Reasoning About Changes

Carmen S. Chui

April 2011

A thesis submitted in partial fulfillment of the requirements
for the degree of

BACHELOR OF APPLIED SCIENCE

Supervisor: Professor Michael Grüninger

Department of Mechanical and Industrial Engineering
Faculty of Applied Science and Engineering
University of Toronto

I hereby declare that I am the sole author of this thesis.

I authorize the University of Toronto to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Carmen S. Chui

I further authorize the University of Toronto to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Carmen S. Chui

Abstract

In artificial intelligence, the Frame Problem arises when effects of actions or events are described using logic. It is exhaustive to describe what does and does not change when a particular kind of action is performed or event occurs. The Inertia Principle, also known as the common sense law of inertia, eliminates this problem by formalizing that a given property does not change unless there is evidence to the contrary. The purpose of this thesis project is to determine how readily automated reasoning is able to prove the Inertia Principle and two of its propositions using the Process Specification Language (PSL) ontology. The thesis outlines the discovery of a missing axiom within the PSL ontology that aids in solving the Inertia Principle. As well, the experimental results of this project indicate that there is a need to standardize the automated theorem proving environment to ensure that all results and processes involved in proof generation are captured accurately.

Acknowledgements

I would like to thank Professor Michael Grüninger for his support and guidance over the course of this thesis project.

Table of Contents

List of Tables

List of Figures

1. Introduction

With the advancement of technology, theorem provers allow researchers to evaluate the validity of their first-order ontologies. Depending on the degree of automation, theorem provers are able to provide proofs for queries and to assist researchers with correcting their ontologies where necessary.

1.1 Purpose Statement

The purpose of this project is to determine whether automated reasoning is able to prove two propositions of the Process Specification Language (PSL) Ontology and the Inertia Principle. To achieve this objective, the PSL ontology is utilized to provide a formal representation of the theorems in question and any hints or lemmas used in the experimental analysis. The main focus of the investigation is to determine how readily automated reasoning is able to assist in theorem proving, with or without the inclusion of lemmas to assist the theorem prover.

1.2 Motivation

Within the mathematical community, proofs for axioms and theories can be generated by hand and with the use of automated theorem provers. Within the ontology community, however, proofs by hand can be used to verify ontologies, but there is still some uncertainty about whether these proofs are correct. Consequently, the use of semi-automated theorem provers allows ontology designers to determine whether their ontologies are accurate and verifiable, or whether ontologies need to be further revised in order for proofs to be generated. With respect to this thesis report, the main motivation is to determine how readily semi-automated theorem provers are able to generate proofs that are similar to those done by hand for the Inertia Principle and its two propositions.

2. Objectives of the Project

The objective of this project is to apply automated reasoning to determine whether an automated theorem prover is able to generate proofs for the Inertia Principle and two of its propositions in the PSL Ontology. The approach to achieving this will be purely experimental since there currently does not exist a widely-used, standardized method for ontology theorem proving. Consequently, certain portions of the propositions and axioms may need to be modified to assist the theorem prover in the process.

3. Background Information and Literature Review

This section of the report briefly outlines the Process Specification Language Ontology, theorem proving, the Inertia Theorem, and describes other process ontologies.

3.1 The Process Specification Language (PSL) Ontology

The Process Specification Language (PSL) is designed to facilitate the correct and complete exchange of process information among manufacturing systems [1]. With the increasing use of information technology in manufacturing systems, it has been increasingly important to integrate different software applications together to ensure interoperability among them. However, these applications may use the same or different terminology to associate different semantics with the terms in the domain; this is still evident if two applications utilize the same terminology, they may associate different semantics with the terms. This clash of meaning between terms prevents seamless exchange of information among software applications [1].

Consequently, PSL was designed to "create a process representation that is common to all manufacturing applications, generic enough to be decoupled from any given application and robust enough to represent the necessary process information for any given application." [2]

PSL is meant to be a neutral, interchange language that integrates multiple process-related applications throughout the manufacturing life cycle [3]; typically, point-to-point translation programs are created to facilitate communication between applications, but with the increasing number of such programs, it has become very difficult for software developers to provide translators between different pairs of applications [1].

3.1.1 PSL's Use of First Order Logic

The PSL Ontology utilizes first-order logic for its core theory (PSL-Core) and its definitional extensions. Core theories introduce new primitive concepts, while terms introduced in a definitional extension are defined using the terminology of the core theories [4].

In first-order logic, there are two types of symbols: logical and non-logical ones. The following tables, adapted from Brachman and Levesque's *Knowledge Representation and Reasoning*, summarize and explain the differences of each [5].

Logical Symbols	
Punctuation	"(", ")", and "."
Connectives	"¬" Logical negation
	"∧" Logical conjunction
	"∨" Logical disjunction
	"∃" Existential quantifier ("there exists")
	"∀" Universal quantifier ("for all")
	"=" Logical equality
Variables	We will denote these using x, y, and z, sometimes with subscripts and superscripts

Table 1: Logical Syntax in FOL

Non-Logical Symbols	
Function Symbols	We will write in uncapitalized mixed case (e.g., bestFriend) and which will be denoted more generally using $a, b, c, f, g,$ and h, with subscripts and superscripts.
Predicate Symbols	We will write in uncapitalized mixed case (e.g., olderThan) and which will be denoted more generally using $P, Q,$ and R, with subscripts and sueprscripts.

Table 2: Non-Logical Symbols in FOL

3.1.2 PSL Ontology in Prover9 Syntax

For this project, the axioms of the PSL Ontology are written in Prover9 syntax. Prover9 is an automated theorem prover for first-order and equational logic, and is the successor of the Otter theorem prover [6]. Prover9 takes in input files that contain lists of clauses to determine whether a refutation exists by utilizing the resolution inference rule. In order to prove sentences in Prover9, one must specify the parameters of its inference procedure, the sentence to be proved (also known as the 'goal'), and the sentences that can be used in the proof.

The Prover9 syntax is similar to first-order logic:

In English	If x is the mother of y, then x is a parent of y
In first-order logic	$\forall x, y\ (mother(x, y) \rightarrow parent(x, y))$
Prover9 syntax	`-mother | parent(x,y)`

Figure 1: FOL and Prover9 Syntax

3.1.3 Basic Ontological Distinctions in PSL

Within the PSL-Core ontology, the following basic ontological distinctions are made (adapted from [7]):

- *Activities* – a repeatable pattern of behaviour that may have multiple occurrences, or may never occur.
- *Activity Occurrences* – corresponds to a concrete instantiation of a unique activity. Activity occurrences are not instances of activities, since activities are not classes within the PSL ontology.
- *Time* – each activity occurrence is associated with unique timepoints that mark the beginning and end of the occurrence. The set of timepoints is linearly ordered, forwards into the future and backwards into the past; this linear ordering is captured in the PSL Ontology with the *before* relation.
- *Objects* – those elements that are not activities, occurrences, or timepoints.
- *State and Change* – process ontologies are used to represent dynamic behaviour in the world to allow software systems to make predictions about the future and explanations

with the past. PSL-Core captures basic intuitions about state and its relationship to activities with *fluents* which are properties in the domain that can change. A fluent is changed by the occurrence of activities, and a fluent can only be changed by the occurrence of activities.

3.1.4 Occurrence Trees in PSL

As well, activities are classified in the PSL Ontology according to the kinds of constraints their occurrences satisfy. Atomic activities are thus defined with respect to constraints that ask the following two questions [4]:

- Under what conditions does an atomic activity occur?
- How do occurrences of atomic activities change fluents?

These constraints are handled in the PSL Ontology with the notion of occurrence trees ($T_{occtree}$), which are partially ordered sets of activity occurrences where, for a given set of activities, all discrete sequences of their occurrences are branches of the tree [7]. All atomic activities are contained within an occurrence tree; since the tree is discrete, each activity occurrence in the tree has a unique successor occurrence for each activity [4].

3.1.5 Discrete States in PSL

In addition, the PSL-Core theory, $T_{disc\ state}$, is used to describe and "capture the basic intuitions about states and their relationships to activities." [8] The following intuitions are captured by this theory below (from [8]):

1. State is changed by the occurrence of activities.
2. State can only be changed by the occurrence of activities.
3. State does not change during the occurrence of an activity in the occurrence tree.
4. States can constrain the legal occurrence of activities.

3.1.6 Portion of PSL Lexicon Used in this Project

For the purposes of this project, *Table 3* outlines the lexicon for a portion of the PSL Ontology that will be used.

Extension	Predicate	Description
$T_{pslcore}$	`activity(a)`	a is an activity
	`activity_occurrence(o)`	o is an activity occurrence
	`object(x)`	x is an object
	`occurrence_of(o,a)`	o is an occurrence of a
	`before(t1,t2)`	timepoint t_1 precedes timepoint t_2 on the timeline
$T_{subactivity}$	`subactivity(`a_1`,`a_2`)`	a_1 is a subactivity of a_2
	`primitive(a)`	a is a minimal element of the subactivity ordering
$T_{occtree}$	`legal(s)`	s is an element of a legal occurrence
	`initial(s)`	S is the root of an occurrence tree
	`earlier(`s_1`,`s_2`)`	s_1 precedes s_2 in an occurrence tree
$T_{disc\ state}$	`holds(f,s)`	the fluent f is true immediately after the activity occurrence s
	`prior(f,s)`	the fluent f is true immediately before the activity occurrence s
$T_{complex}$	`min_precedes(`s_1`,`s_2`,a)`	The atomic subactivity occurrence s_1 precedes the atomic subactivity s_2 in an activity tree for a
	`root(s,a)`	the atomic subactivity occurrence s is the root of an activity tree for a
T_{actocc}	`root_occ(o)`	the initial atomic subactivity occurrence of o
	`leaf_occ(s,o)`	s is the final atomic subactivity occurrence of o

Table 3: Lexicon for Core Theories in the PSL Ontology that are used in this Project

The language of $T_{disc\ state}$ is introduced in *Table 3*. The two predicates for the relationship between fluents and activity occurrences are described below [8]:

- $prior(f,s)$ - This relation captures the state of the world immediately before an activity occurrence. Precondition axioms, which specify the conditions for legal activity occurrences, will be specified using the prior relation.

- $holds(f,s)$ – This relation captures the state of the world immediately after the activity occurrence. Effect axioms, which specify the effects of activity occurrence, will be specified using the holds relation.

To capture these different states of the world, there exists an initial situation S_0 that is not corresponded with any activity occurrence. Consequently, there is a relationship between $T_{disc\ state}$ and $T_{occtree}$.

3.2 The Frame Problem

In order for the relationship between $T_{disc\ state}$ and $T_{occtree}$ to be understood, the frame problem will be introduced and explained in this section. The frame problem arises when we attempt to describe the effects of actions or events using logic [9]. In classical logic, this means that when a particular kind of action is performed or particular event occurs, we need to describe what does *and* does not change.

To demonstrate the problems that arise, suppose we have three blocks on a table (see *Figure 2*).

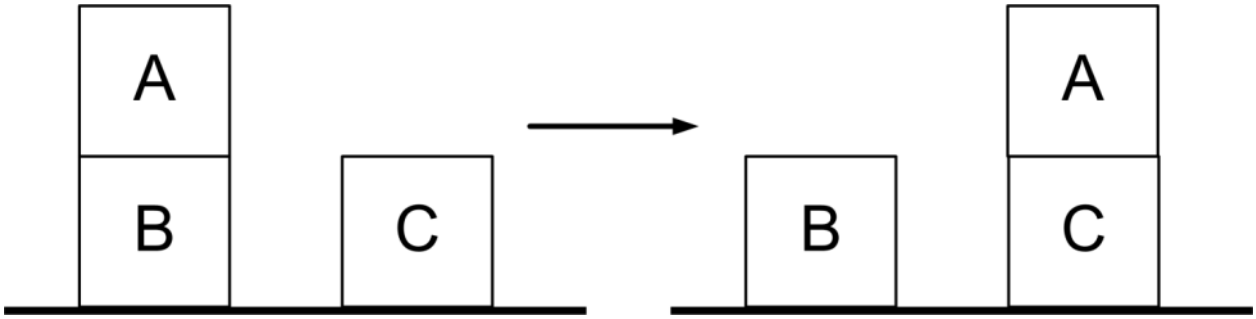


Figure 2: Blocks World Example

If we move Block A onto Block C, the effect axioms in situation calculus describe what changes were carried out (moving Block A), but does not describe what does not change. For example, because Block A is moved onto Block C, the following must also be included:

- Moving Block A does not change its colour, or any of the other blocks' colours.
- Moving Block A does not change its shape, or any of the other blocks' shapes.
- Moving Block A does not change the surface of the table.

For this list, the possibilities are endless. There are an infinite number of things that do not change after Block A is moved. These facts about what does not change appear to be common sense to us, so there is a need for a formal framework that allows us to concentrate on what changes in order to reason about these changes.

To address this problem, it can be addressed with situation calculus. The basics of situation calculus are as follows [9]:

- Situations – snapshots of the world
- Fluents – take on different values in different situations and can be thought of as time-varying properties
- Actions – changes the values of fluents

As previously mentioned, the PSL Ontology contains the $holds(f, s)$ relation which indicates that the fluent f is true in a situation s.

Before we begin to discuss the Inertia Principle, we will need to make the following distinctions about the kinds of logic that can be entailed from given facts:

- Monotonic Logic - occurs where hypotheses can allow one to derive facts by extending assumptions.
- Non-Monotonic Logic - a framework that represents defeasible inference, where reasoners are able to draw conclusions tentatively and then retract them when they come across additional information [10].

3.3 The Inertia Principle

A non-monotonic solution to the frame problem is to formalize the *common sense law of inertia* which Shanahan outlines as the following [9]:

- Inertia is normal. Change is exceptional.
- Normally, any given action (or event type) and any fluent, the action does not affect the fluent.
- Normally, given any action (or event type) and any time point, there is no instance of that action at that time point.

For the rest of this report, the law of inertia will be addressed as 'the Inertia Principle.' It states that "every change of a state is associated with an activity occurrence" [8] and is formalized in first-order logic (in Prover9 syntax) follows[1]:

```
(all f all o_1 all o_2
(((holds(f,o_1) <-> -holds(f,o_2))
& earlierEq(o_1,o_2))
->
(exists a exists o_3
(earlier(o_1,o_3) & earlier(o_3,o_2)
& (holds(f,o_3) <-> -holds(f,successor(a,o_3)))))))).
```

Figure 3: The Inertia Principle (in Prover9 Syntax)

Two propositions can be made from the Inertia Principle:

1. "For any fluent that has not changed, if it is true prior, then it holds true afterward."

   ```
   all f all s (-changes(s,f) <-> (prior(f,s) <-> holds(f,s))).
   ```

   Figure 4: Proposition #01 of the Inertia Principle

2. "For every fluent, if it *possibly* changes, it is possibly true or possibly false."

   ```
   all f exists s (changes(s,f) <-> exists s_1 (prior(f,s_1) &
   exists s_2 (-prior(f,s_2)))).
   ```

   Figure 5: Proposition #02 of the Inertia Principle

[1] Several variations of the Inertia Principle were proposed in the research and can be found in *Appendix A: Variations of the Inertia Principle*. Some of these proposed variations were used in the experiments, but they were used at a time where it was not realized axioms were missing in the PSL Ontology.

The *second* proposition can be further subdivided into three cases which are outlined and described below:

Proposition #02 Case	Description
1	If a fluent changes, s_1 is prior to s_2. `all f all s_1 all s_2 ((state(f) & prior(f,s_1) & arboreal(s_2) & -prior(f,s_2) & earlier(s_1,s_2) & -earlier(s_2,s_1)) -> (exists s changes(s,f) & arboreal(s))).`
2	If a fluent changes, s_2 is prior to s_1. `all f all s_1 all s_2 ((state(f) & prior(f,s_1) & arboreal(s_2) & -prior(f,s_2) & earlier(s_2,s_1) & -earlier(s_1,s_2)) -> (exists s changes(s,f) & arboreal(s))).`
3	If a fluent changes, s_1 and s_2 are not related to each other and are each located in different branches of the timeline tree. `all f all s_1 all s_2 ((state(f) & prior(f,s_1) & arboreal(s_2) & -prior(f,s_2) & -earlier(s_1,s_2) & -earlier(s_2,s_1)) -> (exists s changes(s,f) & arboreal(s))).`

Table 4: The Three Cases of Proposition #02

3.4 Other Process Ontologies

Since PSL utilized first-order logic in allowing elements to be mapped to mathematical structures, it allow for the following advantages (as mentioned in [11]):

- The use of mathematical structures validates that intuitions are formalized on a commonly agreed upon basis
- The use of first-order logic allows for automated reasoning with theorem provers
- First-order logic provides a framework for specifying semantic mappings between different software applications

As well, the PSL methodology can be compared to other approaches [11], which are listed in the table below.

Ontology	Description	Comparison to PSL
Petri Nets	• Designed to model the synchronization of concurrent processes	• No standard semantics • Advanced knowledge of complex mathematics required for understanding
Planning Domain Definition Language (PDDL)	• Spans the domain of planning, including a specification of states, the set of possible activities, the structure of complex activities, and the effects of activities	• Does not provide the first-order logic expressions of the items mapped to mathematical structures
GOLog	• Provides mechanisms for specifying complex activities as programs in a second order language that extends the axiomatization of situation calculus	• Describes its mathematical structures informally • Does not prove equivalence to its axiomatization
Workflow Process Definition Language (WPDL)	• A standard terminology which serves as a common framework for different workflow management systems	• Does not provide a mapping to mathematical structures • Does not provide a semantic domain
DARPA Agent Markup Language (DAML-S)	• Provides a set of process classes that can be specialized to describe a variety of Web services	• Does not provide a mapping to mathematical structures
Unified Modeling Language (UML)	• Describes how objects change over time using a series of object snapshots • Modeled as constraints on object snapshots before and after execution of the behaviour	• Does not validate semantic domains with mathematical structures • Does not axiomatize domains in first-order logic

Table 5: Alternative Approaches to Using PSL

3.5 Theorem Proving

In addition to the aforementioned propositions and axioms, the issue of theorem proving in the ontology lifecycle will be addressed in this project. Katsumi and Grüninger have proposed a methodology for the verification of first-order ontologies and have provided a lifecycle in which it may be implemented to develop a correct ontology. This lifecycle serves as a structured framework that provides guidance during the ontology development process and allows for

existing techniques to be applied throughout. The ontology lifecycle consists of five phases (see *Table 6,* adapted from [12]):

Phase	Description
Requirements	• Produces a specification of the intended models of the ontology arising from a set of use cases
Design	• Produces an ontology to axiomatize the class of models that capture the requirements • Intended models must initially be specified informally, until the design of the ontology has matured such that its requirements may be specified using its vocabulary
Verification	• Guarantees that the intended models of the ontology specified in the Requirements Phase are equivalent to the models produced in the Design Phase
Tuning	• Addresses the pragmatic issues of dealing with cases in which the theorem provers used in the Verification phase fail to retrieve a definitive answer
Application	• Covers the different ways in which the ontology is used

Table 6: The Five Phases of the Ontology Lifecycle

For the purposes of this project, the use of semi-automated reasoning with theorem provers in the Verification and Tuning Phases will assist with the goal of proving the two propositions and the Inertia Principle. Katsumi and Grüninger propose three different outcomes of verification that may arise during the Verification Phase [12]:

	Outcomes of Verification	Description
1	**Unintended Proof Found**	• A proof was found out of a sentence that contradicts the intended semantics of the ontology. • Usually occurs when the theorem prover finds a proof without using all, or any clauses of the proposition or query. • To remedy this situation, two options can be taken: o Revisit to the Design Phase and examining the proof for an identification of an axiom in the ontology that is not entailed by the intended models o Revisit the Requirements Phase to determine any errors that were made in the definition of the requirements
2	**No Proof Found**	• A proof is not found because there may be a mistake in the definition of the requirement, or there may be an error in the axiomatization of the ontology. • To remedy this situation, two options can be taken:

	Outcomes of Verification	Description
		○ Revisit the Requirements Phase or Design Phase if there is an error in the requirements or the design of the ontology, respectively. ○ If the requirements and design of the ontology are correct, it may just be that the theorem prover is preventing a proof from being found. Proceed to the Tuning Phase and attempt to adjust the ontology so that theorem prover performance is improved to produce a solution.
3	**All Requirements Met**	• If a proof is obtained, indicating that a requirement has been satisfied and is consistent with the semantics of the ontology, then the ontology satisfies its predefined requirements.

