\text{beginof}(y) = x) \lor (\text{endof}(y) = x))
\]

\[
(\forall x, y) \text{in}^s(x, y) \equiv ((\text{beginof}(y) = x) \lor (\text{endof}(y) = x))
\]

\[
(\forall x, y) \text{before}(x, y) \equiv \text{leq}(x, y)
\]
Definition 5.14  The translation definitions $\Sigma_{id,vc}$ for the interpretation of $T_{vector\_continuum}$ in $T_{infinite\_linear\_ordering} \cup T_{double\_complete}$ is the set of sentences:

\[(\forall i, t) (\text{begin}(i) = t) \equiv (\forall t') (\text{in}^g(t', i) \supset \text{leq}(t, t')) \]

\[(\forall i, t) (\text{end}(i) = t) \equiv (\forall t') (\text{in}^s(t', i) \supset \text{leq}(t', t))\]

Instantiating (Rep – 1) and (Rep – 2) shows:

Theorem 5.9  $T_{vector\_continuum}$ is definably equivalent to $T_{infinite\_linear\_ordering} \cup T_{double\_complete}$

Proof: Using Prover9, we have shown that

$T_{vector\_continuum} \cup \Sigma_{vc} |\models T_{infinite\_linear\_ordering} \cup T_{double\_complete}$

and

$T_{infinite\_linear\_ordering} \cup T_{double\_complete} \cup \Sigma_{id,vc} \models T_{vector\_continuum}$

□

The definition of the class of intended structures in this case is slightly more complicated as two different incidence substructures are required – a strict graphical incidence structure for forward intervals and a strong graphical incidence structure for backward intervals:

Definition 5.15  $\mathcal{M}_{vector\_continuum}$ is the following class of structures: $\mathcal{M} \in \mathcal{M}_{vector\_continuum}$ iff

1. $\mathcal{M} \cong \mathcal{P} \cup (\mathcal{G} \times \mathcal{S})$, where

   (a) $\mathcal{P} = \langle P, \leq \rangle$ is a linear ordering;
(b) \( G = \langle P, G, \text{in}^G \rangle \) is a strict graphical incidence structure;

(c) \( S = \langle P, S, \text{in}^S \rangle \) is a strong graphical incidence structure.

2. \( \langle t \rangle \in \text{timepoint} \) iff \( t \in P \);

3. \( \langle i \rangle \in \text{timeinterval} \) iff \( i \in G \cup S \);

4. \( \langle i \rangle \in \text{moment} \) iff \( i \in S \) and there exists a unique \( t \in P \) such that \( \langle t, i \rangle \in \text{in}^S \);

5. \( \text{beginof}(i) = t \) iff
   - \( \langle t, i \rangle \in \text{in}^G \) and for any \( t' \in P \) such that \( \langle t', i \rangle \in \text{in}^G \), we have \( t \leq t' \), or
   - \( \langle t, i \rangle \in \text{in}^S \) and for any \( t' \in P \) such that \( \langle t', i \rangle \in \text{in}^S \), we have \( t' \leq t \).

6. \( \text{endof}(i) = t \) iff
   - \( \langle t, i \rangle \in \text{in}^G \) and for any \( t' \in P \) such that \( \langle t', i \rangle \in \text{in}^G \), we have \( t' \leq t \), or
   - \( \langle t, i \rangle \in \text{in}^S \) and for any \( t' \in P \) such that \( \langle t', i \rangle \in \text{in}^S \), we have \( t \leq t' \).

The Representation Theorem for \( T_{\text{vector,continuum}} \) shows that this definition of intended structures does characterize the models of \( T_{\text{vector,continuum}} \) up to isomorphism:

**Theorem 5.10** \( \mathcal{M} \in \mathfrak{M}_{\text{vector,continuum}} \) iff \( \mathcal{M} \in \text{Mod}(T_{\text{vector,continuum}}) \).

**Proof:** This follows from Theorem 5.9 and Theorem 2.1, together with the fact that \( T_{\text{double,complete}} \) axiomatizes the class of double complete incidence structures and \( T_{\text{infinite,linear,ordering}} \) axiomatizes the class of infinite linear orderings. \( \square \)

### 5.3.4 Representation Theorem for \( T_{\text{point,continuum}} \)

As the relationships between the three theories indicate, the verification of \( T_{\text{point,continuum}} \) combines the representation theorems for both \( T_{\text{endpoints}} \) and \( T_{\text{vector,continuum}} \).
**Theorem 5.11** \( T_{\text{point,continuum}} \) is definably equivalent to

\[
T_{\text{infinite,linear,ordering}} \cup T_{\text{strict,graphical}} \cup T_{\text{double,complete}}
\]

**Proof:** We can combine the set of translation definitions from Theorems 5.2 and 5.3 and the representation theorems for \( T_{\text{endpoints}} \) (Theorem 5.8) and for \( T_{\text{vector,continuum}} \) (Theorem 5.10) to get \( \Sigma_{\text{pc,isd}} \) and show that

\[
T_{\text{point,continuum}} \cup \Sigma_{\text{pc,isd}} \models T_{\text{infinite,linear,ordering}} \cup T_{\text{strict,graphical}} \cup T_{\text{double,complete}}
\]

For the other direction, we can use the translation definitions from Theorems 5.7 and 5.9. □

The characterization of the models of \( T_{\text{point,continuum}} \) combines the classes of models that were introduced in Definitions 5.12 and 5.15:

**Definition 5.16** \( \mathcal{M}_{\text{point,continuum}} \) is the following class of structures: \( \mathcal{M} \in \mathcal{M}_{\text{point,continuum}} \) iff

1. \( \mathcal{M} \cong \mathcal{P} \cup \mathcal{O} \cup (\mathcal{G} \times \mathcal{S}), \) where
   
   (a) \( \mathcal{P} = \langle P, \leq \rangle \) is a linear ordering,
   
   (b) \( \mathcal{O} = \langle P, O, \text{in}^{O} \rangle \) is a strict graphical incidence structure,
   
   (c) \( \mathcal{G} = \langle P, G, \text{in}^{G} \rangle \) is a strict graphical incidence structure,
   
   (d) \( \mathcal{S} = \langle P, S, \text{in}^{S} \rangle \) is a strong graphical incidence structure;

2. \( \langle t \rangle \in \text{timepoint} \) iff \( t \in P; \)

3. \( \langle i \rangle \in \text{timeinterval} \) iff \( i \in O \cup G \cup S; \)

4. \( \langle i \rangle \in \text{open} \) iff \( i \in O; \)

5. \( \langle i \rangle \in \text{closed} \) iff \( i \in G \cup S; \)

**Theorem 5.12** \( \mathcal{M} \in \mathcal{M}_{\text{point,continuum}} \) iff \( \mathcal{M} \in \text{Mod}(T_{\text{point,continuum}}). \)
**Proof:** This follows from Theorem 2.1 and Theorem 5.11, together with the fact that $T_{\text{strict,graphical}}$ axiomatizes the class of strict graphical incidence structures, $T_{\text{double,complete}}$ axiomatizes the class of double complete incidence structures and $T_{\text{infinite,linear,ordering}}$ axiomatizes the class of infinite linear orderings. □

### 5.4 Namespacing Issue in Common Logic

There are two scenarios where the lack of namespacing support by the CLIF syntax becomes apparent in an ontology repository environment like COLORE. The first occurs as a result of storing a large number of ontologies spanning a wide breadth of domains, from mathematics to manufacturing processes. Since words of the natural language tend to have multiple meanings, the likelihood that ontologies from unrelated domains have overlapping lexicon is high. Consider two disjoint ontologies, the ontology of time, $T_{\text{point,continuum}}$, that contains the $in$ relation defining the semantics of time points contained in a time interval, and the theory of $T_{\text{tripartite,incidence,structures}}$ that contains its own $in$ relation that specifies the relationship between points, lines, and edges. Although, CLIF was designed to support integration of ontologies by including features like `cl-imports` and `cl-module`, a user would not be able to properly integrate these two ontologies strictly using CLIF. Without a namespacing mechanism to differentiate between the two $in$ relations, CLIF would force the domains of each ontology to merge unintentionally. This leads to possible inconsistencies and unintended models of the integrated theory. To illustrate this, let us consider the mode basic axiom of each $in$ relation that specifies the class-restriction of its arguments:

\[
(\forall i, j) \in(x, i) \supset timepoint(x) \land timeinterval(i)
\]

\[
(\forall e, s) \in(s, t) \supset (point(s) \land line(t)) \lor (line(s) \land plane(t))
\]

Without proper namespacing, every instantiation of $in$ in the integrated ontology would unintentionally result in elements in the model that were both timepoints and points, timepoints and lines, timeintervals and lines, or timeintervals and planes. As you can entail from the two axioms above the sentence:
(forall \(x, y\) in (x, y) \supset timepoint(x) \land (point(x) \lor line(x)) \land timeinterval(y) \land (line(y) \lor plane(y)))

This creates unintended models with far greater consequences when you factor in the rest of the axioms that define \(in\) as CLIF would treat them as the same relation. Therefore, if the CLIF syntax aims to properly support ontology integration and modularization, it needs to incorporate namespacing functionality with its `cl-module` and `cl-imports` features.

The second scenario when namespacing is necessary occurs when mapping between different ontologies that axiomatize the same domain. This scenario occurs naturally in COLORE as defining the similarities and differences of ontologies within the same domain is part of the repository’s core-functionality. An example of this situation can be taken from the results of this chapter in determining the relationships between theories of \(T_{\text{endpoints}}\), \(T_{\text{vector continuum}}\), and \(T_{\text{point continuum}}\). To prove that the models of \(T_{\text{point continuum}}\) are composed from the disjoint models of \(T_{\text{vector continuum}}\) and \(T_{\text{endpoints}}\), namespacing is required as part of the translation definitions between the theories due to their overlapping lexicon (i.e. the class of `timeintervals` from \(T_{\text{vector continuum}}\) must be kept separate from the class of `timeintervals` from \(T_{\text{endpoints}}\), and mapped correctly to the class of `timeintervals` in \(T_{\text{point continuum}}\)). While we have side-stepped this issue by manually re-naming the predicates of each theory to keep them disjoint, we feel that ontology-users should not have to alter verified ontologies for this purpose. Therefore, for CLIF to support ontology engineering functions like integration and verification in an ontology repository environment, it is critical to support namespacing within the CLIF constructs of `cl-module` and `cl-imports`.

### 5.5 Summary

In this chapter we’ve looked at the set of hybrid-time ontologies and determined both the relationships between each of the three theories and verified them using representation theorems. We began by making crucial modifications to the axioms of each theory to remove inconsistencies between the original axioms and unintended models. Then, we managed to prove that the two theories that model time using the presepective of intervals as first-class objects, \(T_{\text{endpoints}}\) and \(T_{\text{vector continuum}}\), are mutually inconsistent. However, we were able to find the axiomatization for the models they had in common
(i.e. their similarity). By showing, that the theory $T_{\text{point, continuum}}$ interpreted both $T_{\text{endpoints}}$ and $T_{\text{vector, continuum}}$ and that its models are actually the disjoint union of those two inconsistent theories, we were able to reuse our work in verifying $T_{\text{endpoints}}$ and $T_{\text{vector, continuum}}$ to find the appropriate representation theorem to verify $T_{\text{point, continuum}}$.

In examining these theories of hybrid-time, we came across the issue of namespacing. These theories shared an overlapping set of lexicon, however, the lexicon was not semantically equivalent between theories (e.g. timepoint objects in $T_{\text{point, continuum}}$ are not guaranteed to be timepoint objects in $T_{\text{endpoints}}$). Without namespacing support as part of the CL syntax, any task requiring the integration of these theories would result in unintended interactions between their structures. Thus, the case was made for including namespacing support for CLIF if it is to support ontology repositories, modularity, and ontology reuse.

The work in this chapter filled out the remaining set of hierarchies of time ontologies of interest. Thus, we move to the last set of relationships missing, those between the hierarchies of ontologies that differ in their perspective of time (i.e. points, intervals, hybrid-time).
Chapter 6

Relationships Between Ontologies Across Perspectives of Time

Having formally defined the relationships between ontologies of time axiomatized from the same perspective (i.e. points, intervals, hybrid-time with both points and intervals), we turn our attention to the relationships between ontologies across these different perspectives. Are there point-only theories and interval-only theories that are equivalent with each other? How do we map between their models? Do the hybrid-time theories interpret points and intervals in the same way as the point-only and interval-only theories?

Although we rely on the same notions of interpretation to define the relationship between the models of each theory, we are able to reduce the amount of work required to prove these new relationships by leveraging the results from previous chapters. Finally, we present an extension to the COLORE architecture to support the semantic integration methodology and results from this thesis.

6.1 Between Interval Theories and Point Theories

In this section we look at the relationship between the time ontologies that represent time purely as points ($\mathbb{H}_{Point}$) and those that represent time purely as intervals ($\mathbb{H}_{Approximate-Point}$ and $\mathbb{H}_{Interval-Meeting}$). We investigate the issue of mapping between models of time as points to time as intervals and define a set of translation definitions required to map between their lexicon.
6.1.1 Relationship between $T_{\text{linear\_point}}$ and $T_{\text{interval\_meeting}}$

The theory of $\text{linear\_point}$ represents time as an infinite linearly ordered set of points, while the theory of $\text{interval\_meeting}$ represents time as an infinite linearly ordered set of intervals. In order to translate between the lexicon of $T_{\text{interval\_meeting}}$ to $T_{\text{linear\_point}}$ we define a relation, $\text{interval\_map}$, as part of the translation definitions that maps a pair of $\text{timepoints}_{\text{linear\_point}}$ to the corresponding $\text{timeinterval}_{\text{interval\_meeting}}$.

![interval_map relation translating between timeintervals and timepoints.]

**Figure 6.1:** $\text{interval\_map}$ relation translating between timeintervals and timepoints.

**Definition 6.1** The translation definitions $\Sigma_{lp\_im}$ for the interpretation of theories in $\mathbb{H}_{\text{Interval-Meeting}}$ by theories in $\mathbb{H}_{\text{Point}}$ is the set of sentences:

\[
(\forall i, j, x, y) \text{interval\_map}(i, x, y) \land \text{interval\_map}(j, x, y) \supset (i = j)
\]

\[
(\forall i, x, y, z, w) \text{interval\_map}(i, x, y) \land \text{interval\_map}(i, z, w) \supset (x = z) \land (y = w)
\]

\[
(\forall i, x, y) \text{interval\_map}(i, x, y) \supset \text{timeinterval}(i) \land \text{timepoint}(x) \land \text{timepoint}(y)
\]

\[
(\forall x, y) \text{before}(x, y) \equiv (\exists i) \text{interval\_map}(i, x, y)
\]

\[
(\forall i) \text{timeinterval}(i) \equiv (\exists x, y) \text{interval\_map}(i, x, y) \land \text{before}(x, y)
\]

\[
(\forall i, j) \text{meets}(i, j) \equiv (\exists x, y, z) \text{interval\_map}(i, x, y) \land \text{interval\_map}(j, y, z)
\]

**Theorem 6.1** $T_{\text{linear\_point}}$ definably interprets $T_{\text{interval\_meeting}}$

**Proof:** Prover9 was used to show $T_{\text{linear\_point}} \cup \Sigma_{lp\_im} \models T_{\text{interval\_meeting}}$ □
In this direction of the translation, the size of elements in each equivalence class is fixed since only two points is mapped to an interval. However, when attempting the translation in the opposite direction (i.e. set of intervals to a point) the size of the equivalence classes are not fixed, but are in fact infinite. This currently poses a problem in determining the correct set of translation definitions to quantify over the set of equivalence classes. Thus, the interpretability of $T_{\text{linear point}}$ in $T_{\text{interval meeting}}$ remains an open issue.

### 6.1.2 Relationship between $T_{\text{linear point}}$ and $T_{\text{ap}}$

While $T_{\text{ap}}$ and $T_{\text{interval meeting}}$ are not definably equivalent theories due to their own set of unique ontological commitments (see Lemma 4.9 and Theorem 4.6), they both represent time as intervals. Thus, the same $\text{interval map}$ relation and axioms is used in this set of translation definitions to interpret $\text{timeinterval}^{\text{ap}}$ as $\text{timepoint}^{\text{linear point}}$.