Table 7: Outcomes of Verification

Encounters with these three cases in this project will be discussed in the Methodology section. One should note, however, that these cases often lead to revisions of the design or the requirements of the ontology. When corrections are made to the design, all test results and proofs related to the error should be rerun, since errors have the potential to positively or negatively impact the results of any tests run prior to its identification and correction [12].

In addition to revising and correcting any errors that arise during the Verification Phase, the Tuning Phase focuses on the mitigation of theorem prover intractability [12]. Automated theorem provers, such as Prover9 and Otter, can have difficulty finding a proof, even if one exists (calculated manually by hand). The purpose of this phase of the ontology lifecycle is to apply techniques to assist and aid the theorem prover to generate a proof. This can be done in two ways (adapted from [12]):

- *Remove axioms of the ontology that are not relevant to the particular reasoning problem.* By reducing the number of clauses that are to be considered by the theorem prover, it decreases the time it takes for the theorem prover to find a solution since it will not consider axioms that would not be used for the proof. This increases the overall efficiency of the theorem prover's ability to find a solution.

- *Introduce lemmas into the ontology to improve theorem prover performance.*

In the mathematical sense, lemmas are short theorems that used to prove larger theorems [13]. Lemmas can be used, in conjunction or independent of subsets, to improve the theorem prover's ability to finding a solution by reducing the number of steps required. In addition, the use of lemmas also 'guides' the theorem prover to the correct solution.

In the next section, we will discuss the different combination of techniques, as described above, used to assist the theorem prover to generate proofs for the following:

1. Proposition #01
2. Proposition #02
    a. Case #01
    b. Case #02
    c. Case 03
3. The Inertia Principle

4. Methodology

The following section provides an overview of the steps taken to accomplish the objectives of the project. The methodology includes identifying the ontology files required to prove the Inertia Principle and its two propositions, identifying whether modifications to the input files are necessary, and introducing any additional lemmas to assist with the process. Where applicable, the theorem proving cases are indicated at every stage of the process.

4.1 Overview of the Methodology

For each of the proposition and parts of the Inertia Principle, a diagram and the experiment summary will be listed. The diagram represents the paths taken to assist the theorem prover into finding a solution, while the experiment summary outlines the steps displayed in the experiment (see below for an example).

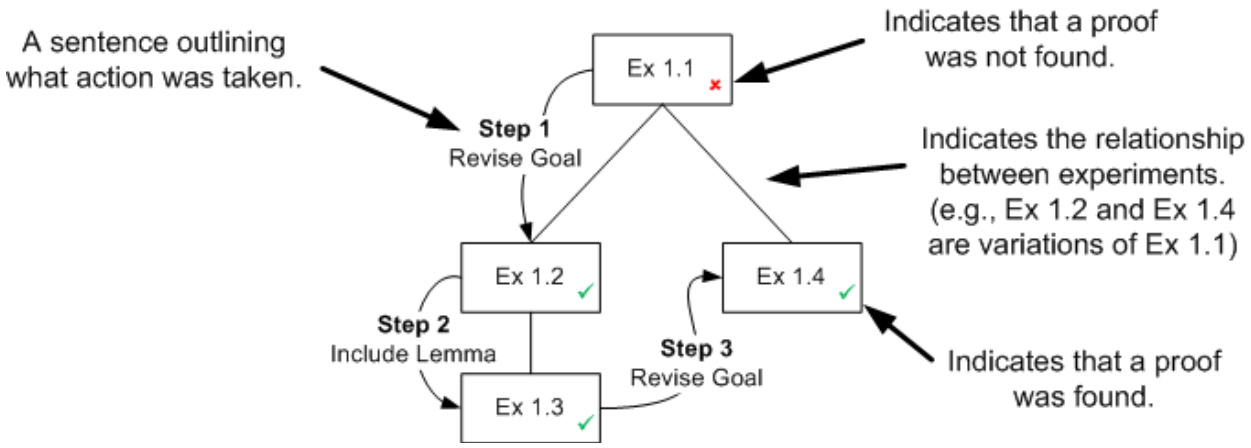


Figure 6: Explanation of Graphical Depiction of Experiments

In addition, Katsumi and Grüninger's outcomes of verification (from *Table 7*) are used to categorize each part of experiment to allow the reader to understand the reasons for the steps taken to find a solution. As well, two different symbols are utilized in the following sections to indicate whether the theorem prover was able to generate a proof or not:

Symbol	Description
✓	Proof Generated
×	Proof Not Found or Prover9 Timed Out

Figure 7: Legend of Symbols Used in Methodology Section

In summary, *Figure 8* shows the general path of the methodology used to solve the Inertia Principle and propositions. The first two propositions were solved first to ensure that the theorem prover was able to generate proofs. Then, the Inertia Principle was put through the theorem prover to determine if any proofs could be generated.

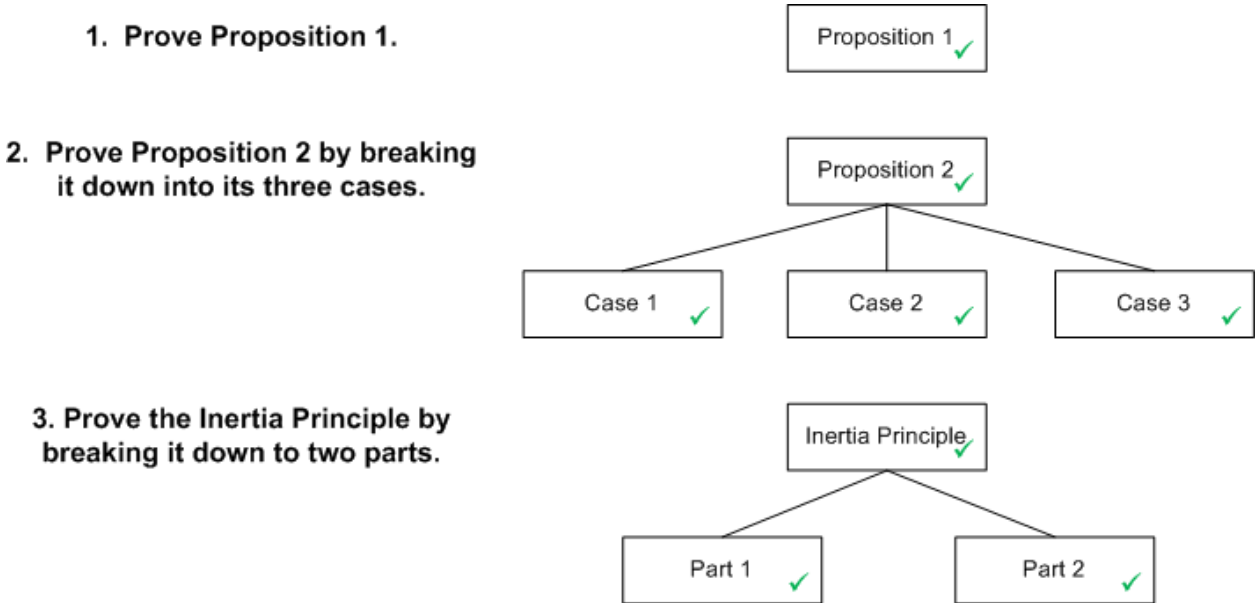


Figure 8: Methodology Overview

All of the Prover9 input files can be found in the accompanying zip file

4.2 Testing Environment

The following environment was utilized to perform the experiments:

- Operating System: Windows 7 (64-bit)
- Hardware Specifications: 2.7 GHz Intel i7 processor,1 TB hard drive, 8 GB RAM
- Software: Prover9-Mace4 (version 0.50)

All experiments were run in Prover9 with a time-out period of 60 seconds.

4.3 Proving Proposition #01

As indicated in *Figure 4*, Proposition #01 is one of the two propositions that are derived from the Inertia Principle. To begin with solving this proposition, the entire PSL Ontology was loaded into Prover9 (see the *Proposition 01* folder in the accompanying zip folder for the input files).

The experiments related to Proposition #01 are summarized in *Table 8* and depicted graphically in *Figure 9*.

Experiment	Description / Change Taken	Proof? / Next Step?	Outcome of Verification
4.3.1	Original Version of Proposition #01	✓ Revise Proposition #01 with tighter constraints	All Requirements Met
4.3.2	Modified Proposition #01 to include state and arboreal constraints	✓	All Requirements Met

Table 8: Summary of Proposition #01 Experiments

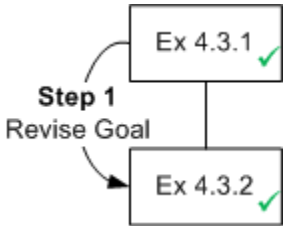


Figure 9: Graphical Depiction of Proof Generation for Proposition #01

The following step(s) correspond to the graphical depiction of proof generation for Proposition #01:

1. A revision was made to Proposition #01 to ensure that the concept of `changes` was stronger. The following additional conditions were added to ensure that any fluent that changes is a state and is arboreal.

The original axiom for Proposition #01 is:

```
all f all s (-changes(s,f) <-> (prior(f,s) <-> holds(f,s))).
```

The revised axiom is as follows (the bolded text indicates the revisions that were made):

```
all f all s (state(f) & arboreal(s) -> (-changes(s,f) <->
                (prior(f,s) <-> holds(f,s)))).
```

As can be seen in *Table 8*, no issues were encountered when using the PSL Ontology to solve Proposition #01, so the next phase is to proceed with proving Proposition #02.

4.4 Proving Proposition #02

As indicated in *Figure 5*, Proposition #02 is one of the two propositions that are derived from the Inertia Principle. To begin with solving this proposition, the entire PSL Ontology was loaded into Prover9 (see the *Proposition 02* folder in the accompanying zip folder for the input files).

4.4.1. Cases #01 and #02

Since Cases #01 and #02 are similar in nature, they will be discussed together in this section. Recall that Cases #01 and #02 are defined as follows:

Case #01: If a fluent changes, s_1 is prior to s_2.

```
all f all s_1 all s_2 ((state(f) & prior(f,s_1) & arboreal(s_2) &
-prior(f,s_2) & earlier(s_1,s_2) & -earlier(s_2,s_1)) -> (exists
s changes(s,f) & arboreal(s))).
```

Case #02: If a fluent changes, s_2 is prior to s_1.

```
all f all s_1 all s_2 ((state(f) & prior(f,s_1) & arboreal(s_2) &
-prior(f,s_2) & earlier(s_2,s_1) & -earlier(s_1,s_2)) -> (exists
s changes(s,f) & arboreal(s))).
```

In Experiment 4.4.1, the Inertia Principle's iff statement was divided into two parts in the background ontology to assist the theorem prover in finding a solution. The iff statement may cause problems for the theorem prover because of it may not be able to resolve clauses using the bi-conditional connective. By dividing the Inertia Principle into two parts, each with one implication connective, the theorem prover will be able to handle the axioms better. The two corresponding parts of the Inertia Principle are shown below:

```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1)
& arboreal(o_2)
& prior(f,o_1)
& -prior(f,o_2)
& earlier(o_1,o_2))
    ->
    (exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2)
& prior(f,o_3)
& -holds(f,o_3))))).
```

```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1)
& arboreal(o_2)
& -prior(f,o_1)
& prior(f,o_2)
& earlier(o_1,o_2))
    ->
    (exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2)
& -prior(f,o_3)
& holds(f,o_3))))).
```

Figure 10: The Inertia Principle Divided into Two Parts

For the remainder of the experiments, the Inertia Principle will be split into two parts in the background ontology input files. All experiments related to Cases 01 and 02 of Proposition #02 are summarized in *Table 9* and depicted graphically in *Figure 11*.

Experiment	Description / Change Taken	Proof? / Next Step?	Outcome of Verification
4.4.1	Inertia Theorem is broken up into 2 parts	✗ Modify the goal	No Proof Found
4.4.2	earlierEq(s_1,s) & earlierEq(s,s_2) are added to the second half of the goal	✓	All Requirements Met
4.4.3	Added goal from Ex 4.4.2 as a lemma Goal is now to prove goal from Ex 4.4.1	✓	All Requirements Met
4.4.4	Goal now contains a iff statement and is further revised	✓	All Requirements Met
4.4.5	Added goal from Ex 4.4.4 as a lemma	✓	All Requirements Met

Experiment	Description / Change Taken	Proof? / Next Step?	Outcome of Verification
4.4.6	Same goal as Ex 4.4.5 Lemma from Ex 4.4.5 removed	✓ Case #01 has been proved	All Requirements Met
4.4.7	Goal is revised to see if Case #02 can be proved	✓ Case #02 has been proved	All Requirements Met

Table 9: Summary of Proposition #02 – Case #01 and Case #02 Experiments

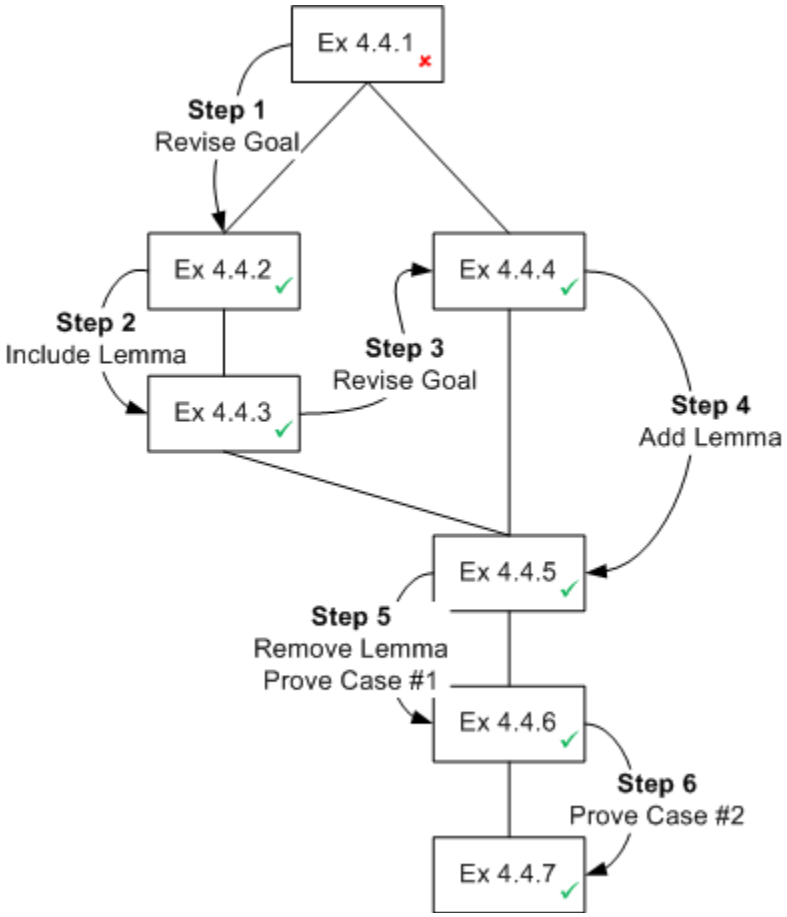


Figure 11: Graphical Depiction of Proof Generation for Proposition #02 – Cases #01 and #02

The following step(s) correspond to the graphical depiction of proof generation for the first two cases of Proposition #02.

1. In addition to breaking up the Inertia Theorem into two parts in the background ontology, the goal of Ex 4.4.2 was revised to include `earlierEq` in the consequent. This makes the goal stronger than before since s_1 can be earlier or equal to s and s can be earlier or equal to s_2, instead of simply stating that s_1 is earlier than s_2.

While this goal change is provable, it merely states that there exists an event s that occurs between s_1 and s_2, that changes, and it is possible that s_1 is equivalent to s_2. This is not the same as the definition of Case 1 – "if a fluent changes, s_1 is *prior* to s_2."

```
-(all f all s_1 all s_2
     ((state(f)
     & prior(f,s_1)
     & arboreal(s_2)
     & -prior(f,s_2)
     & earlier(s_1,s_2))
     ->
     (exists s
          changes(s,f)        &
arboreal(s)))).
```

Figure 12: Goal Used in Ex 4.4.1

```
-(all f all s_1 all s_2
     ((state(f)
     & prior(f,s_1)
     & arboreal(s_2)
     & -prior(f,s_2)
     & earlier(s_1,s_2))
     ->
     (exists s
          (earlierEq(s_1,s)
          & earlierEq(s,s_2)
          &     changes(s,f)     &
arboreal(s))))).
```

Figure 13: Goal Used in Ex 4.4.2

2. To ensure that the definition of Case 1 can be proved, the goal from Ex 4.4.1 was retained in Ex 4.4.3 and the goal from Ex 4.4.2 was introduced in the background ontology as a lemma. The goal from Ex 4.4.2 was introduced as a lemma since it was a stronger form of the goal in Ex 4.4.1.

3. Since Ex 4.4.3 was provable, it was also determined that the definition of *prior* should be reinforced in the goal. If a fluent f is prior to s_1, then it must also be that f is not prior to s_2. This is reflected in the addition of an iff statement in the goal of Ex 4.4.4, as follows:

```
-(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & (prior(f,s_1) <-> -prior(f,s_2))
     & earlier(s_1,s_2))
     ->
     (exists s
          (earlierEq(s_1,s)
          & earlierEq(s,s_2)
          & changes(s,f) & arboreal(s))))).
```

Figure 14: Goal Used in Ex 4.4.4

4. The goal from Ex 4.4.4 was then added as a lemma into the background ontology. The goal in Ex 4.4.5 is based on Ex 4.4.4, except that `earlierEq(s_1,s)` and `earlierEq(s,s_2)` were removed in the consequent, as follows:

```
-(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
```

```
& arboreal(s_2)
& (prior(f,s_1) <-> -prior(f,s_2))
& earlier(s_1,s_2))
->
(exists s
     (changes(s,f) & arboreal(s))))).
```

Figure 15: Goal Used in Ex 4.4.5

The removal of `earlierEq(s_1,s)` and `earlierEq(s,s_2)` indicate that f is prior to s_1 and not prior to s_2.

5. In Ex 4.4.6, the lemma from Ex 4.4.5 was removed, but its goal retained to determine whether the theorem prover was able to generate a proof for Case #01. After the revisions from Ex 4.4.2 to 4.4.5 were made, the theorem prover was able to generate a proof for Case #01 of Proposition #02.

6. Since Case 2 is similar to Case 1, a simple change in the goal is needed to prove Case 2, as shown below:

```
-(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & (prior(f,s_1) <-> -prior(f,s_2))
     & earlier(s_2,s_1))
     ->
     (exists s
          (changes(s,f) & arboreal(s))))).
```

Figure 16: Goal Used in Ex 4.4.7

Now that Cases #01 and #02 have been proved, the next step is to proceed to proving Case #03.

4.4.2. Case #03

Since the definition of Case #03 is different from the first two cases, it will be discussed in its own section. Recall that the definition of Case #03 is, "If a fluent changes, s_1 and s_2 are not related to each other and are each located in different branches of the timeline tree." This indicates that s_1 and s_2 are not equivalent to each other, nor are they earlier in each case. As such, Case #03 is represented in Prover9 syntax as follows, where the bolded text outlines the differences between Cases 01 and 02:

```
(all f all s_1 all s_2
((state(f)
& arboreal(s_1)
& arboreal(s_2)
& (prior(f,s_1) <-> -prior(f,s_2))
& -earlier(s_1,s_2)
& -earlier(s_2,s_1))
->
(exists s
(changes(s,f) & arboreal(s))))).
```

Figure 17: Definition of Case #03 (in Prover9 syntax)

All experiments related to Case #03 of Proposition #02 are summarized in *Table 10* and depicted graphically in *Figure 18*.

Experiment	Description / Change Taken	Proof? / Next Step?	Outcome of Verification
4.4.8	Add lemma that describes arboreal activities	×	No Proof Found
4.4.9	Removed lemma from Ex 4.4.9	×	No Proof Found
4.4.10	Added lemma from Ex 4.4.9 Modified goal by removing iff statement	×	No Proof Found
4.4.11	Modified goal by adding iff statement Goal from Ex 4.4.10 added as a lemma	×	No Proof Found
4.4.12	Modified goal by removing iff statement	×	No Proof Found
4.4.13	Modified goal by introducing a third state s_3 to indicate f is prior	✓	All Requirements Met
4.4.14	Modified goal by introducing a third state s_3 to indicate f is not prior	×	No Proof Found
4.4.15	Case 03(a) – f is prior Divided Case 03 into two parts (a) and (b)	✓	All Requirements Met

Experiment	Description / Change Taken	Proof? / Next Step?	Outcome of Verification
	Modified goal by using `earlier` instead of `earlierEq`		
4.4.16	Case 03(b) – f is not prior Divided Case 03 into two parts (a) and (b) Modified goal by using `earlier` instead of `earlierEq`	✓	All Requirements Met
4.4.17	Modified goal by testing the definitions of $earlier$ and $earlierEq$ to eliminate s_3	✓	All Requirements Met
4.4.18	Goal from Ex 4.4.17 is included as a lemma Goal is from Ex 4.4.14	✗	No Proof Found
4.4.19	Goal from Ex 4.4.18 is included as a lemma Added both Case 03a and 03b as the goal	✓ Case 03a and 03b proved	All Requirements Met
4.4.20	Goal is original definition of Case #03 in Prover9 syntax	✗ Unable to proveCase #03 as a whole	No Proof Found

Table 10: Summary of Proposition #02 – Case #03 Experiments

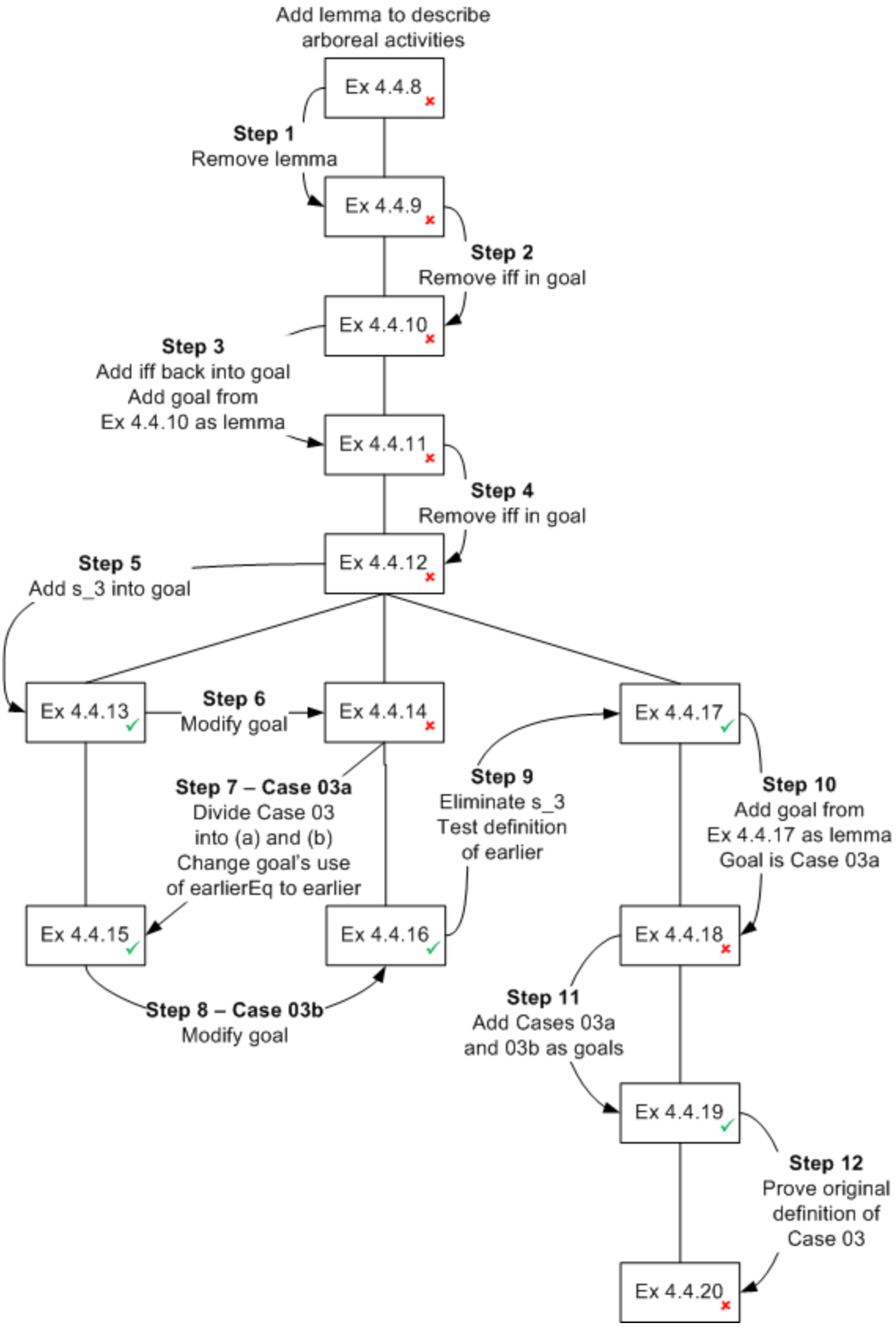


Figure 18: Graphical Depiction of Proof Generation for Proposition #02 – Case #03

A lemma was added to assist the theorem prover with proving Case #03. This lemma indicates that if an activity s_1 is arboreal, there exists an activity s_2 that is initial and is earlier or equal to s_1.

```
all s1
     (arboreal(s1)
     ->
     (exists s2
          (initial(s2)
          & earlierEq(s2,s1)))).
```

Figure 19: Lemma used in Ex 4.4.8

The following step(s) correspond to the graphical depiction of proof generation for Case #03 of Proposition #02.

1. In Ex 4.4.8, the goal could not be proved, so the lemma that was added to the background ontology was removed. No changes were made to the background ontology since it only contained a subset of the PSL Ontology with the Inertia Theorem as a lemma. The goal in Ex 4.4.9 remained the same as the goal in Ex 4.4.8.

```
-(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & (prior(f,s_1) <-> -prior(f,s_2))
     & -earlier(s_1,s_2)
     & -earlier(s_2,s_1))
     ->
     (exists s
          (changes(s,f) & arboreal(s))))).
```

Figure 20: Goal Used in Ex 4.4.8 and 4.4.9

2. Since both Ex 4.4.8 and 4.4.9 could not be proved, the iff statement in *Figure 19* was removed and the lemma in *Figure 18* was added back into the background ontology. The iff statement in the goal probably made it more difficult for the theorem prover since it timed out after 60 seconds. The modified goal is shown below:

```
-(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & prior(f,s_1) & -prior(f,s_2)
     & -earlier(s_1,s_2)
     & -earlier(s_2,s_1))
     ->
```

```
        (exists s
                (changes(s,f) & arboreal(s))))).
```

Figure 21: Goal Used in Ex 4.4.10

3. Since modifying the goal did not work, the iff statement was added back into the goal in Ex 4.4.11. The goal in Figure 19 was added as a lemma in the background ontology for Ex 4.4.11 to see if the theorem prover would be able to generate a proof with this lemma. As well, the lemma in Figure 16 above was also included in the background ontology.

4. Similarly, since the theorem prover timed out again in Ex 4.4.11, the iff statement was removed (same goal as Ex 4.4.10, *Figure 20* above) and the lemmas were retained in the input file in Ex 4.4.12.

5. Since Steps 1 to 4 did not help the theorem prover, a different approach was taken in modifying the goal. A third state, s_3, is included in the goal statement to indicate that s_3 is the initial state that is earlier than, or equal to, s_1. The fluent f is holds true if it is prior to s_3. The introduction of this third state further strengthens the fact that the fluent is true if there is some point where it is initial.

```
-(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlierEq(s_3,s_1)
      & prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

Figure 22: Goal Used in Ex 4.4.13

6. Since adding the third state s_3 allowed the theorem prover to generate a proof, the opposite of Ex 4.4.14 was tested to verify that f does not hold if it is not prior. The goal was changed as follows:

```
-(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlierEq(s_3,s_1)
      & -prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
```

```
                (changes(s,f) & arboreal(s))))).
```

Figure 23: Goal Used in Ex 4.4.14

7. The theorem prover was unable to generate a proof for Ex 4.4.14, so Case 3 was divided into two parts:
   a. The fluent f is prior to state s_3
   b. The fluent f is not prior to state s_3

   Now that Case #03 is divided into two parts, the goal of Ex 4.4.13 was modified by changing `earlierEq(s_3,s_1)` to `earlier (s_3,s_1)` (now referred to as "Case #03a"). The change makes the proposition stronger since it reinforces the point that s_3 must be earlier than s_1.

```
-(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlier(s_3,s_1)
      & -prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

Figure 24: Goal Used in Ex 4.4.15 (Case 03a)

8. Since Ex 4.4.15 generated a proof, half of Case #03 has been proved. A similar change from `earlierEq(s_3,s_1)` to `earlier (s_3,s_1)` was made in the goal of Ex 4.4.14. This change is reflected in the goal of Ex 4.4.16 below:

```
-(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlier(s_3,s_1)
      & prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

Figure 25: Goal Used in Ex 4.4.16 (Case 03b)

9. The addition of s_3 to prove both cases of Case #03 was tedious, so alternative forms of expressing Case #03 were explored. One of these ways was to remove s_3 from the goal

and have it substituted with the definition of *earlier* and *earlierEq*. The purpose of this experiment is to test the definition of *earlierEq*. Since *earlierEq* is stronger than *earlier*, this experiment tests whether *earlier* can be implied from the proposition.

The background ontology and lemmas used in Ex 4.4.16 were retained. The goal was modified as follows:

```
-(all s_1 all s_2 all f
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & earlierEq(s_1,s_2))
        ->
       earlier(s_1,s_2))).
```

Figure 26: Goal Used in Ex 4.4.17

10. Since the theorem prover was able to generate a proof for Ex 4.4.17, the goal used in Ex 4.4.17 was added in as a lemma in Ex 4.4.18. The purpose of Ex 4.4.18 is to determine whether a proof can be generated to prove Case 03a.

11. The theorem prover timed out during Ex 4.4.18, which meant that it was missing an axiom to resolve the Case 03a goal. Since Case 03b is the opposite of Case 03a, both cases were added as goals in the theorem prover in Ex 4.4.19 (shown below).

```
-(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -
prior(f,s_2)
      & initial(s_3)
      & earlier(s_3,s_1)
      & -prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) &
arboreal(s))))).
```

```
-(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -
prior(f,s_2)
      & initial(s_3)
      & earlierEq(s_3,s_1)
      & -prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) &
arboreal(s))))).
```

Figure 27: Goal Used in Ex 4.4.19

12. Since both Case 03a and Case 03b could be proved by the theorem prover, the individual parts were added as lemmas in the background ontology. The goal for Ex 4.4.20 was the original definition of Case #03 (shown below).

    However, this was unsuccessful since the theorem prover timed out. Since both parts (a) and (b) of Case #03 can be proved individually, it was decided that this was sufficient.

```
-(all f all s_1 all s_2
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

Figure 28: Goal Used in Ex 4.4.20

Since all three cases of Proposition #02 were proved, the next step was to examine the methodology used to prove the Inertia Principle.

4.5 Proving the Inertia Principle

As mentioned in the previous section, the definition of the Inertia Principle includes an *iff* statement (**<->** in Prover9):

```
(all f all o_1 all o_2
(((holds(f,o_1) <-> -holds(f,o_2))
& earlierEq(o_1,o_2))
->
(exists a exists o_3
(earlier(o_1,o_3) & earlier(o_3,o_2)
& (holds(f,o_3) <-> -holds(f,successor(a,o_3)))))))).
```

Figure 29: The Inertia Principle (in Prover9 Syntax)

Consequently, the Inertia Principle was divided into two parts and will now be explicitly referred to as Part #01 and Part #02, respectively, in this section:

```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1)
& arboreal(o_2)
& prior(f,o_1)
& -prior(f,o_2)
& earlier(o_1,o_2))
    ->
    (exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2)
& prior(f,o_3)
& -holds(f,o_3))))).
```

Figure 30: The Inertia Principle – Part #01

```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1)
& arboreal(o_2)
& -prior(f,o_1)
& prior(f,o_2)
& earlier(o_1,o_2))
    ->
    (exists o_3

(earlierEq(o_1,o_3)
& earlierEq(o_3,o_2)
& -prior(f,o_3)
& holds(f,o_3))))).
```

Figure 31: The Inertia Principle – Part #02

The methodologies utilized in proving these two parts are discussed in the subsequent sections below.

4.5.1. Proving Part #01

To begin with solving this part of the Inertia Principle, the PSL Ontology from Experiments 4.4.7 was loaded into Prover9 for the subsequent experiments (see the *Inertia Principle/Part 01* folder in the accompanying zip folder for the input files). The experiments related to Part #01 are summarized in *Table 11* and depicted graphically in *Figure 32*.

Experiment	Description / Change Taken	Proof? / Next Step?	Outcome of Verification
4.5.1.1	Added lemma in background ontology to indicate that there is some point between where it changed	×	No Proof Found
4.5.1.2	Prove the lemma that was added in ex211-1	✓	All Requirements Met
4.5.1.3	Removed initial(o) from Ex 4.5.1.2 goal to see if Prover9 can prove this stronger sentence	×	No Proof Found
4.5.1.4	Introduce an $always$ axiom Modified the definition of $holds(f, o)$ to include the new $always$ definition Goal is Part #01 of Inertia Principle	✓ Part #01 has been proved	All Requirements Met
4.5.1.5	Introduce a $never$ axiom Modified the definition of $holds(f, o)$ to include the new $always$ definition Goal is Part #02 of Inertia Principle	×	No Proof Found
4.5.1.6	Check to see if the definition of *never* works	✓	All Requirements Met
4.5.1.7	Included the goal from Ex 4.5.1.6 as a lemma to prove Part #02	× Part #02 cannot be proved. Use a different approach.	No Proof Found

Table 11: Summary of Inertia Principle (Part #01) Experiments

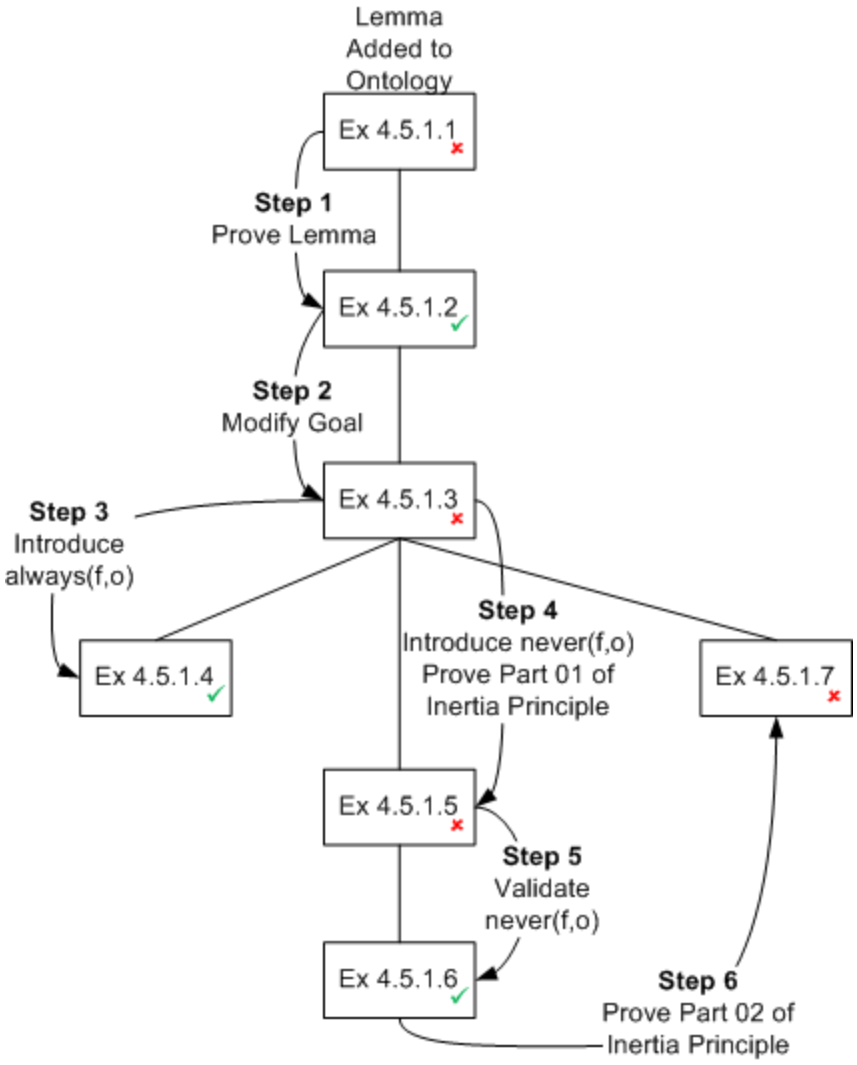


Figure 32: Graphical Depiction of Proof Generation for Inertia Principle (Part #01)

Before running the experiments, a lemma was included in the background ontology to indicate that there exists some point o_3 between o_1 and o_2 where the fluent changed and holds true.

```
(all f all o_1 all o
    ((state(f)
& arboreal(o_1)
& holds(f,o_1)
& initial(o)
& -prior(f,o)
& earlier(o,o_1))
    ->
    (exists o_3
(earlierEq(o_3,o_1)
& -prior(f,o_3)
& holds(f,o_3))))).
```

Figure 33: Lemma Used in Ex 4.5.1.1

The following step(s) correspond to the graphical depiction of proof generation for Part #01 of the Inertia Principle:

1. Since the theorem prover was unable to generate a proof for Ex 4.5.1.1, the lemma was added into the goal section of the input file to determine whether it can be inferred by the background ontology. The following version of Part #01 was used as the goal:

```
-(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1)
& arboreal(o_2)
& -prior(f,o_1)
& prior(f,o_2)
& earlier(o_1,o_2))
    ->
    (exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2)
& -prior(f,o_3)
& holds(f,o_3))))).
```

Figure 34: Goal Used in Ex 4.5.1.1

2. Since the theorem prover could prove the lemma used in Ex 4.5.1.1, the term $initial(o)$ was removed from the goal to see if the theorem prover is able to prove this stronger sentence.

```
(all f all o_1 all o
    ((state(f)
& arboreal(o_1)
& holds(f,o_1)
& initial(o)
& -prior(f,o)
& earlier(o,o_1))
    ->
    (exists o_3
(earlierEq(o_3,o_1)
& -prior(f,o_3)
& holds(f,o_3))))).
```

Figure 35: Lemma Used in Ex 4.5.1.1

```
-(all f all o_1 all o
    ((state(f)
& arboreal(o_1)
& holds(f,o_1)
& -prior(f,o)
& earlier(o,o_1))
    ->
    (exists o_3
(earlierEq(o_3,o_1)
& -prior(f,o_3)
& holds(f,o_3))))).
```

Figure 36: Goal Used in Ex 4.5.1.2

3. The theorem prover timed out during Ex 4.5.1.3, so the first part of the Inertia Principle was broken down. A new axiom, $always(f, o)$, was introduced in Ex 4.5.1.4 to assist the theorem prover in solving Part #01.

   The definition of always is as follows "a fluent f is always true in an occurrence o if and only if f is prior to o_1 such that all occurrences of o_1 are earlier than o.

```
all f all o
      (always(f,o)
      <->
      (all o1
            (earlier(o1,o)
            ->
            prior(f,o1)))).
```

Figure 37: Definition of Always

As well, the definition of the $holds(f,o)$ axiom was modified to include $always(f,o)$: "for all fluents f that hold at an occurrence o_1, f is always true at o_1, or there exists an occurrence o_2 that is earlier or equal to o_1 and f is not prior to o_2, and that all occurrences of o_2 are earlier or equal to o_1 and o_2, such that f holds true at o_3".

```
all f all o1
      (holds(f,o1)
      ->
      (always(f,o1)
      | (exists o2
            (earlierEq(o2,o1)
            & -prior(f,o2)
            & (all o3
                  (earlierEq(o2,o3)
                  & earlierEq(o3,o1)
                  -> holds(f,o3)))))))).
```

Figure 38: Modified Definition of holds(f,o) with always(f,o)

4. The theorem prover was able to generate a proof for Part #01; similarly, the axiom, $never(f,o)$, is introduced to aid the theorem prover in Ex 4.5.1.5.

   The definition of never is as follows "a fluent f is never true in an occurrence o if and only if o is arboreal and f is not prior to o_1 such that all occurrences of o_1 are earlier than o.

```
all f all o
      (never(f,o)
      <->
      (state(f)
      & arboreal(o)
      & (all o1
            (earlier(o1,o)
            ->
            -prior(f,o1))))).
```

Figure 39: Definition of never(f,o)

As well, the definition of the $holds(f,o)$ axiom was modified to include $never(f,o)$: "for all fluents f that do not hold at an occurrence o_1, f is never true at o_1, or there exists an occurrence o_2 that is earlier or equal to o_1 and f is prior to o_2, and that all occurrences of o_2 are earlier or equal to o_1 and o_2, such that f does not hold true at o_3".

```
all f all o1
      ((state(f)
      & arboreal(o1)
      & -holds(f,o1))
      ->
      (never(f,o1)
      | (exists o2
            (earlierEq(o2,o1)
            & prior(f,o2)
            & (all o3
                  (earlierEq(o2,o3)
                  & earlierEq(o3,o1)
                  -> -holds(f,o3)))))))).
```

Figure 40: Modified Definition of holds(f,o) with never(f,o)

5. Since the theorem prover was unable to generate a proof for Ex 4.5.1.5, the definition of $never(f,o)$ was used in the goal to determine whether it was valid within the PSL Ontology. All of the lemmas used in Ex 4.5.1.4 and Ex 4.5.1.5 were retained in Ex 4.5.1.6.

```
-(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1)
& arboreal(o_2)
& prior(f,o_1)
& -prior(f,o_2)
& earlier(o_1,o_2))
    ->
-never(f,o_2))).
```

Figure 41: Goal Used in Ex 4.5.1.6

6. Since the theorem prover was able to generate a proof for the definition of $never(f,o)$, it was added as a lemma to the background ontology to prove Part #02.