**Definition 6.2** The translation definitions $\Sigma_{\text{lp}, \text{ap}}$ for the interpretation of theories in $H_{\text{Approximate–Point}}$ by theories in $H_{\text{Point}}$ is the set of sentences:

\[
(\forall i, j, x, y) \text{interval map}(i, x, y) \land \text{interval map}(j, x, y) \supset (i = j)
\]

\[
(\forall i, x, y, z, w) \text{interval map}(i, x, y) \land \text{interval map}(i, z, w) \supset (x = z) \land (y = w)
\]

\[
(\forall i, x, y) \text{interval map}(i, x, y) \supset \text{timeinterval}(i) \land \text{timepoint}(x) \land \text{timepoint}(y)
\]

\[
(\forall x, y) \text{before}(x, y) \equiv (\exists i)\text{interval map}(i, x, y)
\]

\[
(\forall i) \text{timeinterval}(i) \equiv (\exists x, y)\text{interval map}(i, x, y) \land \text{before}(x, y)
\]

\[
(\forall i, j) \text{meets}(i, j) \equiv (\exists x, y, z)\text{interval map}(i, x, y) \land \text{interval map}(j, y, z)
\]

\[
(\forall i, j) \text{precedes}(i, j) \equiv
\]

\[
(\exists x, y, z, w)\text{interval map}(i, x, y) \land \text{interval map}(j, z, w) \land (\text{before}(y, z) \lor (y = z))
\]

\[
(\forall i, j) \text{finer}(i, j) \equiv
\]

\[
(\exists x, y, z, w)\text{interval map}(i, x, y) \land \text{interval map}(j, z, w) \land \text{before}(z, x) \land \text{before}(y, w)
\]

**Theorem 6.2** $T_{\text{linear point}}$ definably interprets $T_{\text{ap}}$
Proof: Prover9 was used to show $T_{\text{linear point}} \cup \Sigma_{lp-ap} \models T_{ap}$ □

Interpretability in the opposite direction is left open due to the same issue of regarding translating equivalence classes of intervals in one theory to points in the other discussed for $T_{\text{interval-meeting}}$.

![Figure 6.2: Relationships between timepoint theories and timeinterval theories. Clouds define the boundaries of each hierarchy. Single-arrowed solid-lines denote definable interpretation.]

6.2 Exploiting the Network of Previously Proven Relationships

In this section we explore the relationship between the hierarchies of hybrid theories of time $H_{\text{Endpoints}}$, $H_{\text{Vector-Continuum}}$, and $H_{\text{Point-Continuum}}$ to the hierarchy of time as points $H_{\text{Point}}$ and the hierarchy of time as intervals $H_{\text{Interval-Meeting}}$. Although it is perfectly acceptable to discover these relationships in the default manner (i.e. via theorem proving), our network of relationships established between the theories in question is rich enough for us to begin exploiting the transitive property of those relationships. Here we demonstrate the re-usability of our initial work of verifying relationships via theorem proving, to identify new relationships between theories of interest.

6.2.1 Hybrid-time theories and $T_{\text{linear point}}$

When looking at the relation between each hybrid-time theory and the theory of $T_{\text{linear point}}$, we need not look further than their set of axioms to determine the relationship they share. We notice that the axioms of $T_{\text{linear point}}$ form a subset of each axiomatization of the hybrid-time theories. Therefore, all the hybrid-time theories examined trivially entail $T_{\text{linear point}}$. 
Theorem 6.3 \( T_{\text{endpoints}} \) definably interprets \( T_{\text{linear point}} \)

Theorem 6.4 \( T_{\text{vector continuum}} \) definably interprets \( T_{\text{linear point}} \)

Theorem 6.5 \( T_{\text{point continuum}} \) definably interprets \( T_{\text{linear point}} \)

Figure 6.3: Relationships between hybrid-time, timepoint and timeinterval theories. Clouds define the boundaries of each hierarchy. Single-arrowed solid-lines denote definable interpretation.

6.2.2 Hybrid-time theories and \( T_{\text{interval meeting}} \)

To understand the relationship between each hybrid-time theory and the theory of \( T_{\text{interval meeting}} \), we can look to the theorems already proven in this chapter. In the case of relating \( T_{\text{endpoints}} \) to \( T_{\text{interval meeting}} \), we look at Theorem 6.3 to show that \( T_{\text{endpoints}} \) definably interprets \( T_{\text{linear point}} \) and Theorem 6.1 to show that \( T_{\text{linear point}} \) definably interprets \( T_{\text{interval meeting}} \). Since the notion of definable interpretation is transitive, we can now state:

Theorem 6.6 \( T_{\text{endpoints}} \) definably interprets \( T_{\text{interval meeting}} \)

Similarly for \( T_{\text{vector continuum}} \) and \( T_{\text{point continuum}} \):
Theorem 6.7 \( T_{\text{vector-continuum}} \) definably interprets \( T_{\text{interval-meeting}} \)

Theorem 6.8 \( T_{\text{point-continuum}} \) definably interprets \( T_{\text{interval-meeting}} \)

Through the chaining of the meta-theoretic relationships associated with interpretation, we can identify the relationships between the hybrid-time theories and every other theory already integrated with \( T_{\text{interval-meeting}} \) and \( T_{\text{linear-point}} \). Thus, as more theories are integrated in the repository only a small subset of the total possible relationships must be proven via the automated reasoner before we are able to utilize the transitive nature of those relationships to achieve complete integration with no additional work.

### 6.3 Extending COLORE to support Mapping Modules

With all the work done to uncover the relationships between theories, it is necessary for COLORE to store the proofs associated with verifying each relationship in an accessible manner to users of the repository. The repository requires a mechanism for users to view these relationships and associated verification proofs. Currently, COLORE consists only of modules of CLIF ontologies and the metadata for each one. Incorporating proofs for relationships between theories within the theory-module itself would be unwieldy as not only do they require their own set of metadata, but also because it is possible that a single theory may have many relationships to other theories in the repository. Furthermore, we want to preserve the integrity of the stored ontologies so alteration of the theory-module each time a new relationship is found is not reasonable. Thus, it appears that the current theory-module structure is insufficient for storing the data for the meta-theoretic relationships.

Instead, we propose an extension to the COLORE architecture that remains in line with the modular nature of the repository and of the CLIF syntax with what we call a mapping-module. This mapping module would be associated with a CLIF module that contained the required translation definition between the theories being related and contain its own set of metadata to include information about the automated reasoner used and the input/output files used to verify the relationship. One mapping-module would exist per direction required of the relationship between pairs theories (i.e one for each direction of interpretation). This provides an optimal level of granularity, since files related to verifying the relationship remains independent from the theory-modules themselves and other mapping-modules. This
allows users of the repository to update a specific relationship without unintentionally compromising the integrity of the related theories, or other relationships.

A mapping-module would consist of the following metadata:

**Module Name** - Name of the mapping-module.

**CLIF Author** - Author responsible for the translation definitions and proofs contained in the mapping-module.

**Maps** - Name of the interpreting theory.

**Definitional Extension of** - Name of the theory being interpreted.

**Axioms** - Associated CLIF file containing the axioms of the translation definition.

**Proof Input File(s)** - Associated automated reasoner input file for the proofs required for verification.

**Proof Output File(s)** - Associated automated reasoner output file for the proofs required for verification.

### 6.4 Summary

We have completed the set of relationships between the various ontologies of time in this chapter. Included are the translation definitions required for interpreting theories of time intervals in a theory of time points, allowing us to integrate the theories of intervals with those of points. In the second half of the chapter, we showed how existing relationships between theories in the repository can be utilized in completing the network of relationships between all theories; that is, we can exploit the transitive nature of interpretation through the chaining of translation definitions to determine relationships between theories that are indirectly connected. This is an important feature of the repository as it greatly reduces the workload in completing the set of direct relationships between all relatable hierarchies. Furthermore, we proposed an extension to COLORE that facilitates the storage of proofs associated with each relationship between theories, allowing users to independently verify the relationships shown. This is an important mechanism that allows the repository to maintain its integrity and increase user confidence.
Chapter 7

Conclusion

Throughout this thesis we have proven numerous relationships between ontologies of time that either verify the models of their axiomatizations against classes of intended structures or show how they can be semantically integrated, all within the context of an ontology repository. In doing so, we sought to answer the larger question of whether the set of time ontologies we have provide a complete characterization of time itself.

We began with the theory of $T_{\text{linear point}}$, that axiomatized time as a set of points, and proved the set of representation theorems between it and the mathematical theory of $\text{infinite linear ordering}$. Thus, we were able to verify our first time ontology by showing that the models of the axioms are exactly those that were intended by the ontology. Also, it was here that we examined the issue of interpreting between a theory with class-quantified relations and a theory without such restrictions on its relations. We presented the method used to resolve this issue by defining a domain class in the translation definitions between the theories and reformulating the queries to incorporate domain so that class-quantification is respected by the translations.

Next we moved on to the set of ontologies that represented time as intervals, and proved the relationships both between theories within the same hierarchy and between theories across different hierarchies. With the relationships between theories in the same hierarchy defined, we were able to draw a connection between the differences in their models and their different ontological commitments. For example, $T_{\text{allen hayes}}$ nonconservatively extends $T_{\text{interval meeting}}$ by stipulating that no duplicate intervals can exist, and so $T_{\text{allen hayes}}$ is the stronger theory of the two in the hierarchy of $\mathbb{H}_{\text{Interval Meeting}}$. We
also showed how we are able to use relationships between theories of different hierarchies to build new theories in one hierarchy that are definably equivalent to theories in the other through the translation of their axioms via translation definitions. For example, we specified a new theory $T_{ap\text{-rational}}$ in $\mathbb{H}_{\text{Approximate-Point}}$ that is definably equivalent to $T_{\text{INT}(\mathbb{Q})}$ in $\mathbb{H}_{\text{Periods}}$. This chapter illustrated how a repository that supports these meta-theoretic relationships can be used to semantically integrate the different ontologies found in the same domain.

The last set of time ontologies we looked at were those that represented time as both points and intervals. Here we found that the theories of $T_{\text{endpoints}}$ and $T_{\text{vector\_continuum}}$ belonged to the same hierarchy since the semantics of their nonlogical lexicon were equivalent, but were mutually inconsistent theories. We were then able to define a new theory that captured their similarity. Next, we discovered that the models of $T_{\text{point\_continuum}}$ are the disjoint union of the models of the mutually inconsistent theories of $T_{\text{endpoints}}$ and $T_{\text{vector\_continuum}}$. However, to find the relationships between these theories, we had to address the issue surrounding the lack of support for namespacing by the CLIF syntax. We discussed the various circumstances where namespacing is required in the context of an ontology repository with the functionality of COLORE, and examined the consequences of using the current CLIF syntax under those conditions.

Finally we looked at the relationships across different perspectives of time and was able to prove that the ontology that represented time-as-points $T_{\text{linear\_point}}$ is able to interpret models of time as intervals as axiomatized by $T_{\text{interval\_meeting}}$ and $T_{\text{ap}}$. We also showed that each of the hybrid-time theories interpreted both the point-theory of $T_{\text{linear\_point}}$ and the interval-theory of $T_{\text{interval\_meeting}}$. To do so, we were able to leverage the existing network of proven relationships between theories along with the transitive property of those relationships, illustrating how the task of identifying a complete network of relationships between theories in the repository becomes easier as the number of relationships proven increases. Thus, not only do the ontologies in the repository become more reusable as more relationships are defined, but so do the relationships themselves.

To come back to our initial goal of fully characterizing the models of time, we can say that not only have we come along way in doing so, but also that we are very close to that result. With the current set of relationships between time theories we have verified ontologies of time as timepoints and those of hybrid-time (with the verification of specific time interval theories covered by ladkin in [17] and
van Benthem in [22]). Furthermore, we have almost all the translation definitions between the different hierarchies. This allows us to determine the semantic integration possibilities between all of the theories that we do have a complete set of translation definitions for (e.g. we’ve been able to create definably equivalent theories across the hierarchies of $H_{\text{Periods}}$ and $H_{\text{Approximate-Point}}$). However, while we’ve managed to take large steps forward toward our goal of finding the complete characterization of models of time, there remain a few open issues before we can make that claim.

7.1 Future Work

The last barriers to completing our work on fully characterizing models of time are the tasks of finding the remaining translation definitions between hierarchies of time ontologies (specifically the axioms required to interpret models of timepoints in timeinterval theories) and with it the remaining relationships between theories, such that if we look at any two theories of time we can determine the relationship between their models. If we can then show that between every pair of time theories there exists a common subtheory and a common extension, then we have shown that all the representations of time can be made equivalent.

In addition, we want to be able to extend the methodology of proving meta-theoretic relationships used here to the entire set of ontologies stored in COLORE. We would require a full method for determining the relationship between any two theories across relatable hierarchies and the relationship between any two hierarchies within the same hierarchy. Along these lines, we would need to make the process of using automated reasoners more efficient in returning results and if possible, automate the process of discovering relationships between theories when provided with the translation definitions. While the techniques of using lemmas and excluding unnecessary axioms employed to increase the efficiency of the automated reasoner proved extremely useful for our work, they are highly manual in nature and this does not scale well with the workload required for finding the large number of relationships between theories stored in COLORE. With these meta-theoretic relationships in place, COLORE functionality can also be expanded for applications in ontology engineering and design, especially in the area of semantic integration.
Bibliography


## Appendix A

### Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{H}_{\text{Periods}}$</td>
<td>The hierarchy of Periods theories.</td>
</tr>
<tr>
<td>$T_{\text{linear point}}$</td>
<td>The set of axioms that constitute the theory of linear point.</td>
</tr>
<tr>
<td>$\mathcal{L}(T_{\text{linear point}})$</td>
<td>The language (nonlogical lexicon) of $T_{\text{linear point}}$ (e.g. <em>before</em> and <em>timepoints</em>).</td>
</tr>
<tr>
<td>$\text{Mod}(T_{\text{linear point}})$</td>
<td>The set of models that satisfy the axioms of $T_{\text{linear point}}$.</td>
</tr>
<tr>
<td>$\Sigma_{\text{ilo},lp}$</td>
<td>The axioms that constitute the translation definitions from $\mathcal{L}(T_{\text{linear point}})$ to $\mathcal{L}(T_{\text{infinite linear ordering}})$.</td>
</tr>
<tr>
<td>$\mathfrak{m}_{\text{infinite linear ordering}}$</td>
<td>The class of mathematical structures of infinite linear ordering.</td>
</tr>
<tr>
<td>$\text{linear point}$</td>
<td>The original set of axioms for the theory of linear point found in [14].</td>
</tr>
<tr>
<td>$\text{endpoints}$</td>
<td>The original set of axioms for the theory of endpoints found in [14].</td>
</tr>
<tr>
<td>$\text{vector continuum}$</td>
<td>The original set of axioms for the theory of vector continuum found in [14].</td>
</tr>
<tr>
<td>$\text{point continuum}$</td>
<td>The original set of axioms for the theory of point continuum found in [14].</td>
</tr>
</tbody>
</table>
Appendix B

Glossary

**Common Logic** - Common Logic is a logic framework intended for information exchange and transmission. It defines an abstract syntax and an associated model-theoretic semantics for a specific extension of first-order logic. The intent is that the content of any system using first-order logic can be represented in this International Standard (adapted from [5]).

**first-order logic** - First-order logic is a sound and complete logical system that allows for the use of quantifiers.

**elementary equivalence** - Elementary equivalence exists between two or more theories when they possess the same set of first-order sentences as consequences [6].

**model** - We refer to a model of an ontology in the sense of a satisfying interpretation of the ontology’s axiomatization; a Tarskian model as in [6].

**structure** - A structure for a first-order language tells us the sets of things the universal quantifier symbol refers to, and what the predicate and function symbols denote. It can be thought of as providing the dictionary for translations from the formal language into English [6].