Since the theorem prover was unable to generate a proof for Part #02 in this set of experiments, a different approach was taken to aid the theorem prover and will be discussed in the next section.

4.5.2. Proving Part #02

To begin with solving this part of the Inertia Principle, the PSL Ontology from Experiments 4.5.1.7 was loaded into Prover9 for the subsequent experiments (see the Inertia Principle/*Part 02* folder in the accompanying zip folder for the input files). The experiments related to Part #02 are summarized in *Table 12* and depicted graphically in *Figure 42*.

Experiment	Description / Change Taken	Proof? / Next Step?	Outcome of Verification
4.5.2.1	Retained the background ontology used in Ex 4.5.1.7 Inclusion of 2 lemmas that utilize the definitions of $never(f, o)$, $arboreal(o)$, $initial(s)$, $successor(a, o)$, and $generator(a)$	×	No Proof Found
4.5.2.2	Lemma #2 is used as the goal to ensure that it is valid in the background ontology	✓	All Requirements Met
4.5.2.3	Lemma #2 is added back into the background ontology Goal tests the definitions of $successor(a, o)$ and $generator(a)$	✓	All Requirements Met
4.5.2.4	Added goal from Ex 4.5.2.3 as a lemma Modified goal to test the definition that all successor events are generator events in occurrence trees	×	No Proof Found
4.5.2.5	Background ontology contains a subset of PSL Ontology that includes occtree and core axioms Testing the definitions of the successor and generator	×	No Proof Found
4.5.2.6	Switched the antecedent and consequent of the goal used in Ex 4.5.2.5 to test the definitions of successor, generator, and initial	✓	All Requirements Met
4.5.2.7	Removed –initial(o1) from goal of Ex 4.5.2.6 Testing the definition that all successor events are occurrences of generator activities	×	No Proof Found
4.5.2.8	Testing the definition of arboreal	✓	All Requirements Met
4.5.2.9	Modified goal – similar to Ex 4.5.2.5 Testing the definition of arboreal and –initial	✓	All Requirements Met
4.5.2.10	Modified goal to test the definition of arboreal, successor, and generator	×	No Proof Found
4.5.2.11	Testing the opposite direction of goal used in Ex 4.5.2.10	✓	All Requirements Met
4.5.2.12	Goal from Ex 4.5.211 is added as a lemma Modified Part #02 of Inertia Principle as goal	✓ Part #02 has been proved	All Requirements Met

Table 12: Summary of Inertia Principle (Part #02) Experiments

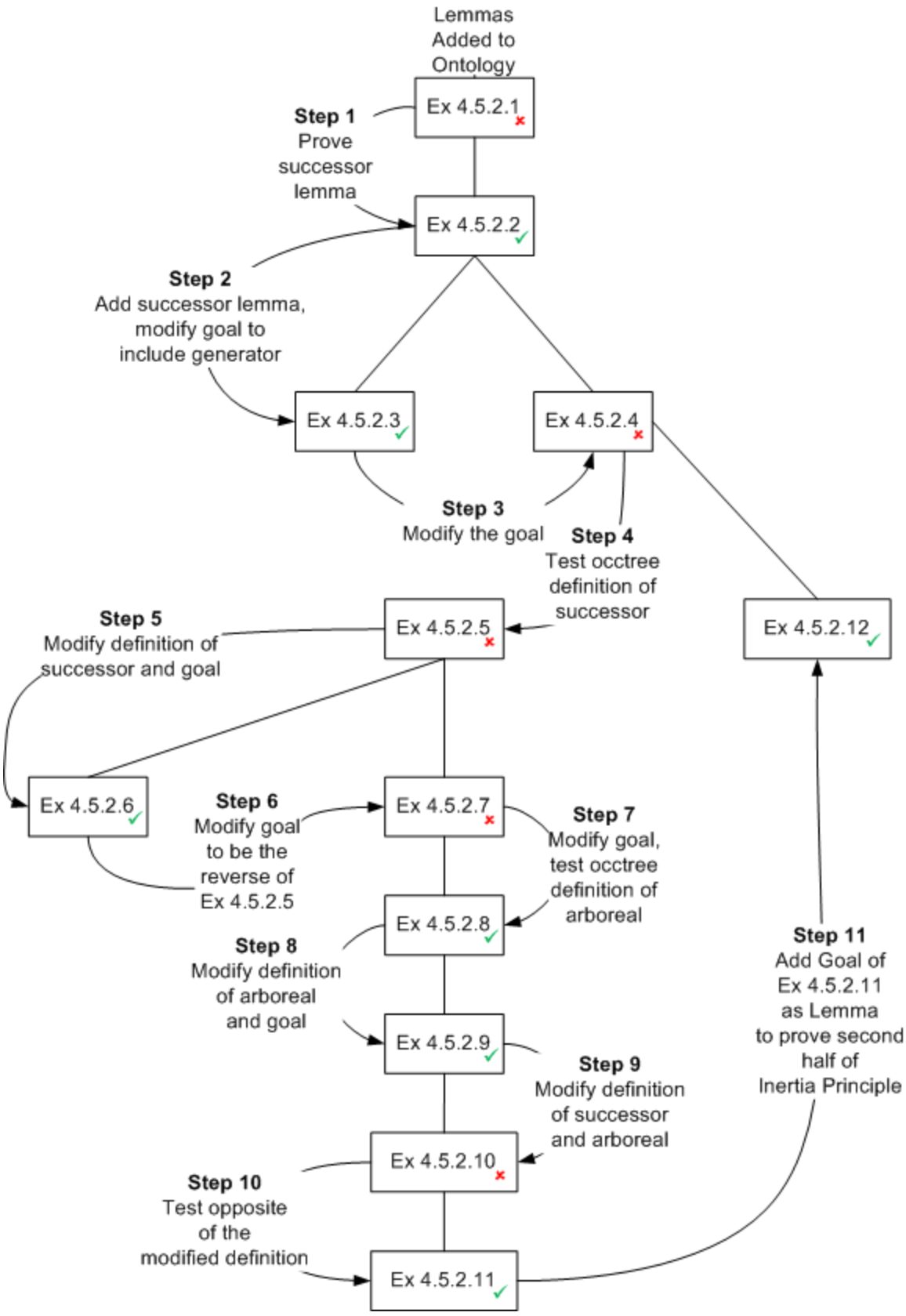


Figure 42: Graphical Depiction of Proof Generation for Inertia Principle (Part #02)

Continuing from Ex 4.5.1.7, the lemmas defining $always(f,o)$ and $never(f,o)$ and their respective modifications to the $holds(f,o)$ axiom were retained for the Part #02 experiments. To aid the theorem prover, two additional lemmas were added to the background ontology:

- A lemma defining that, if an a fluent f is prior to an occurrence o_1 and not prior to an occurrence o_2, and o_1 is earlier than o_2, then the fluent f is not never true at o_2. The 'not never' ($-never(f,o_2)$) does not mean that f is always true, but it is *possibly* true at o_2.

```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1)
& arboreal(o_2)
& prior(f,o_1)
& -prior(f,o_2)
& earlier(o_1,o_2))
    ->
-never(f,o_2))).
```

Figure 43: Lemma #1 Used in Ex 4.5.2.1

- A lemma utilizing the definitions of initial, successor, generator, and arboreal. In the PSL Ontology in the Theory of Occurrence Trees, the following definitions and axioms are made [14]:
  - $arboreal(o)$ – an activity occurrence is arboreal iff it is an element of an occurrence tree
  - $initial(s)$ – there is an initial occurrence of each activity such that no occurrence in the occurrence tree is earlier than an initial occurrence
  - $successor(a,o)$ – the successor of an arboreal activity occurrence is an occurrence of a generator activity
  - $generator(a)$ – an activity is a generator iff it has an initial occurrence in the occurrence tree

  These definitions and axioms are already defined within the background ontology:

```
all s
        (arboreal(s)
        ->
        activity_occurrence(s)).
```

Figure 44: Definition of an Arboreal Activity

```
all s
        (initial(s)
        <->
        (arboreal(s)
        & -( exists sp
                ( earlier(sp,s))))).
```

Figure 45: Definition of an Initial Activity

```
all a all o

(occurrence_of(successor(a,o),a)
        <->
        (generator(a)
        & arboreal(o))).
```

Figure 46: Definition of a Successor Activity

```
all a
        (generator(a)
        ->
        (exists s
                (initial(s)
                &
occurrence_of(s,a)))).
```

Figure 47: Definition of a Generator Activity

With these definitions, the lemma below was added to the background ontology. It states that "if an occurrence o_2 is arboreal and not initial, then there exists an activity a and an occurrence o_3, such that o_2 is equal to a being the successor of o_3."

What this implies is that for any activity o_2, that is not the initial occurrence of the occurrence tree, it is the successor of some other activity o_3 further down in the occurrence tree.

```
(all o2
        ((arboreal(o2)
        & -initial(o2))
        ->
        (exists o3 exists a
                ((o2 = successor(a,o3)))))).
```

Figure 48: Lemma #2 Used in Ex 4.5.2.1

With the inclusion of these two lemmas, the methodology used in this section can now be outlined. The following step(s) correspond to the graphical depiction of proof generation for Part #02 of the Inertia Principle:

Steps:

1. Since a proof could not be generated in Ex 4.5.2.1, Lemma #2 was used as the goal in Ex 4.5.2.2 to ensure that it is valid. The theorem prover was able to generate a proof for this lemma, so it means that it can be used to help solve Part #02 of the Inertia Principle.

2. Now that Lemma #2 has been proved to be valid, it is added back into the background ontology. The goal was modified to test the definitions of $successor(a,o)$ and $generator(a)$ as shown below. What it means is that if an occurrence o_1 is the successor event of o and the fluent f is prior to o_1, then f holds at o and a is a generator event of that occurrence tree.

```
-(all o1 all a all o all f
      (((o1 = successor(a,o))
```

```
      & prior(f,o1))
      ->
      (holds(f,o)
      & generator(a)))).
```

Figure 49: Goal Used in Ex 4.5.2.3

3. The theorem prover was able to generate a proof when testing the definitions of the occurrence tree axioms. The goal from Ex 4.5.2.3 is added into the ontology as a lemma, and the next step is to test the direct definition that all successor events are generator events in the occurrence tree.

```
-(all o1 all a all o
      (((o1 = successor(a,o))
      ->
      generator(a)))).
```

Figure 50: Goal Used in Ex 4.5.2.4

4. The theorem prover timed out during Ex 4.5.2.4, which indicated that there was something wrong within the background ontology. In Ex 4.5.2.5, a subset of the PSL Ontology containing the PSL-Core and occurrence tree axioms was then used in the background ontology to test the definition of $successor(a,o)$ and $generator(a)$.

   The following goal states that, if an occurrence o_1 is equal to a being the successor activity of o and not initial, then it indicates that a must be a generator activity in the occurrence tree.

```
-(all o1 all a all o
      (((o1 = successor(a,o))
      & arboreal(o)
      & arboreal(o1)
      & -initial(o1))
      ->
      generator(a))).
```

Figure 51: Goal Used in Ex 4.5.2.5

5. The theorem prover also timed out during Ex 4.5.2.5, which indicated that there was something wrong within the definition of the successor axiom. The antecedent and consequent in the goal of Ex 4.5.2.5 were then switched in Ex 4.5.2.6 to see if the theorem prover was able to generate a proof.

   The following goal states that, if an occurrence o_1 is equal to a being the successor activity of o and a is a generator activity, then o_1 is not the initial activity in the occurrence tree.

```
-(all o1 all a all o
      (((o1 = successor(a,o))
      & arboreal(o)
      & arboreal(o1)
```

```
      & generator(a))
      ->
       -initial(o1))).
```

Figure 52: Goal Used in Ex 4.5.2.6

6. Since the theorem prover was able to generate a proof for Ex 4.5.2.6, the term $initial(o1)$ was removed from the goal in Ex 4.5.2.7 to see if the background ontology can entail that if a is a successor to o, a must be a generator activity in the occurrence tree.

```
-(all o1 all a all o
      (((o1 = successor(a,o))
      & arboreal(o)
      & arboreal(o1))
      ->
      generator(a))).
```

Figure 53: Goal Used in Ex 4.5.2.7

7. Again, the theorem prover timed out during Ex 4.5.2.7, so it was determined that the definition of $arboreal(o)$ should be tested. The following goal indicates that if an occurrence o is arboreal, then there exists an activity a such that o is an occurrence of a generator activity a.

```
-(all o
      (arboreal(o)
      ->
      (exists a
            (occurrence_of(o,a)
            & generator(a))))).
```

Figure 54: Goal Used in Ex 4.5.2.8

8. Since a proof was generated in Ex 4.5.2.8, $-initial(o)$ was included back into the goal (the goal for Ex 4.5.2.9 is similar to Ex 4.5.2.5). The following goal indicates that if o is arboreal and not initial, then there exists an activity a and occurrence $o1$ such that o is equal to the successor activity of a and o_1 in the occurrence tree.

```
-(all o
      ((arboreal(o)
      & -initial(o))
      ->
      (exists a exists o1
            ((o = successor(a,o1)))))).
```

Figure 55: Goal Used in Ex 4.5.2.9

9. Since a proof was generated in Ex 4.5.2.9, the definitions of arboreal, successor and generator within the Theory of Occurrence Trees are valid. The goal was then revised to determine whether a successor activity a is arboreal, such that an activity a is a generator activity in the occurrence tree.

```
-(all o all a
      (arboreal(successor(a,o))
      ->
      generator(a))).
```

Figure 56: Goal Used in Ex 4.5.2.10

10. The theorem prover timed out during Ex 4.5.2.10, so the opposite direction of the goal was tested by switching the antecedent and consequent in Ex 4.5.2.11. The following goal states that if *a* is a generator activity and *o* is arboreal, then *o* is the successor of *a* and *o* must be arboreal.

```
-(all o all a
      ((generator(a)
      & arboreal(o))
      ->
      arboreal(successor(a,o)))).
```

Figure 57: Goal Used in Ex 4.5.2.11

11. Because a proof was generated in Ex 4.5.2.11, the goal was added as a lemma in the background ontology for Ex 4.5.2.12. Consequently, Part #02 of the Inertia Principle was modified to include the definition of successor in the goal of Ex 4.5.2.12:

```
-(all f all o2
     ((state(f)
     & arboreal(o2)
     & -initial(o2)
     & -prior(f,o2))
     ->
     (exists o3 exists a
           ((o2 = successor(a,o3))
           & -holds(f,o3))))).
```

Figure 58: Goal Used in Ex 4.5.2.12

The theorem prover was able to generate a proof for Ex 4.5.2.12, so this indicates that Part #02 of the Inertia Principle has been solved.

4.6 Missing Axiom in PSL

While attempting to prove the second half of the Inertia Theorem, it was found from modifying the input files that an axiom was missing in the PSL Ontology. Experiment 4.5.2.11 and 4.5.2.12

indicate that the inclusion of an axiom defining arboreal, successor, and generator events greatly assists with the solving the second half of the Inertia Principle.

With the addition of the following axiom, Prover9 is able to generate a proof for the second half of the Inertia Principle.

```
all a all o
       (arboreal(successor(a,o))
       ->
       (generator(a)
        & arboreal(o))).
```

Figure 59: Missing Axiom in the PSL Ontology (in Prover9 Syntax)

This axiom indicates that if an activity `a` in an occurrence `o` is a successor event and is arboreal, then the activity `a` is a generator event and that the occurrence `o` is arboreal. This missing axiom will need to be included in the next revision of the PSL Ontology because the Inertia Theorem cannot be proved without it.

With the inclusion of this missing axiom, all of the above experiments were re-run in Prover9 to determine if the inclusion of the axiom had any effect on the results. A summary of these revised results can be found in the next section.

5. Summary of Results

The Proofs section contains the Prover9 proofs that were generated for each of the each of the following:

- Proposition #01
- Proposition #02
    - Case #01
    - Case #02
    - Case #03
- The Inertia Principle

All of the experiment input files and their proofs can be found in the accompanying zip folder .

Table 13 below summarizes the results from the aforementioned experimental methodology used in semi-automated theorem proving. It can be seen that the inclusion of lemmas in the background ontology does aid the theorem prover in finding a solution, but no correlation can be made as to whether the lemmas hasten the process since, for each proposition, the resulting duration and length of the proofs vary.

	Part/Version	Proof Generated?	Duration (s)	Length (# lines)	Lemmas Required?
Proposition 1	Normal Version	✓	0.05	46	N/A
	Revised Version	✓	0.08	42	N/A
Proposition 2	Case 1	✓	0.20	85	✓
	Case 2	✓	0.14	79	✓
	Case 3 (a)	✓	2.25	42	✓
	Case 3 (b)	✓	17.53	70	✓
Inertia Principle	Part 01	✓	0.12	28	✓
	Part 02	✓	0.16	48	✓

Table 13: Summary of Results

Note: Duration here refers to the CPU time required to generate a result

With the inclusion of the missing axiom (*Figure 59*), all of the above experiments were re-run to determine whether the lemma affected the results. These re-run experiments are summarized in *Table 14* below and their proofs can be found in the *Proofs Generated by Prover9* section of the report:

	Part/Version	Proof Generated?	Duration (s)	Length (# lines)	Lemmas Required?
Proposition 1 + Missing Axiom	Normal Version	✓	0.12	46	N/A
	Revised Version	✓	0.08	42	N/A
Proposition 2 + Missing Axiom	Case 1	✓	0.23	85	✓
	Case 2	✓	0.28	79	✓
	Case 3 (a)	✓	1.89	42	✓
	Case 3 (b)	✓	28.97	70	✓
Inertia Principle + Missing Axiom	Part 01	✓	0.14	28	✓
	Part 02	✓	0.20	48	✓

Table 14: Summary of Results (with the inclusion of the missing axiom)
Note: Duration here refers to the CPU time required to generate a result

As can be seen in both tables, the inclusion of this missing axiom does not affect the length of time for the re-run experiments, but causes the theorem prover to take more time in generating proofs. The additional axiom only aids in the theorem proving process for goals that involve `successor` and `generator` states.

6. Difficulties Encountered

Over the course of this thesis project, the following difficulties were encountered:

- It is difficult to formalize the methodology and steps used in the theorem proving lifecycle. This is partly due to the fact that most of the steps taken were ad-hoc – changes, whether they were major or minor, to the background ontology, goals, and/or lemmas were made on-the-fly as the experiments progressed.

- Because the theorem proving methodology is ad-hoc, it is difficult and cumbersome to keep track of the changes that were made to input files. As one continues to make further revisions to the ontology, goal, or lemmas, one might forget about the original purpose of the experiment and become 'lost' in the theorem proving lifecycle. In order to determine where one is in the experiments, one will have to retrace his/her steps again, which may be a cumbersome process.

- Currently in the Prover9 environment, it is difficult to keep track of the differences between the background ontology and lemmas used, unless it is explicitly stated in a comment (shown below in the highlighted comments in *Figure 60*).

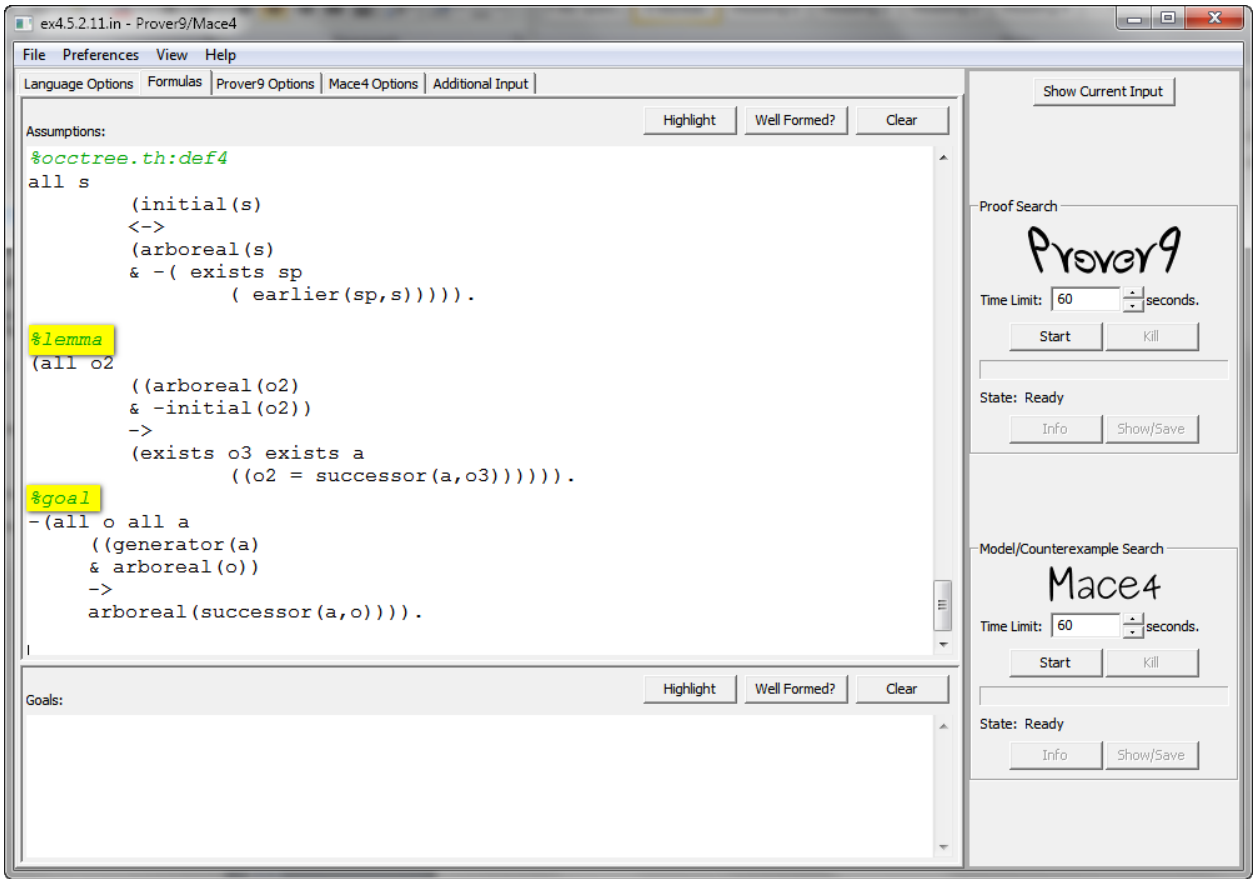


Figure 60: Distinguishing Lemmas and Background Ontology in Prover9

7. Recommendations for Theorem Proving in the Ontology Lifecycle

One of the major difficulties with this thesis project was that it was difficult to keep track of which minor or major changes were made in the Prover9 input files. As discussed in *Section 4: Methodology*, the entire process was purely experimental in nature. While most of the process seemed ad-hoc, it is preferred that there are easier ways to document what was done to the Prover9 input files. Documentation of the changes made to the input files allows reasoners to be able to trace their steps in the theorem proving process and note what changes were made to help generate the proofs.

This section proposes several recommendations that would improve the automated theorem proving environment and allow users to be able to easily recall actions or steps that were made throughout the theorem proving process.

7.1. A Spreadsheet to Keep Track of Goals, Lemmas, and Descriptions of Process

Similar to the summary tables found in the Methodology section, a spreadsheet should be utilized to tally the following information when running experiments:

- Experiment Number
- Proof Generated? (Yes/No)
- Time/Duration of Proof Generation (if any)
- Length of Proof (if any)
- Description of Experiment and Steps Taken
- Goal
- Lemmas Used (if any)

By tallying the relevant information for the experiment, it allows users to determine what was done to conduct the experiment and whether a proof was generated by the theorem prover. However, this method is cumbersome to keep track changes made to the input files since users would have to copy and paste their goals and lemmas for every experiment that is conducted.

A sample spreadsheet is displayed below:

Experiment	Proof	Time	Length	Description	Goal	Lemmas Used
Ex 2.5.1	No	N/A	N/A	Use part 1 of Inertia Theorem as goal	(all f all o_1 all o_2 ((state(f) & arboreal(o_1) & arboreal(o_2) & prior(f,o_1) & -prior(f,o_2) & earlier(o_1,o_2)) ->(exists o_3 (earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & prior(f,o_3) & -holds(f,o_3))))).	N/A

Table 15: Sample of Spreadsheet Used to Tally Experiment Details

7.2. A Graphical Outline of Different Paths for Theorem Proving

Another approach to keeping track of the different paths taken during the theorem proving process is to develop a graphical notation to outline these paths and outcomes. The concept is similar to a flowchart methodology, where every node is an experiment that results in different paths. If a proof is not generated for a certain experiment, an alternate path can be taken; such paths include minor modifications made to the contents of the input file (such as modifying the goal, adding lemmas). The paths reach an end when the theorem prover is able to prove the goal(s) in question. A sample schematic of this graphical outline is shown below.

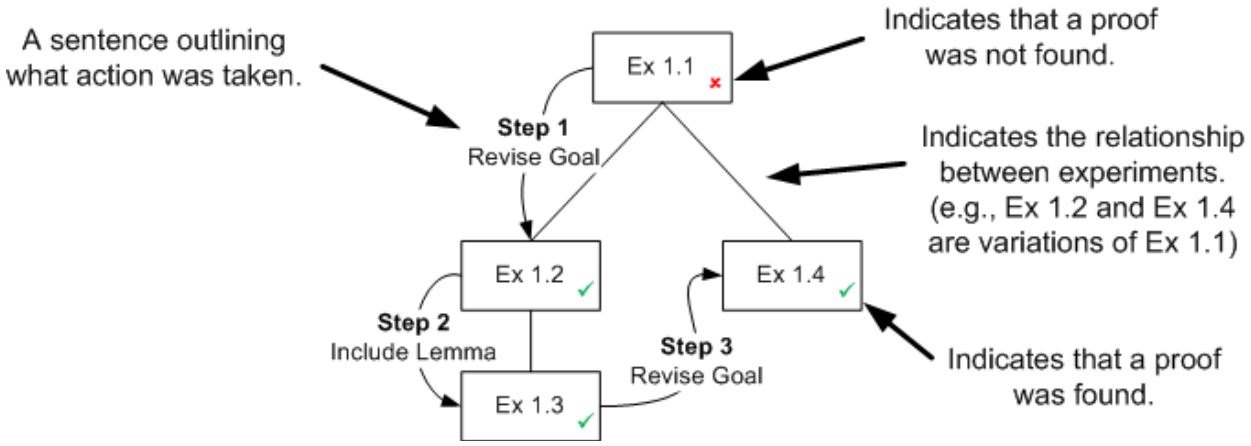


Figure 61: Sample Schematic of Proposed Graphical Outline of Paths

The advantages of having a graphical schematic to outline the paths taken in the experiment are as follows:

- It allows users to keep track of where they are in the theorem proving process
- It allows users to go back and re-trace their steps in the theorem proving process
- It is easier, from a graphical point-of-view, to determine which subset of experiments guide users to a proof in the overall experiment

7.3. A Dynamic Software Environment for Automated Theorem Proving

The final recommendation to make automated theorem proving easier is to develop a software environment that dynamically captures the changes that are made to input files in the process. Such an environment would allow users to:

- Keep track of which version of the experiment they are testing and/or modifying
- Determine which lemmas or additional pieces of information guided the theorem prover to come up with a proof for the goal in question
- Determine the number of revisions made to the original input file that allowed for a proof to be generated
- Determine what was changed and where it was changed in the input file
- Retrace/backtrack their steps in the theorem proving process

This proposed software environment should provide a graphical user interface (GUI) that allows users to determine which portion of the input file pertains to the process description, goal, and

lemma(s). Suggestions for the interface are shown in *Figure 62* below, where the layout of interface is based on Mace4-Prover9's existing interface.

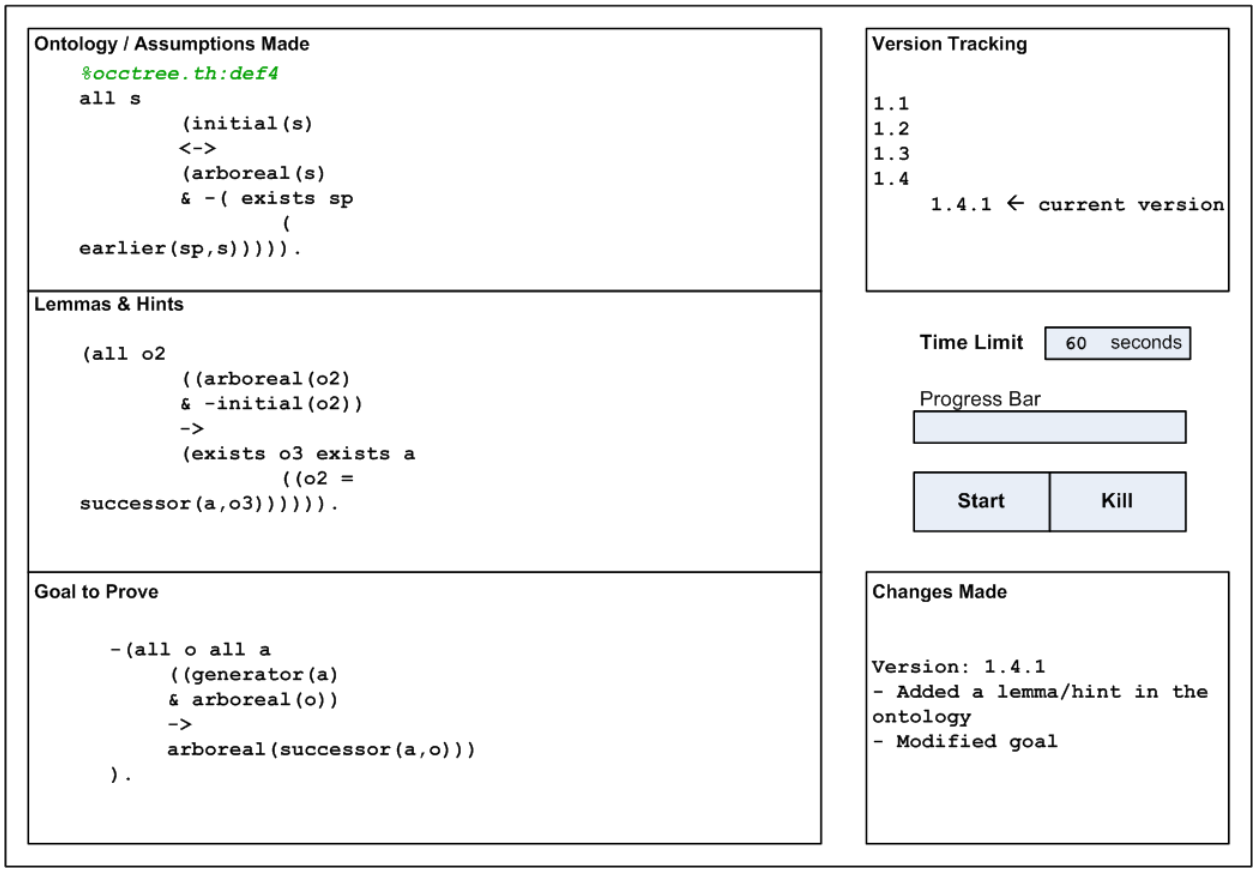


Figure 62: Suggested User Interface for Theorem Proving

This software environment should work in conjunction with other theorem provers, not necessarily Mace4-Prover9. It would add an extra, yet intermediate, layer of utilization for users who wish to use it in conjunction with other theorem provers, such as Otter or SNARK.

8. Conclusions

By generating semi-automated proofs for propositions and principles that aid in solving the Frame Problem, this thesis project has successfully tested these principles using the PSL Ontology. Applications of the Katsumi and Grüninger's ontology theorem proving lifecycle made it evident that there is a need for a more structured methodology in the lifecycle process. The results obtained from this thesis project support the need for a formal methodology to assist reasoners with automated theorem provers. Since there is no current standard, an area of future research is to determine if there are any other formalized methodologies that make the process easier and efficient. As well, this thesis project identified an axiom that is missing in the PSL Ontology and will need to be included in the next revision of the ISO standard.

9. Areas for Further Research & Future Work

Throughout the theorem proving process, there were some experiments that were not able to generate proofs. While manual proofs exist for both propositions and the Inertia Principle (see *Appendices B, C, and D*), it is interesting to determine the differences between the manual and semi-automated proofs. Further research can be conducted in this thesis project to determine why there are discrepancies between these handwritten and automated proofs, and to determine why Prover9 was unable to prove the experiments in the same way as the manual proofs. Furthermore, another area of future research would be to identify additional (or missing) axioms in the PSL Ontology to help streamline the theorem proving process for other propositions that can be derived from the Inertia Principle.

References

The following sources were referenced in the body of the report.

[1] Michael Gruninger and Christopher Menzel, "The Process Specification Language (PSL) Theory and Applications," *AI Magazine*, vol. 24, no. 3, 2003.

[2] National Institute of Standards and Technology. (2007, January) Manufacturing Systems Integration Division. [Online]. http://www.mel.nist.gov/psl/faq.html

[3] A.F. Cutting-Decelle, C.J. Anumba, A.N. Baldwin, N.M Bouchlaghem, and M. Gruninger, "Towards a unified specification of the construction process information: The PSL Approach," *Product and Process Modelling in Building and Construction*, pp. 199-207, 2000.

[4] Michael Gruninger, "Using the PSL Ontology," in *Handbook of Ontologies*, S. Staab, Ed. Berlin: Springer-Verlag, 2009, pp. 419-431.

[5] Ronald Brachman and Hector Levesque, *Knowledge Representation and Reasoning*. New York: Elsevier, Inc., 2004.

[6] William McCune. (2010) Prover9 and Mace4. [Online]. http://www.cs.unm.edu/~mccune/prover9/

[7] Michael Gruninger, "Ontology of the Process Specification Language," in *Handbook of Ontologies and Information Systems*, S. Staab, Ed. Berlin: Springer-Verlag, 2003, pp. 599-618.

[8] Michael Gruninger, "Discrete States in PSL," 2007.

[9] Murray Shanahan, *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. Cambridge, Massachusetts: MIT Press, 1997.

[10] G. Aldo Antonelli. Non-Monotonic Logic. [Online]. http://plato.stanford.edu/archives/sum2010/entries/logic-nonmonotonic/

[11] Conrad Bock and Michael Gruninger, "PSL:Asemantic domain for flowmodels," *Software System Model*, vol. 4, pp. 209-231, 2005.

[12] Megan Katsumi and Michael Gruninger, "Theorem Proving in the Ontology Lifecycle," in *Knowledge Engineering and Ontology Design 2010*, Valencia, Spain, 2010.

[13] Eric W. Weisstein. (2011, March) MathWorld - A Wolfram Web Resource. [Online]. http://mathworld.wolfram.com/Lemma.html

[14] National Institute of Standards and Technology. (2002, September) PSL - Occurrence Trees. [Online]. http://www.mel.nist.gov/psl/psl-ontology/part12/occtree.th.html

Appendices

The following section contains relevant appendices that were referenced in the body of this report.

Appendix A: Variations of the Inertia Principle

The Inertia Principle can also be rewritten in four different ways:

(i)
```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1) & arboreal(o_2)
& (holds(f,o_1) <-> -holds(f,o_2))
& earlier(o_1,o_2))
    ->
    (exists a exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2)
& (holds(f,o_3) <-> -holds(f,successor(a,o_3)))))))).
```

(ii)
```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1) & arboreal(o_2)
& (holds(f,o_1) <-> -holds(f,o_2))
& earlier(o_1,o_2))
    ->
    (exists a exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2) & changes(o_3,f))))).
```

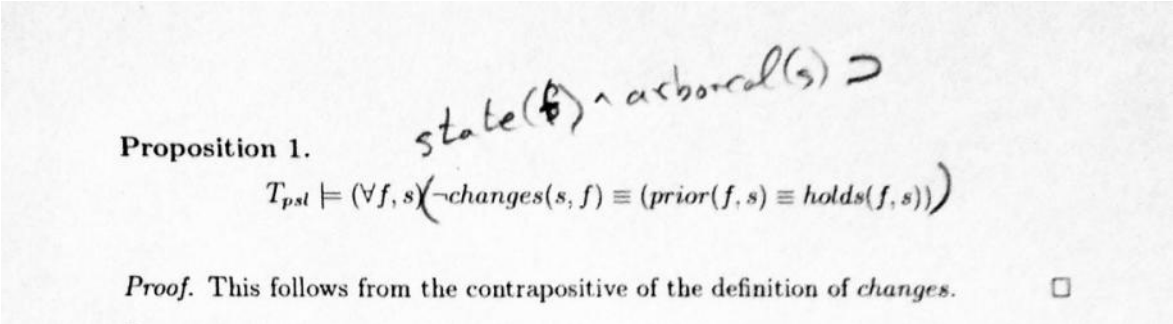
(iii)
```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1) & arboreal(o_2)
& (prior(f,o_1) <-> -prior(f,o_2))
& earlier(o_1,o_2))
    ->
    (exists a exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2) & changes(o_3,f))))).
```

(iv)
```
(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1) & arboreal(o_2)
& prior(f,o_1) & -prior(f,o_2)
& earlier(o_1,o_2))
    ->
    (exists a exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2) & falsifies(o_3,f))))).

(all f all o_1 all o_2
    ((state(f)
& arboreal(o_1) & arboreal(o_2)
& -prior(f,o_1) & prior(f,o_2)
& earlier(o_1,o_2))
    ->
    (exists a exists o_3
        (earlierEq(o_1,o_3)
& earlierEq(o_3,o_2) & achieves(o_3,f))))).
```

Appendix B: Hand Proof for Proposition #01

The following is the hand proof for solving Proposition #01[2]:



Proposition 1. $T_{psl} \models (\forall f, s)(\neg changes(s, f) \equiv (prior(f, s) \equiv holds(f, s)))$

Proof. This follows from the contrapositive of the definition of *changes*. □

[2] This hand proof was done by Xing Tan, but was not formally published.

Appendix C: Hand Proof for Proposition #02

The following is the hand proof for solving Proposition #02[3]:

Proposition 2.

$$T_{psl} \models (\forall f)\,(\exists s)\; changes(s, f) \equiv (\exists s_1)\; prior(f, s_1) \wedge (\exists s_2)\; \neg prior(f, s_2)$$

Proof. $\Rightarrow$)

By the definition of *changes*, we have

$$T_{psl} \models (\forall f, s)\; changes(s, f) \supset$$
$$prior(f, s) \wedge \neg holds(f, s) \vee \neg prior(f, s) \wedge holds(f, s)$$

Axiom 5 of T_{disc_state} gives us

$$T_{psl} \models (\forall f, s)\; changes(s, f) \supset$$
$$(\exists a_1)\; prior(f, s) \wedge \neg prior(f, successor(a_1, s))$$
$$\vee (\exists a_2)\; \neg prior(f, s) \wedge prior(f, successor(a_2, s))$$

which implies

$$T_{psl} \models (\forall f, s)\; changes(s, f) \supset$$
$$(\exists s_1)\; prior(f, s) \wedge \neg prior(f, s_1)$$
$$\vee (\exists s_2)\; \neg prior(f, s) \wedge prior(f, s_2)$$

which simplifies to

$$T_{psl} \models (\forall f, s)\; changes(s, f) \supset$$
$$(\exists s_1)\; prior(f, s_1)$$
$$\wedge (\exists s_2)\; \neg prior(f, s_2)$$

$\Leftarrow$)

Suppose

$$T_{psl} \models (\exists s_1)\; prior(f, s_1) \wedge (\exists s_2)\; \neg prior(f, s_2)$$

By Axiom 2 of $T_{occtree}$, there are three cases:

Case 1:

$$T_{psl} \models (\exists s_1)\; prior(f, s_1) \wedge (\exists s_2)\; \neg prior(f, s_2) \wedge earlier(s_1, s_2) \wedge \neg earlier(s_2, s_1)$$

By the Inertia Theorem,

$$T_{psl} \models (\forall s_1, s_2)\; prior(f, s_1) \wedge \neg prior(f, s_2) \wedge earlier(s_1, s_2) \supset$$
$$(\exists s_3)\; earlier(s_1, s_3) \wedge earlier(s_3, s_2) \wedge prior(f, s_3) \wedge \neg holds(f, s_3)$$

The definition of *changes* gives us

$$T_{psl} \models (\forall s_1, s_2)\; prior(f, s_1) \wedge \neg prior(f, s_2) \wedge earlier(s_1, s_2) \supset$$
$$(\exists s_3)\; earlier(s_1, s_3) \wedge earlier(s_3, s_2) \wedge changes(s_3, f)$$

which simplifies to

$$T_{psl} \models (\forall s_1, s_2)\; prior(f, s_1) \wedge \neg prior(f, s_2) \wedge earlier(s_1, s_2) \supset$$
$$(\exists s)\; changes(s_3, f)$$

[3] This hand proof was done by Xing Tan, but was not formally published.