**mapping** - The translation between members of a theory in one language, to members of a theory in another language such that the semantics are preserved [6].
Appendix C

Common Logic Theories

C.1 \( T_{linear\_point} \)

(cl-module linear_point.clif

(forall (x y z)
 (if (and (before x y)
 (before y z))
 (before x z)))

(forall (x)
 (not (before x x)))

(forall (x y)
 (or (before x y)
 (before y x)
 (= x y)))

(forall (x y)
 (if (before x y)
 (and (timepoint x)
 (timepoint y)))

(forall (x)
\( (\exists y) \) \\
(\text{and} \ (\text{timepoint} \ y) \) \\
(\text{before} \ x \ y)))))

(foreall \ (x) \\
(exists \ (y) \\
(\text{and} \ (\text{timepoint} \ y) \) \\
(\text{before} \ y \ x))))))

\textbf{C.2} \ T_{\text{infinite\_linear\_ordering}}

(cl-module \ partial-ordering.clif \\

(foreall \ (x) \\
(\text{leq} \ x \ x))

(foreall \ (x \ y) \\
 \quad (\text{if} \quad (\text{and} \ (\text{leq} \ x \ y) \\
 (\text{leq} \ y \ x)) \\
 (= \ x \ y)))))

(foreall \ (x \ y \ z) \\
 \quad (\text{if} \quad (\text{and} \ (\text{leq} \ x \ y) \\
 \quad \quad (\text{leq} \ y \ z)) \\
 \quad \quad (\text{leq} \ x \ z)))

(foreall \ (x \ y) \\
(\text{iff} \ (\text{lt} \ x \ y) \\
(\text{and} \ (\text{leq} \ x \ y) \\
(\text{not} \ (\text{leq} \ y \ x))))))

(foreall \ (x) \\
(exists \ (y) \\
(\text{lt} \ x \ y)))
(forall (x)
(exists (y)
(lt y x))))

C.3 \( T_{\text{periods,oot}} \)

(cl-module periods_root.clif

(cl-comment "Precedence TRANS")
(forall (x y z)
(if (and (precedence x y)
(precedence y z))
(precedence x z)))

(cl-comment "Precedence IREFF")
(forall (x)
(not (precedence x x)))

(cl-comment "Inclusion TRANS")
(forall (x y z)
(if (and (inclusion x y)
(inclusion y z))
(inclusion x z)))

(cl-comment "Inclusion REF")
(forall (x)
(inclusion x x))

(cl-comment "Inclusion ANTIS")
(forall (x y)
(if (and (inclusion x y)
(inclusion y x))
(= x y)))
APPENDIX C. COMMON LOGIC THEORIES

\[(\text{cl-comment \"MON-1\")}
\[\text{(forall (x y z)}\]
\[\text{(if (and (precedence x y)}\]
\[\text{(inclusion z x)}\]
\[\text{(precedence z y))})\]

\[(\text{cl-comment \"MON-2\")}
\[\text{(forall (x y z)}\]
\[\text{(if (and (precedence x y)}\]
\[\text{(inclusion z y)}\]
\[\text{(precedence x z))})\]
\]

C.4 \(T_{\text{periods}}\)

\[(\text{cl-module periods.clif}\)

\[(\text{cl-imports periods_root.clif})\]

\[(\text{cl-comment \"Overlaps\")}
\[\text{(forall (x y)}\]
\[\text{(iff (overlaps x y)}\]
\[\text{(exists (z)}\]
\[\text{(and (inclusion z x)}\]
\[\text{(inclusion z y))))))\]

\[(\text{cl-comment \"Greatest Lower Bound\")}
\[\text{(forall (x y z)}\]
\[\text{(iff (glb x y z)}\]
\[\text{(and (inclusion z x)}\]
\[\text{(inclusion z y)}\]
\]
(forall (u)
  (if (and (inclusion u x)
           (inclusion u y))
   (inclusion u z)))))

(cl-comment "CONJ")
(forall (x y)
  (if (overlaps x y)
   (exists (z)
    (glb x y z)))))
)

\section*{C.5 $T_{mixed\_periods}$}

(cl-module periods_mixed.clif

(cl-imports periods.clif)

(cl-comment "SUCC-1")
(forall (x)
  (exists (y)
    (precedence x y)))

(cl-comment "SUCC-2")
(forall (x)
  (exists (y)
    (precedence y x)))

(cl-comment "NEIGH-1")
(forall (x y)
  (if (precedence x y)
   (exists (w)
    (and (precedence x w)
     ...
(not (exists (z)
(and (precedence x z)
(precedence z w))))

(cl-comment "NEIGH-2")
(forall (x y)
(if (precedence y x)
(exists (w)
(and (precedence w x)
(not (exists (z)
(and (precedence w z)
(precedence z x))))))

(cl-comment "Underlaps")
(forall (x y)
(iff (underlaps x y)
(exists (z)
(and (inclusion x z)
(inclusion y z))))

(cl-comment "Lower Upper Bound")
(forall (x y z)
(iff (lub x y z)
(and (inclusion x z)
(inclusion y z)
(forall (u)
(if (and (inclusion x u)
(inclusion y u))
(inclusion z u))))))

(cl-comment "DISJ")
(forall (x y)
(if (underlaps x y)
(exists (z)
(lub x y z))))
(cl-comment "FREE")
(forall (x y)
(if (not (inclusion x y))
(exists (z)
(and (inclusion z x)
(not (overlaps z y))))))

(cl-comment "DIR")
(forall (x y)
(exists (z)
(and (inclusion x u)
(inclusion y u))))

(cl-comment "MOND-1")
(forall (x y z)
(if (and (precedence x y)
(precedence z y))
(exists (w)
(and (glb x z w)
(precedence w y))))))

(cl-comment "MOND-2")
(forall (x y z)
(if (and (precedence y x)
(precedence y z))
(exists (w)
(and (glb x z w)
(precedence y w))))))

(cl-comment "CONV")
(forall (x y z)
(if (and (precedence x y)
(precedence y z))
(forall (u)
(if (and (inclusion x u)
(inclusion z u))
(inclusion y u))))))

(cl-comment "LIN**")
(forall (x y)
(or (precedence x y)
(precedence y x)
(overlaps x y)))
)

C.6 \( T_{INT(\mathbb{Q})} \)

(cl-module periods_over_rationals.clif

(cl-imports periods_mixed.clif)

(cl-comment "DENS**")
(forall (x)
(exists (y1 y2)
(and (precedence y1 y2)
(lub y1 y2 x)))))

(cl-comment "ORIENT")
(forall (x y)
(if (overlaps x y)
(or (= x y)
(and (inclusion x y)
(or (exists (z1)
(and (precedence x z1)
(lub x z1 y)))))
(exists (z2)
(and (precedence z2 x)
(lub z2 x y))))


(exists (z3 z4 z5)
(and (precedence z3 x)
(precedence x z4)
(lub z3 x z5)
(lub z5 z4 y)))))
(and (inclusion y x)
(or (exists (u1)
(precedence y u1)
(lub y u1 x))
(exists (u2)
(precedence u2 y)
(lub u2 y x))
(exists (u3 u4 u5)
(precedence u3 y)
(precedence y u4)
(lub u3 y u5)
(lub u5 u4 x)))))
(exists (i1 i2)
(precedence i1 i2)
(lub i1 i2 x)
(inclusion i2 y)
(exists (i3)
(lub i2 i3 y)
(precedence x i3)))))
(exists (k1 k2)
(precedence k1 k2)
(lub k1 k2 y)
(precedence k1 x)
(inclusion k2 x)
(exists (k3)
(lub k2 k3 x)
(precedence y k3)))))))
C.7  $T_{\text{INT}(\mathbb{Z})}$

(cl-module periods_over_integers.clif
(cl-imports periods_mixed.clif)
(cl-comment "ATOM")
(forall (x)
(exists (y)
(and (inclusion y x)
(forall (z)
(if (inclusion z y)
(= x y)))))))

C.8  $T_{\text{ap\_root}}$

(cl-module ap_root.clif
(forall (x y z)
(if (and (precedes x y)
(precedes y z))
(precedes x z)))
(forall (x y z)
(if (and (finer x y)
(finier y z))
(finier x z)))
(forall (x y)
(if (and (finer x y)
(finier y x))
(= x y)))
(forall (x)
(not (precedes x x)))
(forall (x) (finer x x))

(forall (x y) (if (finer x y) (not (precedes x y))))

(forall (x y z) (if (and (finer x y) (precedes y z)) (precedes x z)))

\textbf{C.9} \quad T_{ap}

(cl-module approximate-point.clif

(forall (x y z) (if (and (precedes x y) (precedes y z)) (precedes x z)))

(forall (x y z) (if (and (finer x y) (finer y z)) (finer x z)))

(forall (x y) (if (and (finer x y) (finer y x)) (= x y)))

(forall (x) (not (precedes x x)))

(forall (x)
(finer x x))

(forall (x y)
  (or (ncdf x y)
    (precedes x y)
    (precedes y x))))

(forall (x y)
  (if (finer x y)
    (not (precedes x y))))

(forall (x y z)
  (if (and (finer x y)
            (precedes y z))
    (precedes x z))))

(forall (x y)
  (iff (ncdf x y)
       (exists (z)
        (and (finer z x)
             (finer z y)))))

(forall (x)
  (exists (y)
           (precedes y x)))

(forall (x)
  (exists (y)
           (precedes x y)))

(forall (x y)
  (exists (z)
           (and (finer x z)
                (finer y z))))
**Appendix C. Common Logic Theories**

**C.10 $T_{dense\_ap}$**

(cl-module approximate-dense-point.clif

(cl-imports approximate-point.clif)

(forall (x y)
  (iff (meets x y)
    (and (precedes x y)
      (forall (z)
        (if (and (ncdf x z)
          (ncdf z y))
          (exists (u)
            (and (finer u z)
              (ncdf x u)
              (ncdf u y))))))))

(forall (x)
  (exists (y)
    (and (finer y x)
      (not (finer x y))))))
)

**C.11 $T_{discrete\_ap}$**

(cl-module approximate-discrete-point.clif

(cl-imports approximate-point.clif)

(forall (x y)
  (iff (meets x y)
    (and (precedes x y)
      (not (exists (z)
        (and (precedes x z)
          (and (precedes x z)
            (precedes z y))))))))
(forall (x)
  (iff (moment x)
    (not (exists (y)
      (and (finer y x)
        (not (= x y)))))))

(forall (x)
  (exists (y)
    (and (meets x y)
      (moment y))))

(forall (x)
  (exists (y)
    (and (meets y x)
      (moment y))))
)

C.12  \textit{T}_{ap\_interval}

(cl-module ap_interval.clif

(cl-imports approximate_point.clif)

(cl-comment "Glb")
(forall (x y z)
  (iff (glb x y z)
    (and (finer z x)
      (finer z y)
      (forall (u)
        (if (and (finer u x)
                    (finer u y)
                    (finer u z)))))))

(cl-comment "CONJ")
(forall (x y)
  (if (ncdf x y)
    (exists (z)
      (glb x y z))))

(cl-comment "Underlaps")
(forall (x y)
  (iff (underlaps x y)
    (exists (z)
      (and (finer x z)
        (finer y z))))))

(cl-comment "Lub")
(forall (x y z)
  (iff (lub x y z)
    (and (finer x z)
      (finer y z)
      (forall (u)
        (if (and (finer x u)
          (finer y u))
          (finer z u))))))

(cl-comment "DISJ")
(forall (x y)
  (if (underlaps x y)
    (exists (z)
      (lub x y z))))

(cl-comment "FREE")
(forall (x y)
  (if (not (finer x y))
    (exists (z)
      (and (finer z x)
        (not (ncdf z y))))))

(cl-comment "MOND-1")
(forall (x y z)
  (if (and (precedes x y)
            (precedes z y))
    (exists (w)
      (and (glb x z w)
           (precedes w y))))))

(cl-comment "MOND-2")
(forall (x y z)
  (if (and (precedes y x)
            (precedes y z))
    (exists (w)
      (and (glb x z w)
           (precedes y w))))))

(cl-comment "NEIGH-1")
(forall (x y)
  (if (precedes x y)
    (exists (w)
      (and (precedes x w)
           (not (exists (z)
                     (and (precedes x z)
                          (precedes z w)))))))

(cl-comment "NEIGH-2")
(forall (x y)
  (if (precedence y x)
    (exists (w)
      (and (precedes w x)
           (not (exists (z)
                     (and (precedes w z)
                          (precedes z x)))))))

(cl-comment "CONV")
(forall (x y z)
  (if (and (precedes x y)
           (not (exists (w)
                     (and (precedes w y)
                          (glb x w w)))))
    (exists (w)
      (and (precedes w y)
           (not (glb x w w))))))
(precedes y z))
(forall (u)
   (if (and (finer x u)
             (finer z u))
      (finer y u))))
)

\textbf{C.13} \ T_{\text{ap\_rational}}

(cl-module ap_rational.clif
(cl-imports approximate_dense_point.clif)
(cl-imports ap_interval.clif))

\textbf{C.14} \ T_{\text{ap\_integer}}

(cl-module ap_integer.clif
(cl-imports approximate_discrete_point.clif)
(cl-imports ap_interval.clif))

\textbf{C.15} \ T_{m\_exist}

(cl-module m_exist.clif
(cl-imports approximate_point.clif)
(forall (i j)
   (if (not (meets i j))
      (or (exists (x)
            (and (meets i x)
                 (meets x j)))
         (exists (y)
   )

)
(and (meets j y)
(meets y i)))))))

C.16 \( T_{\text{meets\_root}} \)

(cl-module meets_root.clif

(forall (i j)
(if (meets i j)
(and (timeinterval i)
(timeinterval j))))

(forall (i j k m)
(if (and (meets i k)
(meets j k)
(meets i m))
(meets j m))

(forall (i)
(exists (j k)
(and (meets j i)
(meets i k))))

C.17 \( T_{\text{interval\_meeting}} \)

(cl-module interval_meeting.clif

(forall (i j)
(if (meets i j)
(and (timeinterval i)
(timeinterval j))))

(forall (i j k m)
(if (and (meets i k)
(meets i k))))
(meets j k)
(meets i m))
(meets j m))

(forall (i)
(exists (j k)
(and (meets j i)
(meets i k))))

(forall (i j k m)
(if (and (meets i j)
(meets k l))
(or (meets i l)
(exists (n)
(or (and (meets i n)
(meets n l))
(and (meets k n)
(meets n j)))))))

(forall (i j)
(if (meets i j)
(not (meets j i))))

(forall (i j k m)
(if (and (meets i j)
(meets j k)
(meets k m))
(exists (n)
(and (meets i n)
(meets n m))))))

(forall (i)
(if (moment i)
(forall (j k)
(not (= i (plus j k)))))

APPENDIX C. COMMON LOGIC THEORIES
(forall (i j)
  (iff (precedes i j)
    (exists (k)
      (and (meets i k)
        (meets k j))))))

(forall (i j)
  (iff (overlaps i j)
    (exists (k m n o p)
      (and (meets k m)
        (meets m n)
        (meets n o)
        (meets o p)
        (meets m j)
        (meets j p)
        (meets k i)
        (meets i o))))))

(forall (i j)
  (iff (starts i j)
    (exists (k m n)
      (and (meets k i)
        (meets i m)
        (meets m n)
        (meets k j)
        (meets j n))))))

(forall (i j)
  (iff (during i j)
    (exists (k m n o)
      (and (meets k m)
        (meets m i)
        (meets i n)
        (meets n o)
        (meets k j)
        (meets j o))))))
\[(\forall (i \ j)\\ (\text{iff} \ (\text{finishes} \ i \ j)\\ (\exists (k \ m \ n)\\ (\text{and} \ (\text{meets} \ k \ m)\\ (\text{meets} \ m \ i)\\ (\text{meets} \ m \ i)\\ (\text{meets} \ i \ n)\\ (\text{meets} \ k \ j)\\ (\text{meets} \ j \ n))))))\\

\textbf{C.18} \ \ T_{\text{allen\_hayes}}\\

(cl-module allen\_hayes.clif\\
(cl-imports interval\_meeting.clif)\\
(forall (p \ q \ r \ s)\\ (\text{if} \ (\text{and} \ (\text{meets} \ p \ q)\\ (\text{meets} \ q \ s)\\ (\text{meets} \ p \ r)\\ (\text{meets} \ r \ s)\\ (= \ q \ r)))))\\

\textbf{C.19} \ \ T_{\text{ladkin\_intq}}\\

(cl-module ladkin\_intq.clif\\
(cl-imports allen\_hayes.clif)\\
(forall (p \ q \ r \ s)\\ (\text{iff} \ (\text{equiv} \ p \ q \ r \ s)\\ (\text{and} \ (\text{meets} \ p \ q)\\ (\text{meets} \ r \ s)\\ (\text{meets} \ p \ s))))
C.20  \textit{Sim}(\textit{endpoints}, \textit{vector} \textit{continuum})

(cl-module sim_ep_vc.clif
(cl-imports linear_point.clif)

(forall (i)
  (if (timeinterval i)
    (and (timepoint (beginof i))
     (timepoint (endof i))))))

(forall (p q)
  (if (before p q)
    (= p (beginof (between p q)))
    (= q (endof (between p q))))))

(forall (x y z)
  (if (meets-at x y z)
    (timeinterval x)
    (timeinterval z)
    (timepoint y))))

(forall (i p j)
(iff (meets-at i p j)
(and (= p (endof i))
(= p (beginof j))))

(forall (p q)
(iff (before p q)
(exists (i j k)
(and (meets-at i p j)
(meets-at j q k))))))

(forall (i j)
(iff (meets i j)
(and (timeinterval i)
(timeinterval j)
(= (endof i) (beginof j))))

(forall (x)
(iff (moment x)
(= (beginof x) (endof x))))

(forall (i)
(iff (forwards i)
(before (beginof i) (endof i)))

(forall (i)
(= (back i) (between (endof i) (beginof i))))

(forall (x)
(iff (moment x)
(and (timepoint x)
(timeinterval x)))
(forall (x)
(or (timepoint x)
(timeinterval x)))

(forall (i x y)
(if (and (= (beginof i) (x)) (= (endof i)) (y)) (= (between x y) (i)))))

C.21 $T_{no\textit{backwards}}$

(cl-module backwards.clif
(cl-imports sim_ep_vc.clif)
(forall (il)
(if (timeinterval il)
(exists (i2)
(and (timeinterval i2)
(= (beginof i2)
(endof il))
(= (endof i2)
(beginof il))))))

C.22 $T_{backwards}$

(cl-module no_backwards.clif
(cl-imports sim_ep_vc.clif)
(forall (il)
(if (timeinterval il)
(not (exists (i2)
(and (timeinterval i2)
(= (beginof i2)
(endof il))
(= (endof i2)
(beginof il)))))))
C.23 $T_{\text{no\_moment}}$

(cl-module no_moment.clif
(cl-imports sim_ep_vc.clif)
(forall (t)
  (if (timepoint t)
    (not (exists (i)
      (and (timeinterval i)
        (= (beginof i) (t))
        (= (endof i) (t))))))))