Case 2:

$$T_{psl} \models (\exists s_1)\, prior(f, s_1) \wedge (\exists s_2)\, \neg prior(f, s_2) \wedge earlier(s_2, s_1) \wedge \neg earlier(s_1, s_2)$$

By the Inertia Theorem,

$$T_{psl} \models (\forall s_1, s_2)\, prior(f, s_1) \wedge \neg prior(f, s_2) \wedge earlier(s_2, s_1) \supset$$
$$(\exists s_3)\, earlier(s_2, s_3) \wedge earlier(s_3, s_1) \wedge \neg prior(f, s_3) \wedge holds(f, s_3)$$

The definition of *changes* gives us

$$T_{psl} \models (\forall s_1, s_2)\, prior(f, s_1) \wedge \neg prior(f, s_2) \wedge earlier(s_1, s_2) \supset$$
$$(\exists s_3)\, earlier(s_1, s_3) \wedge earlier(s_3, s_2) \wedge changes(s_3, f)$$

which simplifies to

$$T_{psl} \models (\forall s_1, s_2)\, prior(f, s_1) \wedge \neg prior(f, s_2) \wedge earlier(s_1, s_2) \supset$$
$$(\exists s)\, changes(s_3, f)$$

Case 3:

$$T_{psl} \models (\exists s_1)\, prior(f, s_1) \wedge (\exists s_2)\, \neg prior(f, s_2)$$
$$\wedge \neg earlier(s_1, s_2) \wedge \neg earlier(s_2, s_1)$$

By Axiom 6 of $T_{occtree}$,

$$T_{psl} \models (\forall s_1, s_2)\, \neg earlier(s_1, s_2) \wedge \neg earlier(s_2, s_1)$$
$$\supset (\exists s_3, s_4)\, initial(s_3) \wedge initial(s_4) \wedge earlier(s_3, s_1) \wedge earlier(s_4, s_2)$$

By Axiom 4 of $T_{discstate}$,

$$T_{psl} \models (\forall f, s_1, s_2)\, \neg earlier(s_1, s_2) \wedge \neg earlier(s_2, s_1)$$
$$\supset (\exists s_3, s_4)\, initial(s_3) \wedge initial(s_4) \wedge earlier(s_3, s_1) \wedge earlier(s_4, s_2)$$
$$\wedge (prior(f, s_3) \equiv prior(f, s_4))$$

The Inertia Theorem gives us

$$T_{psl} \models (\forall f, s_1, s_2)\, \neg earlier(s_1, s_2) \wedge \neg earlier(s_2, s_1)$$
$$\wedge prior(f, s_1) \wedge \neg prior(f, s_2)$$
$$\supset (\exists s_3, s_4, s_5, s_6)\, initial(s_3) \wedge initial(s_4) \wedge earlier(s_3, s_1) \wedge earlier(s_4, s_2)$$
$$\wedge earlier(s_3, s_5) \wedge earlier(s_5, s_1) \wedge earlier(s_4, s_6) \wedge earlier(s_6, s_2)$$
$$\wedge (\neg prior(f, s_5) \wedge holds(f, s_5) \vee prior(f, s_6) \wedge \neg holds(f, s_6))$$

By the definition of *changes* we have

$$T_{psl} \models (\forall f, s_1, s_2)\, \neg earlier(s_1, s_2) \wedge \neg earlier(s_2, s_1)$$
$$\wedge prior(f, s_1) \wedge \neg prior(f, s_2)$$
$$\supset (\exists s_3, s_4, s_5, s_6)\, initial(s_3) \wedge initial(s_4) \wedge earlier(s_3, s_1) \wedge earlier(s_4, s_2)$$
$$\wedge earlier(s_3, s_5) \wedge earlier(s_5, s_1) \wedge earlier(s_4, s_6) \wedge earlier(s_6, s_2)$$
$$\wedge (changes(s_5, f) \vee changes(s_6, f)$$

which simplifies to

$$T_{psl} \models (\forall f, s_1, s_2)\, \neg earlier(s_1, s_2) \wedge \neg earlier(s_2, s_1)$$
$$\wedge prior(f, s_1) \wedge \neg prior(f, s_2)$$
$$\supset (\exists s)\, changes(f, s)$$

Appendix D: Hand Proof for the Inertia Principle

The following is the hand proof for solving the Inertia Principle (taken from [8]):

5.3. **Inertia Principle.** The most interesting consequence of T_{disc_state} is the Inertia Principle: every change of a state is associated with an activity occurrence.

Theorem 15.

$T_{disc\_state} \cup T_{occtree} \cup T_{pslcore} \models (\forall f, o_1, o_2)(holds(f, o_1) \equiv \neg holds(f, o_2)) \wedge earlierEq(o_1, o_2)$
$\supset (\exists a, o_3) earlier(o_1, o_3) \wedge earlier(o_3, o_2) \wedge (holds(f, o_3) \equiv \neg holds(f, successor(a, o_3)))$

Proof.

$$\mathcal{M} \models (\forall f, o_1, o_2)\ (holds(f, o_1) \equiv \neg holds(f, o_2)) \wedge earlierEq(o_1, o_2)$$

$\supset (\exists a, o_3) earlier(o_1, o_3) \wedge earlierEq(o_3, o_2) \wedge (holds(f, o_3) \equiv \neg holds(f, successor(a, o_3)))$
iff for any variable assignment σ and any fluent $\sigma(f)$ and any occurrences $\sigma(o_1), \sigma(o_2)$
if

$$\mathcal{M}, \sigma \models (holds(f, o_1) \equiv \neg holds(f, o_2)) \wedge earlier(o_1, o_2)$$

then

$\mathcal{M}, \sigma \models (\exists a, o_3) earlier(o_1, o_3) \wedge earlier(o_3, o_2) \wedge (holds(f, o_3) \equiv \neg holds(f, successor(a, o_3)))$

Case 1: $\langle \mathbf{f}, \mathbf{o_1} \rangle \in \mathbf{holds}, \langle \mathbf{f}, \mathbf{o_2} \rangle \notin \mathbf{holds}, \langle \mathbf{o_1}, \mathbf{o_2} \rangle \in \mathbf{earlier}$

By condition (3) in the definition of $\mathfrak{M}^{state}$, $\mathbf{o_1}$ is an element of a fluent tree τ for f and $\mathbf{o_2}$ is not an element of τ.

Further, $\mathbf{o_1}$ and $\mathbf{o_2}$ are on the same branch of the occurrence tree containing τ.

Case 1a:

There exists a leaf occurrence $\mathbf{o_3} \in \tau$ such that

$$\langle \mathbf{o_2}, \mathbf{o_3} \rangle, \langle \mathbf{o_1}, \mathbf{o_3} \rangle \in \mathbf{earlier}$$

By the definition of contiguous subtree, we must have $\mathbf{o_2} \in \tau$, which contradicts the hypothesis.

Case 1b:

There exists a leaf occurrence $\mathbf{o_3} \in \tau$ such that

$$\langle \mathbf{o_3}, \mathbf{o_2} \rangle, \langle \mathbf{o_1}, \mathbf{o_3} \rangle \in \mathbf{earlier}$$

since $\mathbf{o_3} \in \tau$, we have

$$\langle \mathrm{f}, \mathbf{o_3} \rangle \in \mathbf{holds}$$

Since $\mathbf{o_3}$ is a leaf occurrence of τ, the successor occurrence of $\mathbf{o_3}$ along the branch containing $\mathbf{o_1}$ and $\mathbf{o_2}$ are elements of τ. By condition (3) in the definition of $\mathfrak{M}^{state}$, this is equivalent to

$$\langle \mathrm{f}, \mathbf{successor}(\mathrm{a}, \mathbf{o_3}) \rangle \notin \mathbf{holds}$$

Case 2: $\langle \mathrm{f}, \mathbf{o_1} \rangle \notin \mathbf{holds}, \langle \mathrm{f}, \mathbf{o_2} \rangle \in \mathbf{holds}, \langle \mathbf{o_1}, \mathbf{o_2} \rangle \in \mathbf{earlier}$

By condition (3) in the definition of $\mathfrak{M}^{state}$, $\mathbf{o_2}$ is an element of a fluent tree τ for f and $\mathbf{o_1}$ is not an element of τ.

Further, $\mathbf{o_1}$ and $\mathbf{o_2}$ are on the same branch of the occurrence tree containing τ.

Case 2a:

There exists a root occurrence $\mathbf{o_3} \in \tau$ such that

$$\langle \mathbf{o_3}, \mathbf{o_1} \rangle \in \mathbf{earlier}$$

By the definition of contiguous subtree, we must have $\mathbf{o_1} \in \tau$, which contradicts the hypothesis.

Case 2b:

There exists a root occurrence $\mathbf{o_3} \in \tau$ such that

$$\langle \mathbf{o_3}, \mathbf{o_2} \rangle, \langle \mathbf{o_1}, \mathbf{o_3} \rangle \in \mathbf{earlier}$$

since $\mathbf{o_3} \in \tau$, we have

$$\langle \mathrm{f}, \mathbf{o_3} \rangle \in \mathbf{holds}$$

Since $\mathbf{o_3}$ is a root occurrence of τ, the predecessor occurrence of $\mathbf{o_3}$ along the branch containing $\mathbf{o_1}$ and $\mathbf{o_2}$ is not an element of τ. By condition (3) in the definition of $\mathfrak{M}^{state}$, this is equivalent to

$$\langle \mathrm{f}, \mathbf{o_4} \rangle \notin \mathbf{holds}$$

where $\mathbf{o_3} = \mathbf{successor}(\mathrm{a}, \mathbf{o_4})$

□

Proofs Generated by Prover9

The following section contains the proofs that were generated in Prover9.

Proposition #01 Proof

This proof corresponds to Ex 4.3.1.

Goal: `all f all s (-changes(s,f) <-> (prior(f,s) <-> holds(f,s))).`

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.05 (+ 0.05) seconds.
% Length of proof is 46.
% Level of proof is 13.
% Maximum clause weight is 14.
% Given clauses 152.

28 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
29 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
35 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
36 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
37 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
40 (all f all s (-changes(s,f) <-> (prior(f,s) <-> holds(f,s)))) #
label(non_clause) # label(goal).  [goal].
62 -holds(x,y) | state(x).  [clausify(28)].
64 -prior(x,y) | state(x).  [clausify(29)].
70 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(35)].
72 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(36)].
81 -changes(x,y) | achieves(x,y) | falsifies(x,y).  [clausify(37)].
83 -falsifies(x,y) | prior(y,x).  [clausify(35)].
84 -falsifies(x,y) | -holds(y,x).  [clausify(35)].
85 changes(x,y) | -falsifies(x,y).  [clausify(37)].
92 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(70,b,64,b)].
93 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(72,b,62,b)].
95 -achieves(x,y) | -prior(y,x).  [clausify(36)].
96 -achieves(x,y) | holds(y,x).  [clausify(36)].
97 changes(x,y) | -achieves(x,y).  [clausify(37)].
103 -changes(x,y) | achieves(x,y) | prior(y,x).  [resolve(81,c,83,a)].
104 -changes(x,y) | achieves(x,y) | -holds(y,x).
[resolve(81,c,84,a)].
135 -holds(x,y) | arboreal(y).  [clausify(28)].
136 -prior(x,y) | arboreal(y).  [clausify(29)].
146 -changes(c2,c1) | -prior(c1,c2) | holds(c1,c2).  [deny(40)].
147 -changes(c2,c1) | prior(c1,c2) | -holds(c1,c2).  [deny(40)].
```

```
148 changes(c2,c1) | prior(c1,c2) | holds(c1,c2).  [deny(40)].
149 changes(c2,c1) | -prior(c1,c2) | -holds(c1,c2).  [deny(40)].
194 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(92,a,85,b)].
195 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(93,a,97,b)].
198 -changes(x,y) | prior(y,x) | holds(y,x).  [resolve(103,b,96,a)].
202 -changes(x,y) | -holds(y,x) | -prior(y,x).  [resolve(104,b,95,a)].
225 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(194,b,d)].
226 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(195,c,d)].
233 prior(c1,c2) | holds(c1,c2).
[resolve(198,a,148,a),merge(c),merge(d)].
234 holds(c1,c2) | -arboreal(c2) | changes(c2,c1).
[resolve(233,a,225,b),merge(c)].
239 holds(c1,c2) | arboreal(c2).  [resolve(233,a,136,a)].
252 arboreal(c2).  [resolve(239,a,135,a),merge(b)].
255 holds(c1,c2) | changes(c2,c1).
[back_unit_del(234),unit_del(b,252)].
287 holds(c1,c2) | -prior(c1,c2).  [resolve(255,b,146,a),merge(c)].
311 holds(c1,c2).  [resolve(287,b,233,a),merge(b)].
312 changes(c2,c1) | -prior(c1,c2).
[back_unit_del(149),unit_del(c,311)].
313 -changes(c2,c1) | prior(c1,c2).
[back_unit_del(147),unit_del(c,311)].
315 prior(c1,c2) | changes(c2,c1).
[resolve(311,a,226,c),unit_del(a,252)].
376 prior(c1,c2).  [resolve(315,b,313,a),merge(b)].
377 changes(c2,c1).  [back_unit_del(312),unit_del(b,376)].
382 $F.  [resolve(377,a,202,a),unit_del(a,311),unit_del(b,376)].

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

Proposition #01 (Revised) Proof

This proof corresponds to Ex 4.3.2.

Goal: `all f all s (state(f) & arboreal(s) -> (-changes(s,f) <-> (prior(f,s) <-> holds(f,s)))).`

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.08 (+ 0.01) seconds.
% Length of proof is 42.
% Level of proof is 11.
% Maximum clause weight is 14.
% Given clauses 146.

28 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
29 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
35 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
36 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
37 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
40 (all f all s (state(f) & arboreal(s) -> (-changes(s,f) <->
(prior(f,s) <-> holds(f,s))))) # label(non_clause) # label(goal).
[goal].
62 -holds(x,y) | state(x).  [clausify(28)].
64 -prior(x,y) | state(x).  [clausify(29)].
70 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(35)].
72 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(36)].
82 -changes(x,y) | achieves(x,y) | falsifies(x,y).  [clausify(37)].
84 -falsifies(x,y) | prior(y,x).  [clausify(35)].
85 -falsifies(x,y) | -holds(y,x).  [clausify(35)].
86 changes(x,y) | -falsifies(x,y).  [clausify(37)].
93 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(70,b,64,b)].
94 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(72,b,62,b)].
96 -achieves(x,y) | -prior(y,x).  [clausify(36)].
97 -achieves(x,y) | holds(y,x).  [clausify(36)].
98 changes(x,y) | -achieves(x,y).  [clausify(37)].
104 -changes(x,y) | achieves(x,y) | prior(y,x).  [resolve(82,c,84,a)].
105 -changes(x,y) | achieves(x,y) | -holds(y,x).
[resolve(82,c,85,a)].
147 arboreal(c2).  [deny(40)].
148 -changes(c2,c1) | -prior(c1,c2) | holds(c1,c2).  [deny(40)].
```

```
149 -changes(c2,c1) | prior(c1,c2) | -holds(c1,c2).  [deny(40)].
150 changes(c2,c1) | prior(c1,c2) | holds(c1,c2).  [deny(40)].
151 changes(c2,c1) | -prior(c1,c2) | -holds(c1,c2).  [deny(40)].
199 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(93,a,86,b)].
200 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(94,a,98,b)].
203 -changes(x,y) | prior(y,x) | holds(y,x).  [resolve(104,b,97,a)].
207 -changes(x,y) | -holds(y,x) | -prior(y,x).  [resolve(105,b,96,a)].
230 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(199,b,d)].
231 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(200,c,d)].
243 prior(c1,c2) | holds(c1,c2).
[resolve(203,a,150,a),merge(c),merge(d)].
290 holds(c1,c2) | changes(c2,c1).
[resolve(243,a,230,b),merge(c),unit_del(b,147)].
296 holds(c1,c2) | -prior(c1,c2).  [resolve(290,b,148,a),merge(c)].
303 holds(c1,c2).  [resolve(296,b,243,a),merge(b)].
304 changes(c2,c1) | -prior(c1,c2).
[back_unit_del(151),unit_del(c,303)].
305 -changes(c2,c1) | prior(c1,c2).
[back_unit_del(149),unit_del(c,303)].
306 prior(c1,c2) | changes(c2,c1).
[resolve(303,a,231,c),unit_del(a,147)].
350 prior(c1,c2).  [resolve(306,b,305,a),merge(b)].
351 changes(c2,c1).  [back_unit_del(304),unit_del(b,350)].
353 $F.  [resolve(351,a,207,a),unit_del(a,303),unit_del(b,350)].

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

Proposition #02 (Case #01) Proof

This proof corresponds to Ex 4.4.6.