C.24 $T_{\text{moment}}$

(cl-module moment.clif
(cl-imports sim_ep_vc.clif)
(forall (t)
  (if (timepoint t)
    (exists (i)
      (and (timeinterval i)
        (= (beginof i) (t))
        (= (endof i) (t))))))

C.25 $T_{\text{endpoints}}$

(cl-module endpoints.clif
(cl-imports linear_point.clif)
(forall (i)
  (if (timeinterval i)
    (and (timepoint (beginof i))
      (timepoint (endof i))))))
(forall (p q)
  (iff (before p q)
      (and (= p (beginof (between p q)))
         (= q (endof (between p q))))))

(forall (x y z)
  (if (meets-at x y z)
      (and (timeinterval x)
           (timeinterval z)
           (timepoint y))))

(forall (i p j)
  (iff (meets-at i p j)
      (and (= p (endof i))
           (= p (beginof j))))))

(forall (i j)
  (iff (meets i j)
      (and (timeinterval i)
           (timeinterval j)
           (= (endof i) (beginof j))))))

(forall (i j)
  (iff (precedes i j)
      (and (timeinterval i j)
           (before (endof i) (beginof j))))))

(forall (i j)
  (iff (overlaps i j)
      (and (timeinterval i j)
           (before (beginof i) (beginof j))
           (before (beginof j) (endof i))))))

(forall (i j)
  (iff (before i j)
      (and (timeinterval i j)
           (before (beginof i) (beginof j))
           (before (beginof j) (endof i))))))

(forall (i j)
  (iff (after i j)
      (and (timeinterval i j)
           (after (endof i) (beginof j))
           (after (endof j) (beginof i))))))

(forall (i j)
  (iff (during i j)
      (and (timeinterval i j)
           (between (beginof i) (endof i))
           (between (beginof j) (endof j))))))

(forall (i j)
  (iff (starts i j)
      (and (timeinterval i j)
           (startof i)
           (beginof i) (endof i))
           (before (endof i) (beginof j))))

(forall (i j)
  (iff (finishes i j)
      (and (timeinterval i j)
           (finishof i)
           (beginof i) (endof i))
           (before (endof i) (beginof j))))

(forall (i j)
  (iff (meets-at i j)
      (and (timeinterval i)
           (timeinterval j)
           (meets i j)
           (meets-at i j)
           (meets-at j i))
(iff (starts i j)
  (and (timeinterval i j)
      (= (beginof i) (beginof j))
      (before (endof i) (endof j))))

(forall (i j)
  (iff (during i j)
        (and (timeinterval i j)
             (before (beginof j) (beginof i))
             (before (endof i) (endof j))))))

(forall (i j)
  (iff (finishes i j)
       (and (timeinterval i j)
            (before (beginof j) (beginof i))
            (= (endof i) (endof j))))))

(forall (x)
  (or (timepoint x)
      (timeinterval x)))

(forall (x)
  (if (timepoint x)
      (not (timeinterval x))))

(forall (x y)
  (if (before x y)
      (timeinterval (between x y))))
)

C.26 \( T_{vector\_continuum} \)

(cl-module vector_continuum.clif
(cl-imports linear_point.clif)

(forall (i)
  (if (timeinterval i)
      (and (timepoint (beginof i))
           (timepoint (endof i))))

(forall (p q)
  (and (= p (beginof (between p q)))
       (= q (endof (between p q)))))

(forall (i j)
  (iff (meets i j)
       (and (timeinterval i)
            (timeinterval j)
            (= (endof i) (beginof j))))

(forall (x y z)
  (if (meets-at x y z)
      (and (timeinterval x)
           (timeinterval z)
           (timepoint y))))

(forall (i p j)
  (iff (meets-at i p j)
       (and (= p (endof i))
            (= p (beginof j))))

(forall (p q)
  (iff (before p q)
       (exists (i j k)
               (and (meets-at i p j)
                    (meets-at j q k))))

(forall (i j)
(iff (meets i j)
(and (timeinterval i)
(timeinterval j)
(= (endof i) (beginof j)))))

(forall (i j)
(if (meets i j)
(= (plus i j) (between (beginof i) (endof j))))))

(forall (x)
(iff (moment x)
(= (beginof x) (endof x))))

(forall (i)
(iff (forwards i)
(before (beginof i) (endof i))))

(forall (i)
(= (back i) (between (endof i) (beginof i))))

(forall (x)
(iff (moment x)
(and (timepoint x)
(timeinterval x))))

(forall (x)
(or (timepoint x)
(timeinterval x)))

(forall (i x y)
(if (and (= (beginof i) (x))
(= (endof i) (y))
(= (between x y) (i))))))}
C.27 \( T_{point\_continuum} \)

(cl-module point_continuum.clif

(cl-imports linear_point.clif)

(forall (x y)
  (if (in x y)
    (and (timepoint x)
      (timeinterval y)))))

(forall (i)
  (if (timeinterval i)
    (or (and (open i)
      (not (closed i))
      (forall (p)
        (iff (in p i)
          (and (before (beginof i) p)
            (before p (endof i))))))
    (and (closed i)
      (not (open i)))
    (forall (p)
      (iff (in p i)
        (and (bbefore (beginof i) p)
          (bbefore p (endof i)))))))))

(forall (p q)
  (if (and (timepoint p)
      (timepoint q))
    (or (before q p)
      (exists (i)
(and (timeinterval i)
(closed i)
(= (beginof i) p)
(= (endof i) q))))

(forall (p q)
(if (and (timepoint p)
(timepoint q))
(if (before p q)
(exists (i)
(and (timeinterval i)
(open i)
(= (beginof i) p)
(= (endof i) q))))))

(forall (p q)
(if (and (timepoint p)
(timepoint q))
(if (bbefore p q)
(and (timeinterval (between p q))
(= p (beginof (between p q)))
(= q (endof (between p q)))))

(forall (i)
(if (timeinterval i)
(and (timeinterval (closure i))
(closed (closure i))
(= (beginof i) (beginof (closure i)))
(= (endof i) (endof (closure i))))))

(forall (i j)
(if (acoao i j)
(if (and (timeinterval i)
(timeinterval j))
(or (and (open i)
(open j))
(forall (i j)
  (iff (meets i j)
    (if (and (timeinterval i)
            (timeinterval j))
      (and (not (acoao i j))
        (= (endof i) (beginof j))))))

(forall (i j)
  (iff (starts i j)
    (if (and (timeinterval i)
             (timeinterval j))
      (and (acoao i j)
        (= (beginof i) (beginof j))
        (before (endof i) (endof j))))))

(forall (i j)
  (iff (finishes i j)
    (if (and (timeinterval i)
             (timeinterval j))
      (and (acoao i j)
        (before (beginof j) (beginof i))
        (= (endof i) (endof j))))))

(forall (x)
  (or (timepoint x)
      (timeinterval x)))

(forall (i)
  (if (open i)
      (timeinterval i)))

(forall (i)
  (if (closed i)
(forall (i)
  (if (timeinterval i)
    (and (timepoint (beginof i))
    (timepoint (endof i))))))
)

C.28 \( T_{\text{strict\_graphical}} \)

(cl-module strict-graphical

(forall (x y)
  (if (in x y)
    (in y x)))

(forall (x)
  (in x x))

(forall (p)
  (if (point p)
    (not (line p))))

(forall (x y)
  (if (and (in x y)
    (point x)
    (point y))
    (= x y)))

(forall (x y)
  (if (and (in x y)
    (line x)
    (line y)))


(= x y)))

(cl-comment "A line is incident with at most two points.")
(forall (x y z w)
 (if (and (point x)
 (point y)
 (point z)
 (line w)
 (in x w)
 (in y w)
 (in z w))
 (or (= z x)
 (= z y))))

(forall (l)
 (if (line l)
 (exists (p)
 (and (point p)
 (in p l))))))

(cl-comment "Two points are in at most one line.")
(forall (p1 p2 11 12)
 (if (and (in p1 11)
 (in p1 12)
 (not (= p1 p2))
 (in p2 11)
 (in p2 12))
 (= 11 12))
)

C.29 $T_{\text{strong\_graphical}}$

(cl-module strict-graphical
(forall (x y)
  (if (in x y)
    (in y x))
)

(forall (x)
  (in x x))

(forall (p)
  (if (point p)
    (not (line p)))))

(forall (x y)
  (if (and (in x y)
            (point x)
            (point y))
    (= x y))
)

(forall (x y)
  (if (and (in x y)
            (line x)
            (line y))
    (= x y))
)

(cl-comment "A line is incident with at most two points."
  (forall (x y z w)
    (if (and (point x)
             (point y)
             (point z)
             (line w)
             (in x w)
             (in y w)
             (in z w))
      (or (= z x)
          (= z y)))))

(forall (x)
(if (point x)
(exists (y)
(and (line y)
(in x y)
(forall (z)
(if (and (point z)
(in z y))
(= z x)))))))
)

C.30 \( T_{\text{double-complete}} \)

(cl-module double-complete

(forall (x y)
(if (in_g x y)
(in_g y x)))

(forall (x)
(in_g x x))

(forall (p)
(if (point p)
(not (line p))))

(forall (x y)
(if (and (in_g x y)
(point x)
(point y))
(= x y)))

(forall (x y)
(if (and (in_g x y)
(line x)
(line y))
(= x y))

(forall (l)
 (if (line l)
  (exists (p)
   (and (point p)
    (in_g p l))))))

(forall (x y z w)
 (if (and (point x)
           (point y)
           (point z)
           (line w)
           (in_g x w)
           (in_g y w)
           (in_g z w))
  (or (= z x)
      (= z y)
      (= x y))))

(forall (x y)
 (if (and (point x)
           (point y)
           (not (= x y)))
  (exists (l)
   (and (line l)
    (in_g x l)
    (in_g y l))))))

(forall (l)
 (if (line l)
  (exists (p1 p2)
   (and (point p1)
    (point p2)
    (in_g p1 l)
    (in_g p2 l))))))
(forall (x y)
  (if (in_d x y)
      (in_d y x))))

(forall (x)
  (in_d x x))

(forall (x y)
  (if (and (in_d x y)
           (point x)
           (point y))
       (= x y)))

(forall (x y)
  (if (and (in_d x y)
           (line x)
           (line y))
       (= x y)))

(forall (l)
  (if (line l)
      (exists (p)
               (and (point p)
                    (in_d p l))))))

(forall (x y z w)
  (if (and (point x)
            (point y)
            (point z)
            (line w)
            (in_d x w)
            (in_d y w)
            (in_d z w))
      (or (= z x)
          (= z y)
(= x y))))

(forall (x y)
(if (and (point x)
(point y)
(not (= x y)))
(exists (l)
(and (line l)
(in_d x l)
(in_d y l))))))

(forall (l)
(if (line l)
(exists (p1 p2)
(and (point p1)
(point p2)
(in_d p1 l)
(in_d p2 l))))))

(forall (p1 p2 l1 l2)
(if (and (point p1)
(point p2)
(line l1)
(line l2)
(in_d p1 l1)
(in_d p1 l2)
(in_d p2 l1)
(in_d p2 l2)
(not (= p1 p2))
(in_d p2 l1)
(in_d p2 l2))
(= l1 l2)))

)
Appendix D

Prover9 Proof Output

D.1 Theorem 3.1

\[ T_{linear\ point} \cup \Sigma_{lp\_ilo} \models T_{infinite\ linear\ ordering} \]

================================================================================ PROOF =================================================================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 6.
% Level of proof is 3.
% Maximum clause weight is 6.000.
% Given clauses 6.

6 (all x all y (leq(x,y) <-> before(x,y) | x = y)) # label(non_clause).
[assumption].
7 (all x (domain(x) -> leq(x,x))) # label(non_clause) # label(goal). [goal].
17 leq(x,y) | y != x. [clausify(6)].
18 -leq(c1,c1). [deny(7)].
25 leq(x,x). [xx_res(17,b)].
26 $F. [resolve(25,a,18,a)].

================================================================================ end of proof =================================================================================
% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 15.
% Level of proof is 4.
% Maximum clause weight is 9.000.
% Given clauses 5.

1 (all x all y all z (before(x,y) & before(y,z) -> before(x,z)))
   # label(non_clause). [assumption].

2 (all x -before(x,x)) # label(non_clause). [assumption].

6 (all x all y (leq(x,y) <-> before(x,y) | x = y)) # label(non_clause).
   [assumption].

7 (all x all y (domain(x) & domain(y) -> (leq(x,y) & leq(y,x) -> x = y)))
   # label(non_clause) # label(goal). [goal].

13 -leq(x,y) | before(x,y) | y = x. [clausify(6)].

15 leq(c1,c2). [deny(7)].

16 leq(c2,c1). [deny(7)].

17 -before(x,y) | -before(y,z) | before(x,z). [clausify(1)].

18 -before(x,x). [clausify(2)].

20 c2 != c1. [deny(7)].

21 before(c1,c2) | c2 = c1. [resolve(15,a,13,a)].

22 before(c1,c2). [copy(21),unit_del(b,20)].

23 before(c2,c1) | c1 = c2. [resolve(16,a,13,a)].

24 before(c2,c1). [copy(23),flip(b),unit_del(b,20)].

31 $F. [ur(17,b,22,a,c,18,a),unit_del(a,24)].

============================== end of proof ==========================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 27.
% Level of proof is 8.
% Maximum clause weight is 9.000.
% Given clauses 31.

1 (all x all y all z (before(x,y) & before(y,z) -> before(x,z)))
   # label(non_clause). [assumption].
2 (all x -before(x,x)) # label(non_clause). [assumption].
3 (all x all y (before(x,y) | before(y,x) | x = y))
   # label(non_clause). [assumption].
6 (all x all y (leq(x,y) <-> before(x,y) | x = y)) # label(non_clause).
5 # label(goal). [assumption].
7 (all x all y (domain(x) & domain(y) & domina(z) ->
   (leq(x,y) & leq(y,z) -> leq(x,z)))) # label(non_clause) # label(goal). [goal].
12 -before(x,y) | -before(y,z) | before(x,z). [clausify(1)].
13 -before(x,x). [clausify(2)].
14 before(x,y) | before(y,x) | y = x. [clausify(3)].
15 -leq(x,y) | before(x,y) | y = x. [clausify(6)].
16 leq(x,y) | -before(x,y). [clausify(6)].
17 leq(x,y) | y != x. [clausify(6)].
18 leq(c2,c3). [deny(7)].
19 leq(c3,c1). [deny(7)].
20 -leq(c2,c1). [deny(7)].
28 before(c2,c3) | c3 = c2. [resolve(18,a,15,a)].
29 before(c3,c1) | c3 = c1. [resolve(19,a,15,a),flip(b)].
30 c2 != c1. [ur(17,a,20,a),flip(a)].
31 -before(c2,c1). [ur(16,a,20,a)].
34 before(c1,c2). [resolve(31,a,14,b),unit_del(b,30)].
37 -before(x,c1) | before(x,c2). [resolve(34,a,12,b)].
38 -before(c2,x) | before(c1,x). [resolve(34,a,12,a)].
40 c3 = c2 | -before(x,c2) | before(x,c3). [resolve(28,a,12,b)].
49 before(c3,c2) | c3 = c1. [resolve(37,a,29,a)].
50 before(c1,c3) | c3 = c2. [resolve(38,a,28,a)].
74 c3 = c2 | c3 = c1. [resolve(40,b,49,a),unit_del(b,13)].
76 c3 = c1. [para(74(a,1),19(a,1)),unit_del(b,20)].
78 $F.
[back_rewrite(50), rewrite([(76(2),76(4))]), flip(b), unit_del(a,13), unit_del(b,30)].

================================ end of proof =================================

================================ PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.02 (+ 0.00) seconds.
% Length of proof is 23.
% Level of proof is 8.
% Maximum clause weight is 12.000.
% Given clauses 25.

1 (all x all y all z (before(x,y) & before(y,z) -> before(x,z))) # label(non_clause). [assumption].
2 (all x -before(x,x)) # label(non_clause). [assumption].
3 (all x all y (before(x,y) | before(y,x) | x = y)) # label(non_clause). [assumption].
6 (all x all y (leq(x,y) <-> before(x,y) | x = y)) # label(non_clause). [assumption].
7 (all x all y (lt(x,y) <-> leq(x,y) & -leq(y,x))) # label(non_clause). [assumption].
8 (all x all y (domain(x) & domain(y) & domina(z) -> lt(x,y) | lt(y,x) | x = y)) # label(non_clause) # label(goal). [goal].
13 -before(x,y) | -before(y,z) | before(x,z). [clausify(1)].
14 -before(x,x). [clausify(2)].
15 before(x,y) | before(y,x) | y = x. [clausify(3)].
16 -leq(x,y) | before(x,y) | y = x. [clausify(6)].
17 leq(x,y) | -before(x,y). [clausify(6)].
21 lt(x,y) | -leq(x,y) | leq(y,x). [clausify(7)].
22 -lt(c2,c3). [deny(8)].
23 -lt(c3,c2). [deny(8)].
24 c3 != c2. [deny(8)].
29 leq(x,y) | before(y,x) | x = y. [resolve(17,b,15,b)].
35 before(x,y) | y = x | lt(y,x) | leq(x,y). [resolve(29,a,21,b)].
APPENDIX D. PROVER9 PROOF OUTPUT

39 before(c2,c3) | leq(c2,c3). [resolve(35,c,23,a),unit_del(b,24)].
40 before(c3,c2) | leq(c3,c2). [resolve(35,c,22,a),flip(b),unit_del(b,24)].
42 before(c2,c3). [resolve(39,b,16,a),merge(b),unit_del(b,24)].
48 -before(c3,c2). [ur(13,b,42,a,c,14,a)].
49 leq(c3,c2). [back_unit_del(40),unit_del(a,48)].
70 $F. [ur(16,b,48,a,c,24,a(flip)),unit_del(a,49)].

============================== end of proof ==========================

% -------- Comments from original proof --------
% Proof 1 at 0.03 (+ 0.00) seconds.
% Length of proof is 28.
% Level of proof is 7.
% Maximum clause weight is 9.000.
% Given clauses 93.

1 (all x all y all z (before(x,y) & before(y,z) -> before(x,z)))
# label(non_clause). [assumption].
2 (all x -before(x,x)) # label(non_clause). [assumption].
4 (all x all y (before(x,y) -> timepoint(x) & timepoint(y)))
# label(non_clause). [assumption].
5 (all x (domain(x) <-> timepoint(x))) # label(non_clause). [assumption].
6 (all x all y (leq(x,y) <-> before(x,y) | x = y)) # label(non_clause). [assumption].
7 (all x all y (lt(x,y) <-> leq(x,y) & -leq(y,x))) # label(non_clause). [assumption].
8 (all x exists y before(x,y)) # label(non_clause). [assumption].
9 (all x exists y before(y,x)) # label(non_clause). [assumption].
10 (all x (domain(x) -> (exists y (domain(y) & lt(y,x)))))
# label(non_clause) # label(goal). [goal].
11 domain(x) | -timepoint(x). [clausify(5)].
12 -before(x,y) | timepoint(x). [clausify(4)].
15 -domain(x) | -lt(x,c1). [deny(10)].
17 domain(x) | -before(x,y). [resolve(11,b,12,b)].
19 -before(x,y) | -before(y,z) | before(x,z). [clausify(l)].
20 \ -\text{before}(x,x). \ [\text{clausify}(2)].
22 \ -\text{leq}(x,y) \ | \ \text{before}(x,y) \ | \ y = x. \ [\text{clausify}(6)].
23 \ \text{leq}(x,y) \ | \ -\text{before}(x,y). \ [\text{clausify}(6)].
27 \ \text{lt}(x,y) \ | \ -\text{leq}(x,y) \ | \ \text{leq}(y,x). \ [\text{clausify}(7)].
28 \ \text{before}(x,f_1(x)). \ [\text{clausify}(8)].
29 \ \text{before}(f_2(x),x). \ [\text{clausify}(9)].
31 \ -\text{before}(x,y) \ | \ -\text{lt}(x,c_1). \ [\text{resolve}(17,a,15,a)].
44 \ \text{leq}(f_2(x),x). \ [\text{resolve}(29,a,23,b)].
47 \ -\text{before}(x,f_2(x)). \ [\text{ur}(19,b,29,a,c,20,a)].
49 \ -\text{lt}(x,c_1). \ [\text{ur}(31,a,28,a)].
63 \ \text{lt}(f_2(x),x) \ | \ \text{leq}(x,f_2(x)). \ [\text{resolve}(44,a,27,b)].
329 \ \text{leq}(c_1,f_2(c_1)). \ [\text{resolve}(63,a,49,a)].
330 \ f_2(c_1) = c_1. \ [\text{resolve}(329,a,22,a),\text{unit_del}(a,47)].
331 \ $F. \ [\text{para}(330(a,1),29(a,1)),\text{unit_del}(a,20)].

============================== end of proof ===========================

============================== PROOF ===============================
Appendix D. Prover9 Proof Output

10 (all x (domain(x) -> (exists y (domain(y) & lt(x,y))))))

# label(non_clause) # label(goal). [goal].

11 domain(x) | -timepoint(x). [clausify(5)].

12 -before(x,y) | timepoint(x). [clausify(4)].

15 -domain(x) | -lt(cl1,x). [deny(10)].

17 domain(x) | -before(x,y). [resolve(11,b,12,b)].

19 -before(x,y) | -before(y,z) | before(x,z). [clausify(1)].

20 -before(x,x). [clausify(2)].

22 -leq(x,y) | before(x,y) | y = x. [clausify(6)].

23 leq(x,y) | -before(x,y). [clausify(6)].

27 lt(x,y) | -leq(x,y) | leq(y,x). [clausify(7)].

28 before(x,f1(x)). [clausify(8)].

31 -before(x,y) | -lt(cl1,x). [resolve(17,a,15,a)].

40 leq(x,f1(x)). [resolve(28,a,23,b)].

43 -before(f1(x),x). [ur(19,b,28,a,c,20,a)].

49 -lt(cl1,x). [ur(31,a,28,a)].

61 lt(x,f1(x)) | leq(f1(x),x). [resolve(40,a,27,b)].

279 leq(f1(cl1),cl1). [resolve(61,a,49,a)].

330 f1(cl1) = cl1. [resolve(279,a,22,a),flip(b),unit_del(a,43)].

331 $F. [para(330(a,1),28(a,2)),unit_del(a,20)].

============================== end of proof ==========================

T_{\text{infinite\_linear\_ordering}} \cup \Sigma_{ilo\_lp} \vDash T_{\text{linear\_point}}

============================================= PROOF ==================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 5.
% Level of proof is 2.
% Maximum clause weight is 2.000.
% Given clauses 0.

7 (all x timepoint(x)) # label(non_clause). [assumption].

8 (all x all y (before(x,y) -> timepoint(x) & timepoint(y)))
# label(non_clause) # label(goal). [goal].
19 timepoint(x). [clausify(7)].
20 ~timepoint(c1) | ~timepoint(c2). [deny(8)].
21 $F$. [copy(20), unit_del(a, 19), unit_del(b, 19)].

============================== end of proof ==========================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 17.
% Level of proof is 5.
% Maximum clause weight is 9.000.
% Given clauses 26.

3 (all x all y all z (leq(x,y) & leq(y,z) -> leq(x,z)))
# label(non_clause). [assumption].
6 (all x all y (before(x,y) <-> leq(x,y) & ~leq(y,x)))
# label(non_clause). [assumption].
8 (all x all y all z (before(x,y) & before(y,z) -> before(x,z)))
# label(non_clause) # label(goal). [goal].
11 ~leq(x,y) | ~leq(y,z) | leq(x,z). [clausify(3)].
15 ~before(x,y) | leq(x,y). [clausify(6)].
16 ~before(x,y) | ~leq(y,x). [clausify(6)].
17 before(x,y) | ~leq(x,y) | leq(y,x). [clausify(6)].
18 before(c1,c2). [deny(8)].
19 before(c2,c3). [deny(8)].
20 ~before(c1,c3). [deny(8)].
28 leq(c1,c2). [resolve(18,a,15,a)].
29 ~leq(c3,c2). [resolve(19,a,16,a)].
30 leq(c2,c3). [resolve(19,a,15,a)].
36 ~leq(c2,x) | leq(c1,x). [resolve(28,a,11,a)].
37 ~leq(c3,c1). [ur(11,b,28,a,c,29,a)].
40 leq(c1,c3). [resolve(36,a,30,a)].
43 $F. \ [ur(17,a,20,a,c,37,a),unit\_del(a,40)].

================================================ end of proof ==============

================================================ PROOF ==========================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 3.000.
% Given clauses 0.

1 (all x leq(x,x)) # label(non_clause). [assumption].
6 (all x all y (before(x,y) <-> leq(x,y) & -leq(y,x)))
# label(non_clause). [assumption].
8 (all x -before(x,x)) # label(non_clause) # label(goal). [goal].
11 -before(x,y) | -leq(y,x). [clausify(6)].
12 before(c1,c1). [deny(8)].
13 leq(x,x). [clausify(1)].
19 -leq(c1,c1). [resolve(12,a,11,a)].
20 $F. [copy(19),unit\_del(a,13)].

================================================ end of proof ==============

================================================ PROOF ==========================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 18.
% Level of proof is 6.
% Maximum clause weight is 9.000.
% Given clauses 21.
APPENDIX D. PROVER9 PROOF OUTPUT

2 (all x all y (leq(x,y) & leq(y,x) -> x = y))
# label(non_clause). [assumption].
4 (all x all y (lt(x,y) -> leq(x,y) & -leq(y,x)))
# label(non_clause). [assumption].
5 (all x all y (lt(x,y) | lt(y,x) | x = y))
# label(non_clause). [assumption].
6 (all x all y (before(x,y) <-> leq(x,y) & -leq(y,x)))
# label(non_clause). [assumption].
8 (all x all y (before(x,y) | before(y,x) | x = y))
# label(non_clause) # label(goal). [goal].
10 -leq(x,y) | -leq(y,x) | y = x. [clausify(2)].
12 -lt(x,y) | leq(x,y). [clausify(4)].
14 lt(x,y) | lt(y,x) | y = x. [clausify(5)].
17 before(x,y) | -leq(x,y) | leq(y,x). [clausify(6)].
18 -before(c1,c2). [deny(8)].
19 -before(c2,c1). [deny(8)].
20 c2 != c1. [deny(8)].
23 lt(x,y) | x = y | leq(y,x). [resolve(14,a,12,a)].
27 x = y | leq(y,x) | leq(x,y). [resolve(23,a,12,a)].
28 x = y | leq(x,y) | before(y,x). [resolve(27,b,17,b),merge(d)].
34 leq(c1,c2). [resolve(28,c,19,a),flip(a),unit_del(a,20)].
35 leq(c2,c1). [resolve(28,c,18,a),unit_del(a,20)].
39 SF. [resolve(34,a,10,b),flip(b),unit_del(a,35),unit_del(b,20)].

============================== end of proof ==========================

================================================================

% ------- Comments from original proof -------
% Proof 1 at 0.02 (+ 0.00) seconds.
% Length of proof is 12.
% Level of proof is 3.
% Maximum clause weight is 9.000.
% Given clauses 17.
5 (all x exists y lt(y,x)) # label(non_clause). [assumption].
7 (all x all y (before(x,y) <-> leq(x,y) & -leq(y,x)))
    # label(non_clause). [assumption].
8 (all x all y (lt(x,y) <-> leq(x,y) & -leq(y,x)))
    # label(non_clause). [assumption].
9 (all x exists y before(y,x)) # label(non_clause) # label(goal). [goal].
14 lt(x,f1(x)). [clausify(5)].
18 before(x,y) | -leq(x,y) | leq(y,x). [clausify(7)].
19 -lt(x,y) | leq(x,y). [clausify(8)].
20 -lt(x,y) | -leq(y,x). [clausify(8)].
22 -before(c1,x). [deny(9)].
25 leq(x,f1(x)). [resolve(19,a,14,a)].
29 -leq(f1(x),x). [resolve(20,a,14,a)].
36 $F. [ur(18,a,22,a,c,29,a),unit_del(a,25)].

============================================= end of proof =================================

Back to Proof Output

% ------- Comments from original proof -------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 12.
% Level of proof is 3.
% Maximum clause weight is 9.000.
% Given clauses 16.
D.2 Lemma 3.1

\[ T_{\text{periods}} \cup \sum_{ap} \not\models T_{ap} \]

%%%%%%%%%%%%%%%%%%%%%%%% PROOF %%%%%%%%%%%%%%%%%%%%%%%%%

\% -------- Comments from original proof --------
\% Proof 1 at 0.02 (+ 0.01) seconds.
\% Length of proof is 12.
\% Level of proof is 4.
\% Maximum clause weight is 9.000.
\% Given clauses 23.

4 (all x all y (inclusion(x,y) & inclusion(y,x) -> x = y))
# label(non_clause). [assumption].
12 (all x all y (inclusion(x,y) <-> finer(x,y))) # label(non_clause). [assumption].
13 (all x all y (finer(x,y) & finer(y,x) -> x = y))
# label(non_clause) # label(goal). [goal].
29 -inclusion(x,y) | -inclusion(y,x) | y = x. [clausify(4)].
35 inclusion(x,y) | -finer(x,y). [clausify(12)].
36 finer(c1,c2). [deny(13)].
37 finer(c2,c1). [deny(13)].
38 c2 != c1. [deny(13)].
48 inclusion(c1,c2). [hyper(35,b,36,a)].
49 inclusion(c2,c1). [hyper(35,b,37,a)].
66 -inclusion(c2,c1). [ur(29,b,48,a,c,38,a(flip))].
67 $F$. [resolve(66,a,49,a)].
% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 3.000.
% Given clauses 0.

2 (all x -precedence(x,x)) # label(non_clause). [assumption].
11 (all x all y (precedence(x,y) <-> precedes(x,y)))
   # label(non_clause). [assumption].
13 (all x -precedes(x,x)) # label(non_clause) # label(goal). [goal].
24 precedence(x,y) | -precedes(x,y). [clausify(11)].
26 precedes(c1,c1). [deny(13)].
28 -precedence(x,x). [clausify(2)].
45 precedence(c1,c1). [resolve(26,a,24,b)].
46 $F. [resolve(45,a,28,a)].

% -------- Comments from original proof --------
% Proof 1 at 0.02 (+ 0.00) seconds.
% Length of proof is 14.
% Level of proof is 4.
% Maximum clause weight is 9.000.
% Given clauses 24.

3 (all x all y all z (inclusion(x,y) & inclusion(y,z) -> inclusion(x,z)))
# label(non_clause). [assumption].
12 (all x all y (inclusion(x,y) <-> finer(x,y))) # label(non_clause). [assumption].
13 (all x all y all z (finer(x,y) & finer(y,z) -> finer(x,z)))
# label(non_clause) # label(goal). [goal].
28 -inclusion(x,y) | -inclusion(y,z) | inclusion(x,z). [clausify(3)].
34 -inclusion(x,y) | finer(x,y). [clausify(12)].
35 inclusion(x,y) | -finer(x,y). [clausify(12)].
36 finer(c1,c2). [deny(13)].
37 finer(c2,c3). [deny(13)].
38 -finer(c1,c3). [deny(13)].
48 inclusion(c1,c2). [hyper(35,b,36,a)].
49 inclusion(c2,c3). [hyper(35,b,37,a)].
50 -inclusion(c1,c3). [ur(34,b,38,a)].
67 -inclusion(c2,c3). [ur(28,a,48,a,c,50,a)].
68 $F$. [resolve(67,a,49,a)].

========================================= end of proof =================================

% -------- Comments from original proof --------
% Proof 1 at 0.02 (+ 0.00) seconds.
% Length of proof is 16.
% Level of proof is 4.
% Maximum clause weight is 9.000.
% Given clauses 25.

6 (all x all y all z (precedence(x,y) & inclusion(z,x) -> precedence(z,y)))
# label(non_clause). [assumption].
11 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause). [assumption].
12 (all x all y (inclusion(x,y) <-> finer(x,y))) # label(non_clause). [assumption].
13 (all x all y all z (finer(x,y) & precedes(y,z) -> precedes(x,z)))
# label(non_clause) # label(goal). [goal].
28 -precedence(x,y) | -inclusion(z,x) | precedence(z,y). [clausify(6)].
30 -precedence(x,y) | precedes(x,y). [clausify(11)].
31 precedence(x,y) | -precedes(x,y). [clausify(11)].
33 inclusion(x,y) | -finer(x,y). [clausify(12)].
34 finer(c1,c2). [deny(13)].
35 precedes(c2,c3). [deny(13)].
36 -precedes(c1,c3). [deny(13)].
46 inclusion(c1,c2). [hyper(33,b,34,a)].
47 precedence(c2,c3). [hyper(31,b,35,a)].
48 -precedence(c1,c3). [ur(30,b,36,a)].
70 -precedence(c2,c3). [ur(28,b,46,a,c,48,a)].
71 $F$. [resolve(70,a,47,a)].

========================================== end of proof =======================================

% -------- Comments from original proof --------
% Proof 1 at 0.02 (+ 0.00) seconds.
% Length of proof is 15.
% Level of proof is 4.
% Maximum clause weight is 9.000.
% Given clauses 18.

2 (all x -precedence(x,x)) # label(non_clause). [assumption].
7 (all x all y all z (precedence(x,y) & inclusion(z,y) -> precedence(x,z)))
# label(non_clause). [assumption].
11 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause). [assumption].
12 (all x all y (inclusion(x,y) <-> finer(x,y)))
# label(non_clause). [assumption].
13 (all x all y (finer(x,y) -> -precedes(x,y)))
# label(non_clause) # label(goal). [goal].
23 precedence(x,y) | -precedes(x,y). [clausify(11)].
25 precedes(c1,c2). [deny(13)].
27 -precedence(x,x). [clausify(2)].
32 \(-\text{precedence}(x,y) \lor \neg\text{inclusion}(z,y) \lor \text{precedence}(x,z)\). \text{[clausify(7)]}.
34 \text{inclusion}(x,y) \lor \neg\text{finer}(x,y). \text{[clausify(12)]}.
35 \text{finer}(c1,c2). \text{[deny(13)]}.
43 \text{precedence}(c1,c2). \text{[resolve(25,a,23,b)]}.
45 \text{inclusion}(c1,c2). \text{[hyper(34,b,35,a)]}.
50 \neg\text{inclusion}(c1,c2). \text{[ur(32,a,43,a,c,27,a)]}.
51 \$F. \text{[resolve(50,a,45,a)]}.