Goal:

```
(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & (prior(f,s_1) <-> -prior(f,s_2))
     & earlier(s_1,s_2))
     ->
     (exists s
          (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.20 (+ 0.01) seconds.
% Length of proof is 85.
% Level of proof is 31.
% Maximum clause weight is 25.
% Given clauses 268.

16 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
18 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
19 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
20 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
21 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
22 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
23 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
prior(f,o_1) & -prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & prior(f,o_3) & -
holds(f,o_3))))) # label(non_clause).  [assumption].
24 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
-prior(f,o_1) & prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & -prior(f,o_3) &
holds(f,o_3))))) # label(non_clause).  [assumption].
25 -(all f all s_1 all s_2 (state(f) & arboreal(s_1) & arboreal(s_2) &
(prior(f,s_1) <-> -prior(f,s_2)) & earlier(s_1,s_2) -> (exists s
(changes(s,f) & arboreal(s))))) # label(non_clause).  [assumption].
34 -earlierEq(x,y) | arboreal(y).  [clausify(16)].
38 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | earlierEq(y,f5(x,y,z)).  [clausify(23)].
40 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | earlierEq(y,f6(x,y,z)).  [clausify(24)].
```

```
42 -holds(x,y) | state(x).  [clausify(18)].
44 -prior(x,y) | state(x).  [clausify(19)].
46 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(20)].
48 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(21)].
49 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | prior(x,f5(x,y,z)).  [clausify(23)].
50 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | -holds(x,f5(x,y,z)).  [clausify(23)].
51 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | -prior(x,f6(x,y,z)).  [clausify(24)].
52 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | holds(x,f6(x,y,z)).  [clausify(24)].
54 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | arboreal(f5(x,y,z)).  [resolve(38,g,34,a)].
57 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | arboreal(f6(x,y,z)).  [resolve(40,g,34,a)].
64 changes(x,y) | -falsifies(x,y).  [clausify(22)].
67 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(46,b,44,b)].
68 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(48,b,42,b)].
72 changes(x,y) | -achieves(x,y).  [clausify(22)].
95 arboreal(c2).  [clausify(25)].
96 arboreal(c3).  [clausify(25)].
97 -prior(c1,c2) | -prior(c1,c3).  [clausify(25)].
98 prior(c1,c2) | prior(c1,c3).  [clausify(25)].
99 earlier(c2,c3).  [clausify(25)].
100 -changes(x,c1) | -arboreal(x).  [clausify(25)].
113 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)) | -prior(z,u).
[resolve(49,a,44,b)].
114 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)) | -holds(z,u).
[resolve(50,a,42,b)].
116 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)) | -prior(z,u).
[resolve(51,a,44,b)].
118 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)) | -prior(z,u).
[resolve(52,a,44,b)].
121 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | arboreal(f5(z,x,y)) | -prior(z,u).
[resolve(54,a,44,b)].
129 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)) | -prior(z,u).
[resolve(57,a,44,b)].
137 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(67,a,64,b)].
138 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(68,a,72,b)].
```

```
148 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)).  [factor(113,c,g)].
149 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)).  [factor(114,f,g)].
150 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)).  [factor(116,d,g)].
151 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)).  [factor(118,d,g)].
152 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | arboreal(f5(z,x,y)).  [factor(121,c,g)].
155 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)).  [factor(129,d,g)].
158 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(137,b,d)].
159 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(138,c,d)].
175 -prior(x,c2) | prior(x,c3) | prior(x,f5(x,c2,c3)).
[resolve(148,e,99,a),unit_del(a,95),unit_del(b,96)].
176 prior(x,c2) | -prior(x,c3) | -prior(x,f6(x,c2,c3)).
[resolve(150,e,99,a),unit_del(a,95),unit_del(b,96)].
177 prior(x,c2) | -prior(x,c3) | holds(x,f6(x,c2,c3)).
[resolve(151,e,99,a),unit_del(a,95),unit_del(b,96)].
178 -prior(x,c2) | prior(x,c3) | arboreal(f5(x,c2,c3)).
[resolve(152,e,99,a),unit_del(a,95),unit_del(b,96)].
181 prior(x,c2) | -prior(x,c3) | arboreal(f6(x,c2,c3)).
[resolve(155,e,99,a),unit_del(a,95),unit_del(b,96)].
270 prior(c1,c2) | arboreal(f6(c1,c2,c3)).
[resolve(181,b,98,b),merge(c)].
271 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | arboreal(f5(c1,c2,c3)).
[resolve(270,a,178,a)].
272 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | prior(c1,f5(c1,c2,c3)).
[resolve(270,a,175,a)].
277 prior(c1,c2) | holds(c1,f6(c1,c2,c3)).
[resolve(177,b,98,b),merge(c)].
278 prior(c1,c2) | -arboreal(f6(c1,c2,c3)) | prior(c1,f6(c1,c2,c3)) |
changes(f6(c1,c2,c3),c1).  [resolve(277,b,159,c)].
303 arboreal(f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)) | -prior(c1,c2).
[resolve(271,b,97,b)].
305 arboreal(f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)).
[resolve(303,c,270,a),merge(c)].
310 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | -arboreal(f5(c1,c2,c3)) |
holds(c1,f5(c1,c2,c3)) | changes(f5(c1,c2,c3),c1).
[resolve(272,c,158,b)].
414 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | changes(f6(c1,c2,c3),c1) |
arboreal(f5(c1,c2,c3)).  [resolve(278,b,305,a)].
458 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)) | -
arboreal(f6(c1,c2,c3)).  [resolve(414,c,100,a)].
460 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)).
[resolve(458,d,305,a),merge(d)].
461 prior(c1,c2) | arboreal(f5(c1,c2,c3)) | -prior(c1,c3).
[resolve(460,b,176,c),merge(c)].
462 prior(c1,c2) | arboreal(f5(c1,c2,c3)).
```

```
[resolve(461,c,98,b),merge(c)].
469 arboreal(f5(c1,c2,c3)) | prior(c1,c3).
[resolve(462,a,178,a),merge(c)].
492 arboreal(f5(c1,c2,c3)) | -prior(c1,c2).  [resolve(469,b,97,b)].
493 arboreal(f5(c1,c2,c3)).  [resolve(492,b,462,a),merge(b)].
516 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | holds(c1,f5(c1,c2,c3)) |
changes(f5(c1,c2,c3),c1).  [back_unit_del(310),unit_del(c,493)].
518 -changes(f5(c1,c2,c3),c1).  [ur(100,b,493,a)].
519 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | holds(c1,f5(c1,c2,c3)).
[back_unit_del(516),unit_del(d,518)].
536 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | -prior(c1,c2).
[resolve(519,c,149,f),merge(f),unit_del(c,95),unit_del(d,96),unit_del(
f,99)].
541 arboreal(f6(c1,c2,c3)) | prior(c1,c3).
[resolve(536,c,270,a),merge(c)].
545 arboreal(f6(c1,c2,c3)) | -prior(c1,c2).  [resolve(541,b,97,b)].
554 arboreal(f6(c1,c2,c3)).  [resolve(545,b,270,a),merge(b)].
578 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | changes(f6(c1,c2,c3),c1).
[back_unit_del(278),unit_del(b,554)].
580 -changes(f6(c1,c2,c3),c1).  [ur(100,b,554,a)].
581 prior(c1,c2) | prior(c1,f6(c1,c2,c3)).
[back_unit_del(578),unit_del(c,580)].
589 prior(c1,c2) | -prior(c1,c3).  [resolve(581,b,176,c),merge(b)].
593 prior(c1,c2).  [resolve(589,b,98,b),merge(b)].
594 -prior(c1,c3).  [back_unit_del(97),unit_del(a,593)].
613 prior(c1,f5(c1,c2,c3)).  [resolve(593,a,175,a),unit_del(a,594)].
615 holds(c1,f5(c1,c2,c3)).
[resolve(613,a,158,b),unit_del(a,493),unit_del(c,518)].
617 $F.
[ur(149,a,95,a,b,96,a,c,593,a,d,594,a,e,99,a),unit_del(a,615)].

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

Proposition #02 (Case #02) Proof

This proof corresponds to Ex 4.4.7.

Goal:

```
(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & (prior(f,s_1) <-> -prior(f,s_2))
     & earlier(s_2,s_1))
     ->
     (exists s
          (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.14 (+ 0.06) seconds.
% Length of proof is 79.
% Level of proof is 24.
% Maximum clause weight is 25.
% Given clauses 299.

16 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
18 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
19 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
20 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
21 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
22 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
23 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
prior(f,o_1) & -prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & prior(f,o_3) & -
holds(f,o_3))))) # label(non_clause).  [assumption].
24 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
-prior(f,o_1) & prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & -prior(f,o_3) &
holds(f,o_3))))) # label(non_clause).  [assumption].
25 -(all f all s_1 all s_2 (state(f) & arboreal(s_1) & arboreal(s_2) &
(prior(f,s_1) <-> -prior(f,s_2)) & earlier(s_2,s_1) -> (exists s
(changes(s,f) & arboreal(s))))) # label(non_clause).  [assumption].
34 -earlierEq(x,y) | arboreal(y).  [clausify(16)].
38 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | earlierEq(y,f5(x,y,z)).  [clausify(23)].
40 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | earlierEq(y,f6(x,y,z)).  [clausify(24)].
```

```
42 -holds(x,y) | state(x).  [clausify(18)].
44 -prior(x,y) | state(x).  [clausify(19)].
46 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(20)].
48 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(21)].
49 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | prior(x,f5(x,y,z)).  [clausify(23)].
50 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | -holds(x,f5(x,y,z)).  [clausify(23)].
51 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | -prior(x,f6(x,y,z)).  [clausify(24)].
52 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | holds(x,f6(x,y,z)).  [clausify(24)].
54 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | arboreal(f5(x,y,z)).  [resolve(38,g,34,a)].
57 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | arboreal(f6(x,y,z)).  [resolve(40,g,34,a)].
64 changes(x,y) | -falsifies(x,y).  [clausify(22)].
67 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(46,b,44,b)].
68 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(48,b,42,b)].
72 changes(x,y) | -achieves(x,y).  [clausify(22)].
95 arboreal(c2).  [clausify(25)].
96 arboreal(c3).  [clausify(25)].
97 -prior(c1,c2) | -prior(c1,c3).  [clausify(25)].
98 prior(c1,c2) | prior(c1,c3).  [clausify(25)].
99 earlier(c3,c2).  [clausify(25)].
100 -changes(x,c1) | -arboreal(x).  [clausify(25)].
113 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)) | -prior(z,u).
[resolve(49,a,44,b)].
114 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)) | -holds(z,u).
[resolve(50,a,42,b)].
116 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)) | -prior(z,u).
[resolve(51,a,44,b)].
118 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)) | -prior(z,u).
[resolve(52,a,44,b)].
121 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | arboreal(f5(z,x,y)) | -prior(z,u).
[resolve(54,a,44,b)].
129 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)) | -prior(z,u).
[resolve(57,a,44,b)].
137 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(67,a,64,b)].
138 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(68,a,72,b)].
```

```
148 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)).  [factor(113,c,g)].
149 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)).  [factor(114,f,g)].
150 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)).  [factor(116,d,g)].
151 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)).  [factor(118,d,g)].
152 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | arboreal(f5(z,x,y)).  [factor(121,c,g)].
155 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)).  [factor(129,d,g)].
158 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(137,b,d)].
159 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(138,c,d)].
175 -prior(x,c3) | prior(x,c2) | prior(x,f5(x,c3,c2)).
[resolve(148,e,99,a),unit_del(a,96),unit_del(b,95)].
176 prior(x,c3) | -prior(x,c2) | -prior(x,f6(x,c3,c2)).
[resolve(150,e,99,a),unit_del(a,96),unit_del(b,95)].
177 prior(x,c3) | -prior(x,c2) | holds(x,f6(x,c3,c2)).
[resolve(151,e,99,a),unit_del(a,96),unit_del(b,95)].
178 -prior(x,c3) | prior(x,c2) | arboreal(f5(x,c3,c2)).
[resolve(152,e,99,a),unit_del(a,96),unit_del(b,95)].
181 prior(x,c3) | -prior(x,c2) | arboreal(f6(x,c3,c2)).
[resolve(155,e,99,a),unit_del(a,96),unit_del(b,95)].
267 prior(c1,c2) | arboreal(f5(c1,c3,c2)).
[resolve(178,a,98,b),merge(c)].
268 prior(c1,c2) | prior(c1,f5(c1,c3,c2)).
[resolve(175,a,98,b),merge(c)].
273 prior(c1,c2) | -arboreal(f5(c1,c3,c2)) | holds(c1,f5(c1,c3,c2)) |
changes(f5(c1,c3,c2),c1).  [resolve(268,b,158,b)].
274 prior(c1,c3) | arboreal(f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)).
[resolve(181,b,267,a)].
278 prior(c1,c3) | holds(c1,f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)).
[resolve(177,b,267,a)].
303 arboreal(f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)) | -prior(c1,c2).
[resolve(274,a,97,b)].
305 arboreal(f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)).
[resolve(303,c,267,a),merge(c)].
310 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | -arboreal(f6(c1,c3,c2)) |
prior(c1,f6(c1,c3,c2)) | changes(f6(c1,c3,c2),c1).
[resolve(278,b,159,c)].
610 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | prior(c1,f6(c1,c3,c2)) |
changes(f6(c1,c3,c2),c1).  [resolve(310,c,305,a),merge(e)].
632 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | prior(c1,f6(c1,c3,c2)) | -
arboreal(f6(c1,c3,c2)).  [resolve(610,d,100,a)].
635 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | prior(c1,f6(c1,c3,c2)).
[resolve(632,d,305,a),merge(d)].
636 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | -prior(c1,c2).
[resolve(635,c,176,c),merge(c)].
641 prior(c1,c3) | arboreal(f5(c1,c3,c2)).
```

```
[resolve(636,c,267,a),merge(c)].
647 arboreal(f5(c1,c3,c2)) | -prior(c1,c2).  [resolve(641,a,97,b)].
652 arboreal(f5(c1,c3,c2)).  [resolve(647,b,267,a),merge(b)].
684 prior(c1,c2) | holds(c1,f5(c1,c3,c2)) | changes(f5(c1,c3,c2),c1).
[back_unit_del(273),unit_del(b,652)].
686 -changes(f5(c1,c3,c2),c1).  [ur(100,b,652,a)].
687 prior(c1,c2) | holds(c1,f5(c1,c3,c2)).
[back_unit_del(684),unit_del(c,686)].
715 prior(c1,c2) | -prior(c1,c3).
[resolve(687,b,149,f),merge(e),unit_del(b,96),unit_del(c,95),unit_del(
e,99)].
716 prior(c1,c2).  [resolve(715,b,98,b),merge(b)].
717 -prior(c1,c3).  [back_unit_del(97),unit_del(a,716)].
736 arboreal(f6(c1,c3,c2)).  [resolve(716,a,181,b),unit_del(a,717)].
737 holds(c1,f6(c1,c3,c2)).  [resolve(716,a,177,b),unit_del(a,717)].
739 -prior(c1,f6(c1,c3,c2)).  [ur(176,a,717,a,b,716,a)].
745 -changes(f6(c1,c3,c2),c1).  [ur(100,b,736,a)].
753 $F.
[resolve(737,a,159,c),unit_del(a,736),unit_del(b,739),unit_del(c,745)]
.

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

Proposition #02 (Case #03a) Proof

This proof corresponds to Ex 4.4.15.

Goal:

```
(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlier(s_3,s_1)
      & -prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 2.25 (+ 0.14) seconds.
% Length of proof is 42.
% Level of proof is 9.
% Maximum clause weight is 22.
% Given clauses 2430.

20 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
21 (all s (initial(s) <-> arboreal(s) & -(exists sp earlier(sp,s)))) #
label(non_clause).  [assumption].
23 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
24 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
28 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
29 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
31 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
-prior(f,o_1) & prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & -prior(f,o_3) &
holds(f,o_3))))) # label(non_clause).  [assumption].
34 -(all f all s_1 all s_2 all s_3 (state(f) & arboreal(s_1) &
arboreal(s_2) & prior(f,s_1) & -prior(f,s_2) & initial(s_3) &
earlier(s_3,s_1) & -prior(f,s_3) & -earlier(s_1,s_2) & -
earlier(s_2,s_1) -> (exists s (changes(s,f) & arboreal(s))))) #
label(non_clause).  [assumption].
45 -earlierEq(x,y) | arboreal(y).  [clausify(20)].
51 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
```

```
| -earlier(y,z) | earlierEq(y,f8(x,y,z)).  [clausify(31)].
54 -holds(x,y) | state(x).  [clausify(23)].
56 -prior(x,y) | state(x).  [clausify(24)].
60 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(28)].
63 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | -prior(x,f8(x,y,z)).  [clausify(31)].
64 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | holds(x,f8(x,y,z)).  [clausify(31)].
74 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | arboreal(f8(x,y,z)).  [resolve(51,g,45,a)].
86 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(60,b,54,b)].
90 changes(x,y) | -achieves(x,y).  [clausify(29)].
118 -initial(x) | arboreal(x).  [clausify(21)].
128 arboreal(c2).  [clausify(34)].
130 prior(c1,c2).  [clausify(34)].
132 initial(c4).  [clausify(34)].
133 earlier(c4,c2).  [clausify(34)].
134 -prior(c1,c4).  [clausify(34)].
137 -changes(x,c1) | -arboreal(x).  [clausify(34)].
163 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f8(z,x,y)) | -prior(z,u).
[resolve(63,a,56,b)].
165 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f8(z,x,y)) | -prior(z,u).
[resolve(64,a,56,b)].
186 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f8(z,x,y)) | -prior(z,u).
[resolve(74,a,56,b)].
197 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(86,a,90,b)].
210 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f8(z,x,y)).  [factor(163,d,g)].
211 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f8(z,x,y)).  [factor(165,d,g)].
220 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f8(z,x,y)).  [factor(186,d,g)].
225 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(197,c,d)].
233 arboreal(c4).  [resolve(132,a,118,a)].
251 prior(x,c4) | -prior(x,c2) | -prior(x,f8(x,c4,c2)).
[resolve(210,e,133,a),unit_del(a,233),unit_del(b,128)].
252 prior(x,c4) | -prior(x,c2) | holds(x,f8(x,c4,c2)).
[resolve(211,e,133,a),unit_del(a,233),unit_del(b,128)].
261 prior(x,c4) | -prior(x,c2) | arboreal(f8(x,c4,c2)).
[resolve(220,e,133,a),unit_del(a,233),unit_del(b,128)].
575 holds(c1,f8(c1,c4,c2)).  [resolve(252,b,130,a),unit_del(a,134)].
692 arboreal(f8(c1,c4,c2)).  [resolve(261,b,130,a),unit_del(a,134)].
1089 prior(c1,f8(c1,c4,c2)) | changes(f8(c1,c4,c2),c1).
[resolve(575,a,225,c),unit_del(a,692)].
17968 prior(c1,f8(c1,c4,c2)).
```

```
[resolve(1089,b,137,a),unit_del(b,692)].
17969 $F.  [resolve(17968,a,251,c),unit_del(a,134),unit_del(b,130)].

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

Proposition #02 (Case #03b) Proof

This proof corresponds to Ex 4.4.16.

Goal:

```
((all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlier(s_3,s_1)
      & prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 17.53 (+ 0.66) seconds.
% Length of proof is 70.
% Level of proof is 12.
% Maximum clause weight is 20.
% Given clauses 9436.

6 (all occ (activity_occurrence(occ) -> (exists a (activity(a) &
occurrence_of(occ,a))))) # label(non_clause).  [assumption].
8 (all s (arboreal(s) -> activity_occurrence(s))) # label(non_clause).
[assumption].
13 (all s1 all s2 (earlier(s1,s2) -> (exists sp (initial(sp) &
earlierEq(sp,s1))))) # label(non_clause).  [assumption].
16 (all s all a (occurrence_of(s,a) -> (arboreal(s) <->
generator(a)))) # label(non_clause).  [assumption].
19 (all a all s1 all s2 (generator(a) -> (earlier(s1,successor(a,s2))
<-> earlierEq(s1,s2)))) # label(non_clause).  [assumption].
20 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
21 (all s (initial(s) <-> arboreal(s) & -(exists sp earlier(sp,s)))) #
label(non_clause).  [assumption].
24 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
25 (all occ1 all occ2 all f (initial(occ1) & initial(occ2) ->
(prior(f,occ1) <-> prior(f,occ2)))) # label(non_clause).
[assumption].
32 (all s1 (arboreal(s1) -> (exists s2 (initial(s2) &
earlierEq(s2,s1))))) # label(non_clause).  [assumption].
33 (all f all s_1 all s_2 (state(f) & prior(f,s_1) & arboreal(s_2) & -
prior(f,s_2) & earlier(s_1,s_2) -> (exists s (changes(s,f) &
arboreal(s))))) # label(non_clause).  [assumption].
```

```
34 -(all f all s_1 all s_2 all s_3 (state(f) & arboreal(s_1) &
arboreal(s_2) & prior(f,s_1) & -prior(f,s_2) & initial(s_3) &
earlier(s_3,s_1) & prior(f,s_3) & -earlier(s_1,s_2) & -
earlier(s_2,s_1) -> (exists s (changes(s,f) & arboreal(s))))) #
label(non_clause).  [assumption].
41 -generator(x) | earlier(y,successor(x,z)) | -earlierEq(y,z).
[clausify(19)].
42 -earlier(x,y) | earlierEq(f2(x,y),x).  [clausify(13)].
44 -earlierEq(x,y) | arboreal(x).  [clausify(20)].
46 -earlierEq(x,y) | earlier(x,y) | y = x.  [clausify(20)].
53 -arboreal(x) | earlierEq(f9(x),x).  [clausify(32)].
56 -prior(x,y) | state(x).  [clausify(24)].
65 -state(x) | -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z)
| changes(f10(x,y,z),x).  [clausify(33)].
66 -state(x) | -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z)
| arboreal(f10(x,y,z)).  [clausify(33)].
100 -activity_occurrence(x) | occurrence_of(x,f1(x)).  [clausify(6)].
102 -arboreal(x) | activity_occurrence(x).  [clausify(8)].
112 -occurrence_of(x,y) | -arboreal(x) | generator(y).
[clausify(16)].
119 -initial(x) | -earlier(y,x).  [clausify(21)].
123 -initial(x) | -initial(y) | -prior(z,x) | prior(z,y).
[clausify(25)].
127 -arboreal(x) | initial(f9(x)).  [clausify(32)].
129 arboreal(c3).  [clausify(34)].
131 -prior(c1,c3).  [clausify(34)].
132 initial(c4).  [clausify(34)].
134 prior(c1,c4).  [clausify(34)].
137 -changes(x,c1) | -arboreal(x).  [clausify(34)].
146 arboreal(f2(x,y)) | -earlier(x,y).  [resolve(44,a,42,b)].
148 earlier(f2(x,y),x) | x = f2(x,y) | -earlier(x,y).
[resolve(46,a,42,b)].
149 earlier(f2(x,y),x) | f2(x,y) = x | -earlier(x,y).
[copy(148),flip(b)].
153 -arboreal(x) | -generator(y) | earlier(f9(x),successor(y,x)).
[resolve(53,b,41,c)].
154 -arboreal(x) | arboreal(f9(x)).  [resolve(53,b,44,a)].
155 -arboreal(x) | earlier(f9(x),x) | x = f9(x).
[resolve(53,b,46,a)].
156 -arboreal(x) | earlier(f9(x),x) | f9(x) = x.  [copy(155),flip(c)].
167 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
changes(f10(x,y,z),x) | -prior(x,u).  [resolve(65,a,56,b)].
169 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
arboreal(f10(x,y,z)) | -prior(x,u).  [resolve(66,a,56,b)].
212 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
changes(f10(x,y,z),x).  [factor(167,a,f)].
213 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
arboreal(f10(x,y,z)).  [factor(169,a,f)].
229 initial(f9(c3)).  [resolve(129,a,127,a)].
231 activity_occurrence(c3).  [resolve(129,a,102,a)].
242 -initial(x) | prior(c1,x).
[resolve(134,a,123,c),unit_del(a,132)].
```