================================================== end of proof ==================================================

================================================== PROOF ==================================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 14.
% Level of proof is 4.
% Maximum clause weight is 9.000.
% Given clauses 27.

1 (all x all y all z (precedence(x,y) & precedence(y,z) -> precedence(x,z)))
# label(non_clause). \text{[assumption]}. 
11 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause). \text{[assumption]}. 
13 (all x all y all z (precedes(x,y) & precedes(y,z) -> precedes(x,z)))
# label(non_clause) # label(goal). \text{[goal]}. 
24 \neg\text{precedence}(x,y) \lor \neg\text{precedence}(y,z) \lor \text{precedence}(x,z). \text{[clausify(1)]}.
32 \neg\text{precedence}(x,y) \lor \text{precedes}(x,y). \text{[clausify(11)]}.
33 \text{precedes}(x,y) \lor \neg\text{precedes}(x,y). \text{[clausify(11)]}.
36 \text{precedes}(c1,c2). \text{[deny(13)]}.
37 \text{precedes}(c2,c3). \text{[deny(13)]}.
38 \neg\text{precedes}(c1,c3). \text{[deny(13)]}.
49 \text{precedence}(c1,c2). \text{[hyper(33,b,36,a)]}.
50 \text{precedence}(c2,c3). \text{[hyper(33,b,37,a)]}.
51 \neg\text{precedence}(c1,c3). \text{[ur(32,b,38,a)]}.
59 \neg\text{precedence}(c2,c3). \text{[ur(24,a,49,a,c,51,a)]}.
D.3 Lemma 4.3

\[ T_{ap} \cup \Sigma_{ap,p} \not\supseteq T_{\text{periods}} \]

\[ T_{ap} \cup \Sigma_{ap,p} \not\supseteq T_{\text{periods}} \]
% Given clauses 18.

3 (all x all y (finer(x,y) & finer(y,x) -> x = y))
# label(non_clause). [assumption].
10 (all x all y (inclusion(x,y) <-> finer(x,y)))
# label(non_clause). [assumption].
12 (all x all y (inclusion(x,y) & inclusion(y,x) -> x = y))
# label(non_clause) # label(goal). [goal].
23 -finer(x,y) | -finer(y,x) | y = x. [clausify(3)].
28 -inclusion(x,y) | finer(x,y). [clausify(10)].
30 inclusion(c1,c2). [deny(12)].
31 inclusion(c2,c1). [deny(12)].
32 c2 != c1. [deny(12)].
38 finer(c1,c2). [hyper(28,a,30,a)].
39 finer(c2,c1). [hyper(28,a,31,a)].

============================== end of proof ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 3.000.
% Given clauses 0.

4 (all x -precedes(x,x)) # label(non_clause). [assumption].
9 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause). [assumption].
12 (all x -precedence(x,x)) # label(non_clause) # label(goal). [goal].
17 -precedence(x,y) | precedes(x,y). [clausify(9)].
18 precedence(c1,c1). [deny(12)].
APPENDIX D. PROVER9 PROOF OUTPUT

25 -precedes(x,x). [clausify(4)].
33 precedes(c1,c1). [resolve(18,a,17,a)].
34 $F. [copy(33),unit_del(a,25)].

================================ end of proof ===========================

================================ PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 13.
% Level of proof is 3.
% Maximum clause weight is 9.000.
% Given clauses 19.

2 (all x all y all z (finer(x,y) & finer(y,z) -> finer(x,z)))
# label(non_clause). [assumption].
10 (all x all y (inclusion(x,y) <-> finer(x,y)))
# label(non_clause). [assumption].
12 (all x all y all z (inclusion(x,y) & inclusion(y,z) -> inclusion(x,z)))
# label(non_clause) # label(goal). [goal].
22 -finer(x,y) | -finer(y,z) | finer(x,z). [clausify(2)].
28 -inclusion(x,y) | finer(x,y). [clausify(10)].
29 inclusion(x,y) | -finer(x,y). [clausify(10)].
30 inclusion(c1,c2). [deny(12)].
31 inclusion(c2,c3). [deny(12)].
32 -inclusion(c1,c3). [deny(12)].
38 finer(c1,c2). [hyper(28,a,30,a)].
39 finer(c2,c3). [hyper(28,a,31,a)].
40 -finer(c1,c3). [ur(29,a,32,a)].
59 $F. [ur(22,a,38,a,c,40,a),unit_del(a,39)].

================================ end of proof ===========================

================================ PROOF =================================
% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 15.
% Level of proof is 3.
% Maximum clause weight is 9.000.
% Given clauses 22.

7 (all x all y all z (finer(x,y) & precedes(y,z) -> precedes(x,z)))
# label(non_clause). [assumption].
9 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause). [assumption].
10 (all x all y (inclusion(x,y) <-> finer(x,y)))
# label(non_clause). [assumption].
12 (all x all y all z (precedence(x,y) & inclusion(z,x) -> precedence(z,y)))
# label(non_clause) # label(goal). [goal].
25 -finer(x,y) | -precedes(y,z) | precedes(x,z). [clausify(7)].
26 -precedence(x,y) | precedes(x,y). [clausify(9)].
27 precedence(x,y) | -precedes(x,y). [clausify(9)].
28 -inclusion(x,y) | finer(x,y). [clausify(10)].
30 precedence(c1,c2). [deny(12)].
31 inclusion(c3,c1). [deny(12)].
32 -precedence(c3,c2). [deny(12)].
39 precedes(c1,c2). [hyper(26,a,30,a)].
40 finer(c3,c1). [hyper(28,a,31,a)].
41 -precedes(c3,c2). [ur(27,a,32,a)].
49 $F. [ur(25,b,39,a,c,41,a),unit_del(a,40)].

============================================= end of proof ==========================

% -------- Comments from original proof --------
% Proof 1 at 0.02 (+ 0.00) seconds.
% Length of proof is 38.
% Level of proof is 8.
% Maximum clause weight is 11.000.
% Given clauses 53.

1 (all x all y all z (precedes(x,y) & precedes(y,z) -> precedes(x,z)))
   # label(non_clause). [assumption].
2 (all x all y all z (finer(x,y) & finer(y,z) -> finer(x,z)))
   # label(non_clause). [assumption].
6 (all x all y (ncdf(x,y) | precedes(x,y) | precedes(y,x)))
   # label(non_clause). [assumption].
7 (all x all y (finer(x,y) -> -precedes(x,y))) # label(non_clause). [assumption].
8 (all x all y all z (finer(x,y) & precedes(y,z) -> precedes(x,z)))
   # label(non_clause). [assumption].
9 (all x all y (ncdf(x,y) <-> (exists z (finer(z,x) & finer(z,y)))))
   # label(non_clause). [assumption].
10 (all x all y (precedence(x,y) <-> precedes(x,y)))
    # label(non_clause). [assumption].
11 (all x all y (inclusion(x,y) <-> finer(x,y))) # label(non_clause). [assumption].
13 (all x all y all z (precedence(x,y) & inclusion(z,y) -> precedence(x,z)))
   # label(non_clause) # label(goal). [goal].
14 -ncdf(x,y) | finer(f1(x,y),x). [clausify(9)].
15 ncdf(x,y) | precedes(x,y) | precedes(y,x). [clausify(6)].
16 -ncdf(x,y) | finer(f1(x,y),y). [clausify(9)].
21 -inclusion(x,y) | finer(x,y). [clausify(11)].
22 inclusion(c3,c2). [deny(13)].
27 -precedes(x,y) | -precedes(y,z) | precedes(x,z). [clausify(1)].
28 -finer(x,y) | -finer(y,z) | finer(x,z). [clausify(2)].
32 -finer(x,y) | -precedes(x,y). [clausify(7)].
33 -finer(x,y) | -precedes(y,z) | precedes(x,z). [clausify(8)].
34 -precedence(x,y) | precedes(x,y). [clausify(10)].
35 precedence(x,y) | -precedes(x,y). [clausify(10)].
36 precedence(c1,c2). [deny(13)].
37 -precedence(c1,c3). [deny(13)].
38 finer(f1(x,y),x) | precedes(x,y) | precedes(y,x). [resolve(14,a,15,a)].
39 finer(f1(x,y),y) | precedes(x,y) | precedes(y,x). [resolve(16,a,15,a)].
Appendix D. Prover9 Proof Output

42 finer(c3,c2).  [resolve(22,a,21,a)].
45 precedes(c1,c2).  [resolve(36,a,34,a)].
46 -precedes(c1,c3).  [ur(35,a,37,a)].
76 -finer(x,c3) | finer(x,c2).  [resolve(42,a,28,b)].
79 -precedes(c3,c2).  [ur(32,a,42,a)].
89 finer(f1(c3,c1),c1) | precedes(c3,c1).  [resolve(46,a,39,c)].
91 finer(f1(c3,c1),c3) | precedes(c3,c1).  [resolve(46,a,38,c)].
99 -finer(x,c1) | precedes(x,c2).  [resolve(45,a,33,b)].
105 -precedes(c3,c1).  [ur(27,b,45,a,c,79,a)].
108 finer(f1(c3,c1),c3).  [back_unit_del(91),unit_del(b,105)].
110 finer(f1(c3,c1),c1).  [back_unit_del(89),unit_del(b,105)].
207 finer(f1(c3,c1),c2).  [resolve(76,a,108,a)].
257 -precedes(f1(c3,c1),c2).  [ur(32,a,207,a)].
268 $F$.  [resolve(99,a,110,a),unit_del(a,257)].

============================================= end of proof ==================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 13.
% Level of proof is 3.
% Maximum clause weight is 9.000.
% Given clauses 22.

1 (all x all y all z (precedes(x,y) & precedes(y,z) -> precedes(x,z)))
# label(non_clause).  [assumption].
9 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause).  [assumption].
12 (all x all y all z (precedence(x,y) & precedence(y,z) -> precedence(x,z)))
# label(non_clause) # label(goal).  [goal].
19 -precedes(x,y) | -precedes(y,z) | precedes(x,z).  [clausify(1)].
26 -precedence(x,y) | precedes(x,y).  [clausify(9)].
27 precedence(x,y) | -precedes(x,y).  [clausify(9)].
30 precedence(c1,c2). [deny(12)].
31 precedence(c2,c3). [deny(12)].
32 -precedence(c1,c3). [deny(12)].
39 precedes(c1,c2). [hyper(26,a,30,a)].
40 precedes(c2,c3). [hyper(26,a,31,a)].
41 -precedes(c1,c3). [ur(27,a,32,a)].
47 $F. [ur(19,a,39,a,c,41,a),unit_del(a,40)].

============================== end of proof ==========================

================================ PROOF ===============================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 6.000.
% Given clauses 9.

5 (all x finer(x,x)) # label(non_clause). [assumption].
10 (all x all y (inclusion(x,y) <-> finer(x,y)))
# label(non_clause). [assumption].
12 (all x inclusion(x,x)) # label(non_clause) # label(goal). [goal].
25 finer(x,x). [clausify(5)].
29 inclusion(x,y) | -finer(x,y). [clausify(10)].
30 -inclusion(c1,c1). [deny(12)].
35 inclusion(x,x). [hyper(29,b,25,a)].
36 $F. [resolve(35,a,30,a)].

============================== end of proof ==========================

D.4 Lemma 4.4

\[ T_{\text{mixed periods}} \cup \Sigma_{p,ap} \models T_{ap} \]
% -------- Comments from original proof --------
% Proof 1 at 0.11 (+ 0.05) seconds.
% Length of proof is 11.
% Level of proof is 4.
% Maximum clause weight is 6.
% Given clauses 73.

18 (all x all y exists z (inclusion(x,z) & inclusion(y,z)))
# label(non_clause). [assumption].
25 (all x all y (inclusion(x,y) <-> finer(x,y)))
# label(non_clause). [assumption].
27 (all x all y exists z (finer(x,z) & finer(y,z)))
# label(non_clause) # label(goal). [goal].
72 inclusion(x,f11(x,y)). [clausify(18)].
73 inclusion(x,f11(y,x)). [clausify(18)].
78 -inclusion(x,y) | finer(x,y). [clausify(25)].
80 -finer(c1,x) | -finer(c2,x). [deny(27)].
197 finer(x,f11(y,x)). [resolve(78,a,73,a)].
198 finer(x,f11(x,y)). [resolve(78,a,72,a)].
512 -finer(c1,f11(x,c2)). [resolve(197,a,80,b)].
513 $F. [resolve(512,a,198,a)].

================================ end of proof =======================

% -------- Comments from original proof --------
% Proof 1 at 0.05 (+ 0.01) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 6.
% Given clauses 25.

11 (all x exists y precedence(x,y)) # label(non_clause). [assumption].
24 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause). [assumption].
27 (all x exists y precedes(x,y)) # label(non_clause) # label(goal). [goal].
65 precedence(x,f4(x)). [clausify(11)].
78 -precedence(x,y) | precedes(x,y). [clausify(24)].
80 -precedes(c1,x). [deny(27)].
197 precedes(x,f4(x)). [resolve(78,a,65,a)].
198 $F. [resolve(197,a,80,a)].

============================== end of proof ==========================

% -------- Comments from original proof --------
% Proof 1 at 0.03 (+ 0.03) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 6.
% Given clauses 25.

10 (all x exists y precedence(y,x)) # label(non_clause). [assumption].
24 (all x all y (precedence(x,y) <-> precedes(x,y)))
# label(non_clause). [assumption].
27 (all x exists y precedes(y,x)) # label(non_clause) # label(goal). [goal].
64 precedence(f3(x),x). [clausify(10)].
78 -precedence(x,y) | precedes(x,y). [clausify(24)].
80 -precedes(x,c1). [deny(27)].
197 precedes(f3(x),x). [resolve(78,a,64,a)].
198 $F. [resolve(197,a,80,a)].

============================== end of proof ==========================
% Length of proof is 14.
% Level of proof is 3.
% Maximum clause weight is 9.
% Given clauses 72.

23 (all x all y (precedence(x,y) | precedence(y,x) | overlaps(x,y)))
   # label(non_clause). [assumption].
24 (all x all y (precedence(x,y) <-> precedes(x,y)))
   # label(non_clause). [assumption].
26 (all x all y (overlaps(x,y) <-> ncdf(x,y))) # label(non_clause). [assumption].
27 (all x all y (ncdf(x,y) | precedes(x,y) | precedes(y,x)))
   # label(non_clause) # label(goal). [goal].
75 precedence(x,y) | precedence(y,x) | overlaps(x,y). [clausify(23)].
76 -precedence(x,y) | precedes(x,y). [clausify(24)].
78 -overlaps(x,y) | ncdf(x,y). [clausify(26)].
80 -ncdf(c1,c2). [deny(27)].
81 -precedes(c1,c2). [deny(27)].
82 -precedes(c2,c1). [deny(27)].
204 -overlaps(c1,c2). [ur(78,b,80,a)].
205 -precedence(c1,c2). [ur(76,b,81,a)].
206 -precedence(c2,c1). [ur(76,b,82,a)].
411 $\forall \exists. [resolve(204,a,75,c),unit_del(a,205),unit_del(b,206)].$

----------------------------------------------- end of proof -----------------------------------------------

D.5 Lemma 4.6

$T_{INT(Q)} \cup \Sigma_{p,ap} = T_{dense,ap}$

----------------------------------------------- PROOF -----------------------------------------------

% ------- Comments from original proof -------
% Proof 1 at 0.12 (+ 0.01) seconds.
% Length of proof is 22.
% Level of proof is 5.
% Maximum clause weight is 9.
% Given clauses 85.

2 (all x -preference(x,x)) # label(non_clause). [assumption].
7 (all x all y all z (preference(x,y) & inclusion(z,y) -> preference(x,z))) # label(non_clause). [assumption].
12 (all x all y (inclusion(x,y) <-> finer(x,y))) # label(non_clause). [assumption].
18 (all x all y all z (lub(x,y,z) <-> inclusion(x,z) & inclusion(y,z) & (all u (inclusion(x,u) & inclusion(y,u) -> inclusion(z,u))))) # label(non_clause). [assumption].
27 (all x exists y exists z (preference(y,z) & lub(y,z,x))) # label(non_clause). [assumption].
28 (all x exists y (finer(y,x) & -finer(x,y))) # label(non_clause) # label(goal). [goal].
43 -lub(x,y,z) | inclusion(x,z). [clausify(18)].
44 -lub(x,y,z) | inclusion(y,z). [clausify(18)].
49 lub(f15(x),f16(x),x). [clausify(27)].
57 -preference(x,x). [clausify(2)].
62 -preference(x,y) | -inclusion(z,y) | preference(x,z). [clausify(7)].
66 -inclusion(x,y) | finer(x,y). [clausify(12)].
67 inclusion(x,y) | -finer(x,y). [clausify(12)].
81 preference(f15(x),f16(x)). [clausify(27)].
82 -finer(x,c1) | finer(c1,x). [deny(28)].
98 inclusion(f15(x),x). [resolve(49,a,43,a)].
99 inclusion(f16(x),x). [resolve(49,a,44,a)].
374 -preference(f15(x),x). [ur(62,b,98,a,c,57,a)].
405 finer(f16(x),x). [resolve(99,a,66,a)].
572 finer(c1,f16(c1)). [resolve(405,a,82,a)].
618 -inclusion(x,f16(x)). [ur(62,a,81,a,c,374,a)].
641 $F$. [resolve(572,a,67,b),unit_del(a,618)].

--------------------------------- end of proof ----------------------------------

D.6 Theorem 4.3

\[ T_{dense_{ap}} \cup \Sigma_{ap,p} \models T_{ap_rational} \]
% -------- Comments from original proof --------
% Proof 1 at 0.03 (+ 0.00) seconds.
% Length of proof is 26.
% Level of proof is 5.
% Maximum clause weight is 9.000.
% Given clauses 51.

1 (all x all y all z (precedes(x,y) & precedes(y,z) -> precedes(x,z)))
   # label(non_clause). [assumption].
4 (all x -precedes(x,x)) # label(non_clause). [assumption].
8 (all x all y all z (finer(x,y) & precedes(y,z) -> precedes(x,z)))
   # label(non_clause). [assumption].
10 (all x exists y precedes(y,x)) # label(non_clause). [assumption].
11 (all x exists y precedes(x,y)) # label(non_clause). [assumption].
13 (all x all y (precedence(x,y) <-> precedes(x,y)))
   # label(non_clause). [assumption].
14 (all x all y (inclusion(x,y) <-> finer(x,y)))
   # label(non_clause). [assumption].
16 (all x all y all z (lub(x,y,z) <->
                        (all u (inclusion(x,z) & inclusion(y,z) & inclusion(x,u) & inclusion(y,u) ->
                         inclusion(z,u))))) # label(non_clause). [assumption].
19 (all x exists y exists z (precedence(y,z) & lub(y,z,x)))
   # label(non_clause) # label(goal). [goal].
26 precedence(x,y) | -precedes(x,y). [clausify(13)].
28 -precedence(x,y) | -lub(x,y,c1). [deny(19)].
39 -precedes(x,y) | -precedes(y,z) | precedes(x,z). [clausify(1)].
42 -precedes(x,x). [clausify(4)].
45 -finer(x,y) | -precedes(y,z) | precedes(x,z). [clausify(8)].
46 precedes(f2(x),x). [clausify(10)].
47 precedes(x,f3(x)). [clausify(11)].
50 -inclusion(x,y) | finer(x,y). [clausify(14)].
54 lub(x,y,z) | inclusion(y,z). [clausify(16)].
64 -lub(x,y,c1) | -precedes(x,y). [resolve(28,a,26,a)].
D.7 Theorem 4.5

\( T_{\text{discrete,ap}} \cup \Sigma_{\text{ap,p}} \models T_{\text{ap,\text{integer}}} \)

%------- Comments from original proof -------
% Proof 1 at 12.50 (+ 0.12) seconds.
% Length of proof is 41.
% Level of proof is 9.
% Maximum clause weight is 12.000.
% Given clauses 1466.

6 (all x all y (ncdf(x,y) | precedes(x,y) | precedes(y,x))) # label(non_clause). [assumption].
9 (all x all y (ncdf(x,y) <-> (exists z (finer(z,x) & finer(z,y)))))) # label(non_clause). [assumption].
11 (all x all y (meets(x,y) <- (exists z (meets(x,z) & meets(z,y)))))) # label(non_clause). [assumption].
12 (all x (moment(x) <-> (all y (finer(y,x) -> x = y)))) # label(non_clause). [assumption].
13 (all x exists y (meets(x,y) & moment(y))) # label(non_clause). [assumption].
14 (all x exists y (meets(y,x) & moment(y))) # label(non_clause). [assumption].
16 (all x all y (inclusion(x,y) <-> finer(x,y))) # label(non_clause). [assumption].
APPENDIX D. PROVER9 PROOF OUTPUT

18 (all x exists y (inclusion(y,x) & (all z (inclusion(z,y) -> z = y)))))
# label(non_clause) # label(goal). [goal].
19 -ncdf(x,y) | finer(f1(x,y),x). [clausify(9)].
20 ncdf(x,y) | precedes(x,y) | precedes(y,x). [clausify(6)].
21 -ncdf(x,y) | finer(f1(x,y),y). [clausify(9)].
26 -meets(x,y) | precedes(x,y). [clausify(11)].
27 -meets(x,y) | -precedes(x,z) | -precedes(z,y). [clausify(11)].
29 meets(x,f5(x)). [clausify(13)].
30 meets(f6(x),x). [clausify(14)].
32 -moment(x) | -finer(y,x) | y = x. [clausify(12)].
35 moment(f6(x)). [clausify(14)].
51 -inclusion(x,y) | finer(x,y). [clausify(16)].
52 inclusion(x,y) | -finer(x,y). [clausify(16)].
53 -inclusion(x,c1) | inclusion(f7(x),x). [deny(18)].
54 -inclusion(x,c1) | f7(x) != x. [deny(18)].
55 finer(f1(x,y),x) | precedes(x,y) | precedes(y,x). [resolve(19,a,20,a)].
56 finer(f1(x,y),y) | precedes(x,y) | precedes(y,x). [resolve(21,a,20,a)].
61 precedes(x,f5(x)). [resolve(29,a,26,a)].
62 -precedes(x,y) | -precedes(y,f5(x)). [resolve(29,a,27,a)].
63 precedes(f6(x),x). [resolve(30,a,26,a)].
64 -precedes(f6(x),y) | -precedes(y,x). [resolve(30,a,27,a)].
69 -finer(x,f6(y)) | x = f6(y). [resolve(35,a,32,a)].
70 -finer(x,f6(y)) | f6(y) = x. [copy(69),flip(b)].
155 -precedes(x,f6(f5(x))). [resolve(63,a,62,b)].
172 -precedes(f6(f5(x)),x). [resolve(64,b,61,a)].
488 finer(f1(x,f6(f5(x))),f6(f5(x))). [resolve(155,a,56,b),unit_del(b,172)].
490 finer(f1(x,f6(f5(x))),x). [resolve(155,a,55,b),unit_del(b,172)].
2736 inclusion(f1(x,f6(f5(x))),x). [resolve(490,a,52,b)].
9124 f7(f1(c1,f6(f5(c1)))) != f1(c1,f6(f5(c1))). [resolve(2736,a,54,a)].
9125 inclusion(f7(f1(c1,f6(f5(c1)))),f1(c1,f6(f5(c1))). [resolve(2736,a,53,a)].
22035 f1(x,f6(f5(x))) = f6(f5(x)). [resolve(488,a,70,a),flip(a)].
22060 inclusion(f7(f6(f5(c1))),f6(f5(c1))).
[back_rewrite(9125),rewrite([22035(5),22035(9)])].
22061 f7(f6(f5(c1))) != f6(f5(c1)).
[back_rewrite(9124),rewrite([22035(5),22035(9)])].
22828 finer(f7(f6(f5(c1))),f6(f5(c1))). [resolve(22060,a,51,a)].
\( T_{\text{interval meeting}} \cup \Sigma_{\text{im ap}} \models T_{\text{ap root}} \)

------------ PROOF --------------------------
20 (all i all j (finer(i,j) <-> starts(i,j) | finishes(i,j) | during(i,j) | i = j))
# label(non_clause). [assumption].
21 -(all i all j all k (finer(i,j) & finer(j,k) -> finer(i,k)))
# label(non_clause). [assumption].
22 -during(x,y) | -during(y,z) | during(x,z). [clausify(1)].
23 -starts(x,y) | -starts(y,z) | starts(x,z). [clausify(2)].
24 -finishes(x,y) | -finishes(y,z) | finishes(x,z). [clausify(3)].
25 -starts(x,y) | -finishes(y,z) | during(x,z). [clausify(4)].
26 -starts(x,y) | -during(y,z) | during(x,z). [clausify(5)].
28 -finishes(x,y) | -starts(y,z) | during(x,z). [clausify(7)].
29 -finishes(x,y) | -during(y,z) | during(x,z). [clausify(8)].
30 -during(x,y) | -finishes(y,z) | during(x,z). [clausify(9)].
31 -during(x,y) | -starts(y,z) | during(x,z). [clausify(10)].
32 -finer(x,y) | starts(x,y) | finishes(x,y) | during(x,y) | y = x. [clausify(20)].
45 finer(x,y) | -starts(x,y). [clausify(20)].
46 finer(x,y) | -finishes(x,y). [clausify(20)].
47 finer(x,y) | -during(x,y). [clausify(20)].
48 finer(x,y) | y != x. [clausify(20)].
49 finer(c1,c2). [clausify(21)].
50 finer(c2,c3). [clausify(21)].
51 -finer(c1,c3). [clausify(21)].
56 starts(c1,c2) | finishes(c1,c2) | during(c1,c2) | c2 = c1. [resolve(49,a,44,a)].
57 starts(c2,c3) | finishes(c2,c3) | during(c2,c3) | c3 = c2. [resolve(50,a,44,a)].
58 c3 != c1. [ur(48,a,51,a)].
59 -during(c1,c3). [ur(47,a,51,a)].
60 -finishes(c1,c3). [ur(46,a,51,a)].
61 -starts(c1,c3). [ur(45,a,51,a)].
66 finishes(c1,c2) | during(c1,c2) | c2 = c1 | -during(c2,x) | during(c1,x). [resolve(56,a,26,a)].
67 finishes(c1,c2) | during(c1,c2) | c2 = c1 | -finishes(c2,x) | during(c1,x). [resolve(56,a,25,a)].
69 finishes(c1,c2) | during(c1,c2) | c2 = c1 | -starts(c2,x) | starts(c1,x). [resolve(56,a,23,a)].
71 finishes(c2,c3) | during(c2,c3) | c3 = c2 | -during(x,c2) | during(x,c3). [resolve(57,a,31,b)].
72 finishes(c2,c3) | during(c2,c3) | c3 = c2 | -finishes(x,c2) | during(x,c3).
[resolve(57,a,28,b)].
78 finishes(c1,c2) | during(c1,c2) | c2 = c1 | finishes(c2,c3) | during(c2,c3) | c3 = c2. [resolve(69,d,57,a),unit_del(d,61)].
79 finishes(c1,c2) | during(c1,c2) | c2 = c1 | during(c2,c3) | c3 = c2. [resolve(78,d,67,d),merge(f),merge(g),merge(h),unit_del(f,59)].
80 during(c1,c2) | c2 = c1 | during(c2,c3) | c3 = c2 | finishes(c2,c3). [resolve(79,a,72,d),merge(f),merge(g),unit_del(f,59)].
86 during(c1,c2) | c2 = c1 | during(c2,c3) | c3 = c2 | -finishes(c2,x) | finishes(c1,x). [resolve(79,a,24,a)].
93 during(c1,c2) | c2 = c1 | during(c2,c3) | c3 = c2. [resolve(86,e,80,e),merge(f),merge(g),merge(h),merge(i),unit_del(e,60)].
94 during(c1,c2) | c2 = c1 | c3 = c2 | finishes(c1,c2). [resolve(93,c,66,d),merge(e),merge(f),unit_del(e,59)].
102 during(c1,c2) | c2 = c1 | c3 = c2 | -during(c2,x) | during(c1,x). [resolve(94,d,29,a)].
105 during(c1,c2) | c2 = c1 | c3 = c2. [resolve(102,d,93,c),merge(e),merge(f),merge(g),unit_del(d,59)].
106 c2 = c1 | c3 = c2 | finishes(c2,c3) | during(c2,c3). [resolve(105,a,71,d),merge(e),unit_del(e,59)].
110 c2 = c1 | c3 = c2 | -during(c2,x) | during(c1,x). [resolve(105,a,22,a)].
113 c2 = c1 | c3 = c2 | during(c2,c3) | -during(x,c2) | during(x,c3). [resolve(106,c,30,b)].
117 c2 = c1 | c3 = c2 | during(c2,c3). [resolve(113,d,105,a),merge(e),merge(f),unit_del(d,59)].
118 c2 = c1 | c3 = c2. [resolve(117,c,110,c),merge(c),merge(d),unit_del(c,59)].
123 c2 = c1. [para(118(b,1),51(a,2)),unit_del(b,49)].
132 $F$. [back_rewrite(57),rewrite([[123(l),123(4),123(7),123(11)]],unit_del(a,61),unit_del(b,60),unit_del(c,59),unit_del(d,58)].

============================== end of proof ===========================

============================== PROOF ===============================

% -------- Comments from original proof --------

% Proof 1 at 0.01 (+ 0.00) seconds.

% Length of proof is 21.
% Level of proof is 8.
% Maximum clause weight is 6.000.
% Given clauses 1.

1 (all i all j (finishes(i,j) -> -precedes(i,j)))
# label(non_clause). [assumption].
2 (all i all j (starts(i,j) -> -precedes(i,j))) # label(non_clause). [assumption].
3 (all i all j (during(i,j) -> -precedes(i,j))) # label(non_clause). [assumption].
4 (all i all j (i = j -> -precedes(i,j))) # label(non_clause). [assumption].
5 (all i all j (finer(i,j) <-> starts(i,j) | finishes(i,j) | during(i,j) | i = j))
# label(non_clause). [assumption].
6 (all i all j (finer(i,j) -> -precedes(i,j))) # label(non_clause)
# label(goal). [goal].
7 -finer(x,y) | starts(x,y) | finishes(x,y) | during(x,y) | y = x. [clausify(5)].
8 -finishes(x,y) | -precedes(x,y). [clausify(1)].
10 -finer(x,y) | during(x,y) | y = x | -precedes(x,y).
[resolve(7,c,8,a)].
11 -starts(x,y) | -precedes(x,y). [clausify(2)].
13 -finer(x,y) | during(x,y) | y = x | -precedes(x,y) | -precedes(x,y).
[resolve(10,b,11,a)].
14 -during(x,y) | -precedes(x,y). [clausify(3)].
16 -finer(x,y) | y = x | -precedes(x,y) | -precedes(x,y) | -precedes(x,y).
[resolve(13,b,14,a)].
18 finer(c1,c2). [deny(6)].
19 x != y | -precedes(y,x). [clausify(4)].
20 precedes(c1,c2). [deny(6)].
21 c2 = c1 | -precedes(c1,c2) | -precedes(c1,c2) | -precedes(c1,c2).
[resolve(16,a,18,a)].
22 c2 = c1. [copy(21),merge(c),merge(d),unit_del(b,20)].
23 precedes(c1,c1). [back_rewrite(20),rewrite([22(2)])].
24 -precedes(x,x). [ur(19,a,xx)].
25 $F. [resolve(24,a,23,a)].

=============================================================================== end of proof ==============================================================

APPENDIX D. PROVER9 PROOF OUTPUT

155
% ---- Comments from original proof ----
% Proof 1 at 104.10 (+ 2.32) seconds.
% Length of proof is 61.
% Level of proof is 23.
% Maximum clause weight is 27.000.
% Given clauses 6915.

4 (all i all j all k all m (meets(i,j) & meets(j,k) & meets(k,m) ->
(exists n (meets(i,n) & meets(n,m)))))) # label(non_clause). [assumption].
6 (all x all y (precedes(x,y) <-
meets(x,y) | (exists z (meets(x,z) & meets(z,y))))))
# label(non_clause). [assumption].
8 (all x all y all z ((meets(x,y) | (exists u (meets(x,u) & meets(u,y)))) &
(meets(y,z) | (exists w (meets(y,w) & meets(w,z)))))) ->
meets(x,z) | (exists p (meets(x,p) & meets(p,z))))))
# label(non_clause) # label(goal). [goal].
13 -precedes(x,y) | meets(x,y) | meets(x,f4(x,y)). [clausify(6)].
14 -precedes(x,y) | meets(x,y) | meets(f4(x,y),y). [clausify(6)].
15 precedes(x,y) | -meets(x,z) | -meets(z,y). [clausify(6)].
22 -meets(x,y) | -meets(y,z) | -meets(z,u) | meets(x,f2(x,y,z,u)). [clausify(4)].
23 -meets(x,y) | -meets(y,z) | -meets(z,u) | meets(f2(x,y,z,u),u). [clausify(4)].
178 meets(c1,c2) | meets(c1,c4). [deny(8)].
179 meets(c1,c2) | meets(c4,c2). [deny(8)].
180 meets(c2,c3) | meets(c2,c5). [deny(8)].
181 meets(c2,c3) | meets(c5,c3). [deny(8)].
183 -meets(c1,x) | -meets(x,c3). [deny(8)].
186 -meets(x,y) | -meets(y,z) | meets(x,z) | meets(x,f4(x,z)).
[resolve(15,a,13,a)].
187 -meets(x,y) | -meets(y,z) | meets(x,z) | meets(f4(x,z),z).
[resolve(15,a,14,a)].
216 meets(c1,c2) | -meets(c4,x) | -meets(x,y) | meets(f2(c1,c4,x,y),y).
[resolve(178,b,23,a)].
219 meets(c1,c2) | -meets(c4,x) | -meets(x,y) | meets(c1,f2(c1,c4,x,y)).
[resolve(178,b,22,a)].
326  meets(c2,c3) | -meets(x,c2) | -meets(c5,y) | meets(f2(x,c2,c5,y),y). [resolve(180,b,23,b)].
329  meets(c2,c3) | -meets(x,c2) | -meets(c5,y) | meets(x,f2(x,c2,c5,y)). [resolve(180,b,22,b)].
425  -meets(c4,x) | meets(c1,x) | meets(c1,f4(c1,x)) | meets(c1,c2). [resolve(186,a,178,b)].
431  -meets(c4,c2) | meets(c1,x) | meets(c1,f4(c1,c2)). [factor(425,b,d)].
437  -meets(c4,x) | meets(c1,x) | meets(f4(c1,x),x) | meets(c1,c2). [resolve(187,a,178,b)].
443  -meets(c4,c2) | meets(c1,c2) | meets(f4(c1,c2),c2). [factor(437,b,d)].
1991 meets(c1,c2) | meets(c1,f4(c1,c2)). [resolve(431,a,179,b),merge(c)].
2042 meets(c1,c2) | -meets(f4(c1,c2),x) | -meets(x,y) | meets(f2(c1,f4(c1,c2),x),y). [resolve(1991,b,23,a)].
2045 meets(c1,c2) | -meets(f4(c1,c2),x) | -meets(x,y) | meets(c1,f2(c1,f4(c1,c2),x),y). [resolve(1991,b,22,a)].
2152 meets(c1,c2) | meets(f4(c1,c2),c2). [resolve(443,a,179,b),merge(c)].
3622 meets(c1,c2) | -meets(c2,x) | meets(f2(c1,f4(c1,c2),c2,x),x). [resolve(216,b,179,b),merge(d)].
3636 meets(c1,c2) | -meets(c2,x) | meets(c1,f2(c1,c4,c2,x)). [resolve(219,b,179,b),merge(d)].
7474 meets(c1,c2) | meets(f2(c1,f4(c1,c2),c2),c5) | meets(c2,c3). [resolve(3622,b,180,b)].
7482 meets(c1,c2) | meets(c1,f2(c1,c4,c2,c5)) | meets(c2,c3). [resolve(3636,b,180,b)].
7569 meets(c1,c2) | meets(c2,c3) | -meets(x,f2(c1,c4,c2,c5)) | -meets(c5,y) | meets(f2(x,f2(c1,c4,c2,c5),c5,y),y). [resolve(7474,b,23,b)].
7572 meets(c1,c2) | meets(c2,c3) | -meets(x,f2(c1,c4,c2,c5)) | -meets(c5,y) | meets(x,f2(x,f2(c1,c4,c2,c5),c5,y)). [resolve(7474,b,22,b)].
12671 meets(c2,c3) | -meets(x,c2) | meets(f2(x,c2,c5,c3),c3). [resolve(326,c,181,b),merge(d)].
13093 meets(c2,c3) | -meets(x,c2) | meets(x,f2(x,c2,c5,c3)). [resolve(329,c,181,b),merge(d)].
30425 meets(c1,c2) | -meets(c2,x) | meets(f2(c1,f4(c1,c2),c2,x),x). [resolve(2042,b,2152,b),merge(d)].
30439 meets(c1,c2) | meets(f2(c1,f4(c1,c2),c2,c3),c3) | meets(c1,f2(c1,c4,c2,c5)). [resolve(30425,b,7482,b),merge(c)].
30743 meets(c1,c2) | meets(c1,f2(c1,c4,c2,c5)) | -meets(c1,f2(c1,f4(c1,c2),c2,c3)).
[resolve(30439,b,183,b)].

31002 meets(c1,c2) | -meets(c2,x) | meets(c1,f2(c1,f4(c1,c2),c2,x)).
[resolve(2045,b,2152,b),merge(d)].

31017 meets(c1,c2) | meets(c1,f2(c1,f4(c1,c2),c2,c3)) | meets(c1,f2(c1,c4,c2,c5)).
[resolve(31002,b,7482,c),merge(c)].

31535 meets(c1,c2) | meets(c1,f2(c1,c4,c2,c5)).
[resolve(31017,b,30743,c),merge(c),merge(d)].

45068 meets(c1,c2) | meets(c2,c3) | -meets(c5,x) | meets(f2(c1,f2(c1,c4,c2,c5),c5,x),x).
[resolve(7569,c,31535,b),merge(e)].

45072 meets(c1,c2) | meets(c2,c3) | -meets(c5,x) | meets(c1,f2(c1,f2(c1,c4,c2,c5),c5,x)).
[resolve(7572,c,31535,b),merge(e)].

45073 meets(c1,c2) | meets(c2,c3) | meets(f2(c1,f2(c1,c4,c2,c5),c5,c3),c3).
[resolve(45068,c,181,b),merge(d)].

45202 meets(c1,c2) | meets(c2,c3) | -meets(c1,f2(c1,f2(c1,c4,c2,c5),c5,c3)).
[resolve(45073,c,183,b)].

45208 meets(c1,c2) | meets(c2,c3) | meets(c1,f2(c1,f2(c1,c4,c2,c5),c5,c3)).
[resolve(45207,c,181,b),merge(d)].

45222 meets(c1,c2) | meets(c1,f2(c1,f2(c1,c4,c2,c5),c5,c3)) | meets(c1,f2(c1,c4,c2,c3)).
[resolve(45208,b,3636,b),merge(c)].

45223 meets(c1,c2) | meets(c1,f2(c1,f2(c1,c4,c2,c5),c5,c3)) | meets(f2(c1,c4,c2,c3),c3).
[resolve(45208,b,3622,b),merge(c)].

45371 meets(c1,c2) | meets(c1,f2(c1,c4,c2,c3)) | meets(c2,c3).
[resolve(45222,b,45202,c),merge(c)].

45433 meets(c1,c2) | meets(c1,f2(c1,c4,c2,c3)).
[resolve(45371,c,3636,b),merge(c),merge(d)].

45825 meets(c1,c2) | -meets(f2(c1,c4,c2,c3),c3). [resolve(45433,b,183,a)].

46666 meets(c1,c2) | meets(c1,f2(c1,f2(c1,c4,c2,c5),c5,c3)).
[resolve(45223,c,45825,b),merge(c)].

46678 meets(c1,c2) | meets(c2,c3). [resolve(46666,b,45202,c),merge(b)].

46837 meets(c1,c2) | meets(f2(c1,c4,c2,c3),c3).
[resolve(46678,b,3622,b),merge(b)].

47069 meets(c1,c2). [resolve(46837,b,45825,b),merge(b)].

47119 meets(c2,c3) | meets(c1,f2(c1,c2,c5,c3)). [resolve(47069,a,13093,b)].

47120 meets(c2,c3) | meets(f2(c1,c2,c5,c3),c3). [resolve(47069,a,12671,b)].

47303 -meets(c2,c3). [resolve(47069,a,183,a)].
47508 meets(f2(c1,c2,c5,c3),c3). [back_unit_del(47120),unit_del(a,47303)].
47509 meets(c1,f2(c1,c2,c5,c3)). [back_unit_del(47119),unit_del(a,47303)].
48883 $F. [resolve(47508,a,183,b),unit_del(a,47509)].

============================== end of proof ==========================

=================================================================

% ------- Comments from original proof -------
% Proof 1 at 0.01 (+ 0.06) seconds.
% Length of proof is 13.
% Level of proof is 4.
% Maximum clause weight is 8.
% Given clauses 26.

5 (all i all j (meets(i,j) -> ~meets(j,i))) # label(non_clause). [assumption].
16 (all x all y (precedes(x,y) <=>
   meets(x,y) | (exists z (meets(x,z) & meets(z,y)))))
   # label(non_clause). [assumption].
18 (all x ~precedes(x,x)) # label(non_clause) # label(goal). [goal].
20 ~precedes(x,y) | meets(x,y) | meets(x,f5(x,y)). [clausify(16)].
21 ~precedes(x,y) | meets(x,y) | meets(f5(x,y),y). [clausify(16)].
23 precedes(c1,c1). [deny(18)].
35 ~meets(x,y) | ~meets(y,x). [clausify(5)].
49 meets(c1,c1) | meets(c1,f5(c1,c1)). [resolve(23,a,20,a)].
50 meets(c1,c1) | meets(f5(c1,c1),c1). [resolve(23,a,21,a)].
51 ~meets(x,x). [factor(35,a,b)].
55 meets(f5(c1,c1),c1). [back_unit_del(50),unit_del(a,51)].
56 meets(c1,f5(c1,c1)). [back_unit_del(49),unit_del(a,51)].
118 $F. [resolve(55,a,35,b),unit_del(a,56)].

============================== end of proof ==========================
D.9 Theorem 4.9

\( T_{\text{interval.meeting}} \cup \Sigma_{im-ap} \not\models T_{ap} \)

================================ PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.11 (+ 0.00) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 4.000.
% Given clauses 0.

3 (all i exists j exists k (meets(j,i) & meets(i,k)))
# label(non_clause). [assumption].
10 (all x all y (precedes(x,y) <=>
meets(x,y) | (exists z (meets(x,z) & meets(z,y)))))
# label(non_clause). [assumption].
12 -(all x exists y precedes(x,y)) # label(non_clause). [assumption].
71 precedes(x,y) | -meets(x,y). [clausify(10)].
75 -precedes(c1,x). [clausify(12)].
231 meets(x,f2(x)). [clausify(3)].
257 -meets(c1,x). [resolve(75,a,71,a)].
258 $F. [resolve(257,a,231,a)].

================================ end of proof ==========================

================================ PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.12 (+ 0.00) seconds.
% Length of proof is 8.
% Level of proof is 3.
% Maximum clause weight is 4.000.
% Given clauses 0.

3 (all i exists j exists k (meets(j,i) & meets(i,k)))
# label(non_clause). [assumption].
10 (all x all y (precedes(x,y) <-> 

meets(x,y) | (exists z (meets(x,z) & meets(z,y)))))
# label(non_clause). [assumption].
12 -(all x exists y precedes(y,x)) # label(non_clause). [assumption].
71 precedes(x,y) | -meets(x,y). [clausify(10)].
75 -precedes(x,c1). [clausify(12)].
230 meets(f1(x),x). [clausify(3)].
257 -meets(x,c1). [resolve(75,a,71,a)].
258 $F. [resolve(257,a,230,a)].

======================================= end of proof ==========================

D.10 Theorem 4.7

\[ T_{ap} \cup \Sigma_{ap,im} \models T_{meets,root} \]

======================================= PROOF ==============================

% -------- Comments from original proof --------
% Proof 1 at 4.74 (+ 0.16) seconds.
% Length of proof is 45.
% Level of proof is 9.
% Maximum clause weight is 14.
% Given clauses 928.

1 (all x all y all z (precedes(x,y) & precedes(y,z) -> precedes(x,z)))
# label(non_clause). [assumption].
2 (all x all y all z (finer(x,y) & finer(y,z) -> finer(x,z)))
# label(non_clause). [assumption].
6 (all x all y (ncdf(x,y) | precedes(x,y) | precedes(y,x)))
# label(non_clause). [assumption].
7 (all x all y (finer(x,y) -> -precedes(x,y)))
# label(non_clause). [assumption].
8 (all x all y all z (finer(x,y) & precedes(y,z) -> precedes(x,z)))
# label(non_clause). [assumption].
9 (all x all y (ncdf(x,y) <-> (exists z (finer(z,x) & finer(z,y)))))
# label(non_clause). [assumption].
10 (all x all y (meets(x,y) <-> precedes(x,y) & -(exists z (precedes(x,z) & precedes(z,y)))))
# label(non_clause). [assumption].
12 (all x all y (ncdf(x,y) -> -precedes(x,y))) # label(non_clause). [assumption].
13 -(all i all j all k all l (meets(i,j) & meets(i,k) & meets(l,j) -> precedes(l,k)))
# label(non_clause). [assumption].
14 -ncdf(x,y) | finer(f1(x,y),x). [clausify(9)].
15 ncdf(x,y) | precedes(x,y) | precedes(y,x). [clausify(6)].
16 -ncdf(x,y) | finer(f1(x,y),y). [clausify(9)].
17 ncdf(x,y) | -finer(z,x) | -finer(z,y). [clausify(9)].
20 -ncdf(x,y) | -precedes(x,y). [clausify(12)].
21 -meets(x,y) | precedes(x,y). [clausify(10)].
22 -meets(x,y) | -precedes(x,z) | -precedes(z,y). [clausify(10)].
23 meets(c1,c2). [clausify(13)].
24 meets(c1,c3). [clausify(13)].
25 meets(c4,c2). [clausify(13)].
26 -precedes(x,y) | -precedes(y,z) | precedes(x,z). [clausify(1)].
27 -finer(x,y) | -finer(y,z) | finer(x,z). [clausify(7)].
28 -finer(x,y) | -precedes(x,y). [clausify(12)].
29 -finer(x,y) | -precedes(y,z) | precedes(x,z). [clausify(8)].
30 -precedes(x,y) | -precedes(y,z) | precedes(x,z). [clausify(1)].
31 -finer(x,y) | -finer(y,z) | finer(x,z). [clausify(2)].
32 -finer(x,y) | -precedes(x,y). [clausify(7)].
33 -finer(x,y) | -precedes(y,z) | precedes(x,z). [clausify(8)].
34 -precedes(x,y) | -precedes(y,z) | precedes(x,z). [clausify(1)].
35 -finer(x,y) | -finer(y,z) | finer(x,z). [clausify(2)].
36 -finer(x,y) | -precedes(y,z) | precedes(x,z). [clausify(8)].
37 -precedes(x,y) | -precedes(y,z) | precedes(x,z). [clausify(1)].
38 finer(f1(x,y),x) | precedes(x,y) | precedes(y,x). [resolve(14,a,15,a)].
39 finer(f1(x,y),y) | precedes(x,y) | precedes(y,x). [resolve(16,a,15,a)].
40 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
41 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
42 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
43 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
44 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
45 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
46 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
47 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
48 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
49 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
50 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
51 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
52 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
53 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
54 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
55 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
56 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
57 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
58 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
59 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
60 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
61 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
62 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
63 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
64 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
65 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
66 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
67 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
68 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
69 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
70 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
71 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
72 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
73 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
74 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
75 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
76 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
77 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
78 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
79 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
80 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
81 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
82 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
83 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
84 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
85 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
86 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
87 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
88 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
89 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
90 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
91 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
92 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
93 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
94 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
95 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
96 -precedes(x,y) | -precedes(y,z) | -precedes(z,y). [resolve(22,a,17,a)].
135 \textit{\textbf{precedes}}(c_3, c_2). \quad [\textsf{resolve}(59, a, 58, a)].

164 \textit{\textbf{precedes}}(x, c_1) \mid \textit{\textbf{precedes}}(x, c_3). \quad [\textsf{resolve}(59, a, 30, b)].

213 \textit{\textbf{precedes}}(c_4, x) \mid \textit{\textbf{precedes}}(x, c_4) \mid \textit{\textbf{precedes}}(f_1(c_4, x), c_2). \quad [\textsf{resolve}(64, c, 61, a)].

222 \textit{\textbf{precedes}}(c_3, c_4). \quad [\textsf{ur}(30, b, 61, a, c, 135, a)].

743 \textit{\textbf{precedes}}(f_1(c_4, c_3), c_3). \quad [\textsf{ur}(75, a, 37, a, b, 222, a)].

1832 \textit{\textbf{precedes}}(f_1(c_4, c_3), c_1). \quad [\textsf{ur}(164, b, 743, a)].

1867 \textit{\textbf{precedes}}(c_4, x) \mid \textit{\textbf{precedes}}(x, c_4) \mid \textit{\textbf{precedes}}(c_1, f_1(c_4, x)).

[\textsf{resolve}(213, c, 58, b)].

3307 \textit{\textbf{precedes}}(c_1, f_1(c_4, c_3)). \quad [\textsf{ur}(1867, a, 37, a, b, 222, a)].

3579 \textit{\textbf{finer}}(f_1(c_1, f_1(c_4, c_3)), c_3). \quad [\textsf{ur}(96, a, 59, a, c, 3307, a, d, 1832, a)].

21091 \textit{\textbf{finer}}(f_1(c_4, c_3), c_3). \quad [\textsf{ur}(78, a, 3307, a, b, 1832, a, d, 3579, a)].

21121 $F$. \quad [\textsf{resolve}(21091, a, 39, a), \textsf{unit_del}(a, 37), \textsf{unit_del}(b, 222)].

================================ end of proof =================================