```
247 arboreal(f9(c3)).  [resolve(154,a,129,a)].
249 earlier(f9(c3),c3) | f9(c3) = c3.  [resolve(156,a,129,a)].
272 occurrence_of(c3,f1(c3)).  [resolve(231,a,100,a)].
284 activity_occurrence(f9(c3)).  [resolve(247,a,102,a)].
320 occurrence_of(f9(c3),f1(f9(c3))).  [resolve(284,a,100,a)].
400 generator(f1(c3)).  [resolve(272,a,112,a),unit_del(a,129)].
404 -arboreal(x) | earlier(f9(x),successor(f1(c3),x)).
[resolve(400,a,153,b)].
437 prior(c1,f9(c3)).  [resolve(242,a,229,a)].
520 f9(c3) = c3 | -prior(x,f9(c3)) | prior(x,c3) |
arboreal(f10(x,f9(c3),c3)).  [resolve(249,a,213,d),unit_del(c,129)].
521 f9(c3) = c3 | -prior(x,f9(c3)) | prior(x,c3) |
changes(f10(x,f9(c3),c3),x).  [resolve(249,a,212,d),unit_del(c,129)].
688 generator(f1(f9(c3))).  [resolve(320,a,112,a),unit_del(a,247)].
693 -arboreal(x) | earlier(f9(x),successor(f1(f9(c3)),x)).
[resolve(688,a,153,b)].
2609 earlier(f9(c3),successor(f1(c3),c3)).  [resolve(404,a,129,a)].
4201 f9(c3) = c3 | arboreal(f10(c1,f9(c3),c3)).
[resolve(520,b,437,a),unit_del(b,131)].
4274 f9(c3) = c3 | changes(f10(c1,f9(c3),c3),c1).
[resolve(521,b,437,a),unit_del(b,131)].
4366 earlier(f2(f9(c3),successor(f1(c3),c3)),f9(c3)) |
f2(f9(c3),successor(f1(c3),c3)) = f9(c3).  [resolve(2609,a,149,c)].
7232 earlier(f9(c3),successor(f1(f9(c3)),c3)).
[resolve(693,a,129,a)].
14119 arboreal(f2(f9(c3),successor(f1(f9(c3)),c3))).
[resolve(7232,a,146,b)].
25952 initial(f9(f2(f9(c3),successor(f1(f9(c3)),c3)))).
[resolve(14119,a,127,a)].
36056 prior(c1,f9(f2(f9(c3),successor(f1(f9(c3)),c3)))).
[resolve(25952,a,242,a)].
38609 f9(c3) = c3 | -arboreal(f10(c1,f9(c3),c3)).
[resolve(4274,b,137,a)].
39095 f2(f9(c3),successor(f1(c3),c3)) = f9(c3).
[resolve(4366,a,119,b),unit_del(b,229)].
40124 f9(c3) = c3.  [resolve(38609,b,4201,b),merge(b)].
40136 f2(c3,successor(f1(c3),c3)) = c3.
[back_rewrite(39095),rewrite([40124(2),40124(8)])].
40155 $F.
[back_rewrite(36056),rewrite([40124(3),40124(4),40136(7),40124(3)]),un
it_del(a,131)].

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

Inertia Principle (Part #01) Proof

This proof corresponds to Ex 4.5.1.4.

Goal:

```
(all f all o_1 all o
    ((state(f)
& arboreal(o_1)
& holds(f,o_1)
& -prior(f,o)
& earlier(o,o_1))
    ->
    (exists o_3
(earlierEq(o_3,o_1)
& -prior(f,o_3)
& holds(f,o_3))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.12 (+ 0.06) seconds.
% Length of proof is 28.
% Level of proof is 9.
% Maximum clause weight is 20.
% Given clauses 357.

24 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
38 (all f all o (always(f,o) <-> (all o1 (earlier(o1,o) ->
prior(f,o1))))) # label(non_clause).  [assumption].
39 (all f all o1 (holds(f,o1) -> always(f,o1) | (exists o2
(earlierEq(o2,o1) & -prior(f,o2) & (all o3 (earlierEq(o2,o3) &
earlierEq(o3,o1) -> holds(f,o3)))))))) # label(non_clause).
[assumption].
40 -(all f all o_1 all o (state(f) & arboreal(o_1) & holds(f,o_1) & -
prior(f,o) & earlier(o,o_1) -> (exists o_3 (earlierEq(o_3,o_1) & -
prior(f,o_3) & holds(f,o_3))))) # label(non_clause).  [assumption].
99 -always(x,y) | -earlier(z,y) | prior(x,z).  [clausify(38)].
101 -holds(x,y) | always(x,y) | earlierEq(f11(x,y),y).
[clausify(39)].
102 -holds(x,y) | always(x,y) | -prior(x,f11(x,y)).  [clausify(39)].
103 -holds(x,y) | always(x,y) | -earlierEq(f11(x,y),z) | -
earlierEq(z,y) | holds(x,z).  [clausify(39)].
144 -earlierEq(x,y) | arboreal(x).  [clausify(24)].
148 earlierEq(x,y) | -arboreal(x) | -arboreal(y) | y != x.
[clausify(24)].
164 holds(c1,c2).  [clausify(40)].
165 -prior(c1,c3).  [clausify(40)].
166 earlier(c3,c2).  [clausify(40)].
167 -earlierEq(x,c2) | prior(c1,x) | -holds(c1,x).  [clausify(40)].
208 -holds(x,y) | earlierEq(f11(x,y),y) | -earlier(z,y) | prior(x,z).
[resolve(101,b,99,a)].
```

```
209 -holds(x,y) | -prior(x,f11(x,y)) | -earlier(z,y) | prior(x,z).
[resolve(102,b,99,a)].
210 -holds(x,y) | -earlierEq(f11(x,y),z) | -earlierEq(z,y) |
holds(x,z) | -earlier(u,y) | prior(x,u).  [resolve(103,b,99,a)].
221 earlierEq(x,x) | -arboreal(x).  [factor(148,b,c),xx(c)].
250 -holds(x,c2) | earlierEq(f11(x,c2),c2) | prior(x,c3).
[resolve(208,c,166,a)].
251 -holds(x,c2) | -prior(x,f11(x,c2)) | prior(x,c3).
[resolve(209,c,166,a)].
252 -holds(x,c2) | -earlierEq(f11(x,c2),y) | -earlierEq(y,c2) |
holds(x,y) | prior(x,c3).  [resolve(210,e,166,a)].
507 earlierEq(f11(c1,c2),c2).  [resolve(250,a,164,a),unit_del(b,165)].
510 prior(c1,f11(c1,c2)) | -holds(c1,f11(c1,c2)).
[resolve(507,a,167,a)].
513 arboreal(f11(c1,c2)).  [resolve(507,a,144,a)].
515 earlierEq(f11(c1,c2),f11(c1,c2)).  [resolve(513,a,221,b)].
1149 holds(c1,f11(c1,c2)).
[resolve(515,a,252,b),unit_del(a,164),unit_del(b,507),unit_del(d,165)]
.
1151 prior(c1,f11(c1,c2)).  [back_unit_del(510),unit_del(b,1149)].
1167 $F.  [resolve(1151,a,251,b),unit_del(a,164),unit_del(b,165)].

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

Inertia Principle (Part #02) Proof

This proof corresponds to Ex 4.5.2.12.

Goal:

```
(all f all o2
     ((state(f)
     & arboreal(o2)
     & -initial(o2)
     & -prior(f,o2))
     ->
     (exists o3 exists a
          ((o2 = successor(a,o3))
          & -holds(f,o3))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.16 (+ 0.06) seconds.
% Length of proof is 48.
% Level of proof is 8.
% Maximum clause weight is 12.
% Given clauses 353.

6 (all occ (activity_occurrence(occ) -> (exists a (activity(a) &
occurrence_of(occ,a))))) # label(non_clause).  [assumption].
7 (all o all a1 all a2 (occurrence_of(o,a1) & occurrence_of(o,a2) ->
a1 = a2)) # label(non_clause).  [assumption].
8 (all s (arboreal(s) -> activity_occurrence(s))) # label(non_clause).
[assumption].
9 (all s1 all s2 (earlier(s1,s2) -> arboreal(s1) & arboreal(s2))) #
label(non_clause).  [assumption].
16 (all s all a (occurrence_of(s,a) -> (arboreal(s) <->
generator(a)))) # label(non_clause).  [assumption].
17 (all a all o (occurrence_of(successor(a,o),a) <-> generator(a) &
arboreal(o))) # label(non_clause).  [assumption].
26 (all s (initial(s) <-> arboreal(s) & -(exists sp earlier(sp,s)))) #
label(non_clause).  [assumption].
31 (all a all occ all f (prior(f,successor(a,occ)) <-> holds(f,occ) &
generator(a))) # label(non_clause).  [assumption].
43 (all o2 (arboreal(o2) & -initial(o2) -> (exists o3 exists a o2 =
successor(a,o3)))) # label(non_clause).  [assumption].
44 (all a all o (arboreal(successor(a,o)) -> generator(a) &
arboreal(o))) # label(non_clause).  [assumption].
45 -(all f all o2 (state(f) & arboreal(o2) & -initial(o2) & -
prior(f,o2) -> (exists o3 exists a (o2 = successor(a,o3) & -
holds(f,o3))))) # label(non_clause).  [assumption].
52 -occurrence_of(x,y) | -arboreal(x) | generator(y).  [clausify(16)].
55 -occurrence_of(x,y) | arboreal(x) | -generator(y).  [clausify(16)].
57 occurrence_of(successor(x,y),x) | -generator(x) | -arboreal(y).
[clausify(17)].
61 prior(x,successor(y,z)) | -holds(x,z) | -generator(y).
```

```
[clausify(31)].
62 -arboreal(successor(x,y)) | generator(x).  [clausify(44)].
179 -activity_occurrence(x) | occurrence_of(x,f1(x)).  [clausify(6)].
180 -occurrence_of(x,y) | -occurrence_of(x,z) | z = y.  [clausify(7)].
181 -arboreal(x) | activity_occurrence(x).  [clausify(8)].
182 -earlier(x,y) | arboreal(x).  [clausify(9)].
201 initial(x) | -arboreal(x) | earlier(f6(x),x).  [clausify(26)].
213 -arboreal(x) | initial(x) | successor(f15(x),f14(x)) = x.
[clausify(43)].
214 -arboreal(successor(x,y)) | arboreal(y).  [clausify(44)].
215 arboreal(c2).  [clausify(45)].
216 -initial(c2).  [clausify(45)].
217 -prior(c1,c2).  [clausify(45)].
218 successor(x,y) != c2 | holds(c1,y).  [clausify(45)].
227 -occurrence_of(x,y) | arboreal(x) | -occurrence_of(z,y) | -
arboreal(z).  [resolve(55,c,52,c)].
249 -arboreal(successor(x,y)) | occurrence_of(successor(x,z),x) | -
arboreal(z).  [resolve(62,b,57,b)].
252 -arboreal(successor(x,y)) | prior(z,successor(x,u)) | -holds(z,u).
[resolve(62,b,61,c)].
296 successor(f15(c2),f14(c2)) = c2.
[resolve(215,a,213,a),unit_del(a,216)].
297 earlier(f6(c2),c2).  [resolve(215,a,201,b),unit_del(a,216)].
300 activity_occurrence(c2).  [resolve(215,a,181,a)].
305 holds(c1,f14(c2)).  [resolve(296,a,218,a)].
308 arboreal(f14(c2)).  [para(296(a,1),214(a,1)),unit_del(a,215)].
323 occurrence_of(successor(f15(c2),x),f15(c2)) | -arboreal(x).
[para(296(a,1),249(a,1)),unit_del(a,215)].
327 occurrence_of(c2,f1(c2)).  [resolve(300,a,179,a)].
358 arboreal(f6(c2)).  [resolve(297,a,182,a)].
372 -arboreal(successor(x,y)) | prior(c1,successor(x,f14(c2))).
[resolve(305,a,252,c)].
384 -occurrence_of(x,f1(c2)) | arboreal(x).
[resolve(327,a,227,c),unit_del(c,215)].
389 -occurrence_of(c2,x) | f1(c2) = x.  [resolve(327,a,180,b)].
553 occurrence_of(successor(f15(c2),f6(c2)),f15(c2)).
[resolve(323,b,358,a)].
554 occurrence_of(c2,f15(c2)).
[resolve(323,b,308,a),rewrite([296(5)])].
675 f15(c2) = f1(c2).  [resolve(389,a,554,a),flip(a)].
677 occurrence_of(successor(f1(c2),f6(c2)),f1(c2)).
[back_rewrite(553),rewrite([675(2),675(7)])].
686 successor(f1(c2),f14(c2)) = c2.
[back_rewrite(296),rewrite([675(2)])].
883 arboreal(successor(f1(c2),f6(c2))).  [resolve(677,a,384,a)].
1432 $F.  [resolve(372,a,883,a),rewrite([686(6)]),unit_del(a,217)].

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

Proposition #01 Proof – With Missing Axiom

This proof corresponds to Ex 4.3.1 (with the missing axiom).

Goal: `all f all s (-changes(s,f) <-> (prior(f,s) <-> holds(f,s))).`

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.12 (+ 0.00) seconds.
% Length of proof is 46.
% Level of proof is 13.
% Maximum clause weight is 14.
% Given clauses 161.

28 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
29 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
35 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
36 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
37 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
41 (all f all s (-changes(s,f) <-> (prior(f,s) <-> holds(f,s)))) #
label(non_clause) # label(goal).  [goal].
64 -holds(x,y) | state(x).  [clausify(28)].
66 -prior(x,y) | state(x).  [clausify(29)].
72 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(35)].
74 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(36)].
75 -changes(x,y) | achieves(x,y) | falsifies(x,y).  [clausify(37)].
77 -falsifies(x,y) | prior(y,x).  [clausify(35)].
78 -falsifies(x,y) | -holds(y,x).  [clausify(35)].
79 changes(x,y) | -falsifies(x,y).  [clausify(37)].
86 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(72,b,66,b)].
87 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(74,b,64,b)].
89 -achieves(x,y) | -prior(y,x).  [clausify(36)].
90 -achieves(x,y) | holds(y,x).  [clausify(36)].
91 changes(x,y) | -achieves(x,y).  [clausify(37)].
97 -changes(x,y) | achieves(x,y) | prior(y,x).  [resolve(75,c,77,a)].
98 -changes(x,y) | achieves(x,y) | -holds(y,x).  [resolve(75,c,78,a)].
129 -holds(x,y) | arboreal(y).  [clausify(28)].
130 -prior(x,y) | arboreal(y).  [clausify(29)].
153 -changes(c2,c1) | -prior(c1,c2) | holds(c1,c2).  [deny(41)].
154 -changes(c2,c1) | prior(c1,c2) | -holds(c1,c2).  [deny(41)].
155 changes(c2,c1) | prior(c1,c2) | holds(c1,c2).  [deny(41)].
```

```
156 changes(c2,c1) | -prior(c1,c2) | -holds(c1,c2).  [deny(41)].
200 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(86,a,79,b)].
201 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(87,a,91,b)].
204 -changes(x,y) | prior(y,x) | holds(y,x).  [resolve(97,b,90,a)].
208 -changes(x,y) | -holds(y,x) | -prior(y,x).  [resolve(98,b,89,a)].
224 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(200,b,d)].
225 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(201,c,d)].
272 prior(c1,c2) | holds(c1,c2).
[resolve(204,a,155,a),merge(c),merge(d)].
275 holds(c1,c2) | -arboreal(c2) | changes(c2,c1).
[resolve(272,a,224,b),merge(c)].
280 holds(c1,c2) | arboreal(c2).  [resolve(272,a,130,a)].
296 arboreal(c2).  [resolve(280,a,129,a),merge(b)].
299 holds(c1,c2) | changes(c2,c1).
[back_unit_del(275),unit_del(b,296)].
331 holds(c1,c2) | -prior(c1,c2).  [resolve(299,b,153,a),merge(c)].
374 holds(c1,c2).  [resolve(331,b,272,a),merge(b)].
375 changes(c2,c1) | -prior(c1,c2).
[back_unit_del(156),unit_del(c,374)].
376 -changes(c2,c1) | prior(c1,c2).
[back_unit_del(154),unit_del(c,374)].
377 prior(c1,c2) | changes(c2,c1).
[resolve(374,a,225,c),unit_del(a,296)].
460 prior(c1,c2).  [resolve(377,b,376,a),merge(b)].
461 changes(c2,c1).  [back_unit_del(375),unit_del(b,460)].
463 $F.  [resolve(461,a,208,a),unit_del(a,374),unit_del(b,460)].

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

Proposition #01 (Revised) Proof – With Missing Axiom

This proof corresponds to Ex 4.3.2 (with the missing axiom).

Goal: `all f all s (state(f) & arboreal(s) -> (-changes(s,f) <-> (prior(f,s) <-> holds(f,s)))).`

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.08 (+ 0.06) seconds.
% Length of proof is 42.
% Level of proof is 11.
% Maximum clause weight is 14.
% Given clauses 160.

28 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
29 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
35 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
36 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
37 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
41 (all f all s (state(f) & arboreal(s) -> (-changes(s,f) <->
(prior(f,s) <-> holds(f,s))))) # label(non_clause) # label(goal).
[goal].
64 -holds(x,y) | state(x).  [clausify(28)].
66 -prior(x,y) | state(x).  [clausify(29)].
72 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(35)].
74 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(36)].
76 -changes(x,y) | achieves(x,y) | falsifies(x,y).  [clausify(37)].
78 -falsifies(x,y) | prior(y,x).  [clausify(35)].
79 -falsifies(x,y) | -holds(y,x).  [clausify(35)].
80 changes(x,y) | -falsifies(x,y).  [clausify(37)].
87 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(72,b,66,b)].
88 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(74,b,64,b)].
90 -achieves(x,y) | -prior(y,x).  [clausify(36)].
91 -achieves(x,y) | holds(y,x).  [clausify(36)].
92 changes(x,y) | -achieves(x,y).  [clausify(37)].
98 -changes(x,y) | achieves(x,y) | prior(y,x).  [resolve(76,c,78,a)].
99 -changes(x,y) | achieves(x,y) | -holds(y,x).  [resolve(76,c,79,a)].
154 arboreal(c2).  [deny(41)].
155 -changes(c2,c1) | -prior(c1,c2) | holds(c1,c2).  [deny(41)].
156 -changes(c2,c1) | prior(c1,c2) | -holds(c1,c2).  [deny(41)].
```

```
157 changes(c2,c1) | prior(c1,c2) | holds(c1,c2).  [deny(41)].
158 changes(c2,c1) | -prior(c1,c2) | -holds(c1,c2).  [deny(41)].
205 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(87,a,80,b)].
206 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(88,a,92,b)].
209 -changes(x,y) | prior(y,x) | holds(y,x).  [resolve(98,b,91,a)].
213 -changes(x,y) | -holds(y,x) | -prior(y,x).  [resolve(99,b,90,a)].
229 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(205,b,d)].
230 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(206,c,d)].
283 prior(c1,c2) | holds(c1,c2).
[resolve(209,a,157,a),merge(c),merge(d)].
324 holds(c1,c2) | changes(c2,c1).
[resolve(283,a,229,b),merge(c),unit_del(b,154)].
343 holds(c1,c2) | -prior(c1,c2).  [resolve(324,b,155,a),merge(c)].
344 holds(c1,c2).  [resolve(343,b,283,a),merge(b)].
345 changes(c2,c1) | -prior(c1,c2).
[back_unit_del(158),unit_del(c,344)].
346 -changes(c2,c1) | prior(c1,c2).
[back_unit_del(156),unit_del(c,344)].
347 prior(c1,c2) | changes(c2,c1).
[resolve(344,a,230,c),unit_del(a,154)].
443 prior(c1,c2).  [resolve(347,b,346,a),merge(b)].
444 changes(c2,c1).  [back_unit_del(345),unit_del(b,443)].
446 $F.  [resolve(444,a,213,a),unit_del(a,344),unit_del(b,443)].

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

Proposition #02 (Case #01) Proof – With Missing Axiom

This proof corresponds to Ex 4.4.6 (with the missing axiom).

Goal:

```
(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & (prior(f,s_1) <-> -prior(f,s_2))
     & earlier(s_1,s_2))
     ->
     (exists s
          (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.23 (+ 0.03) seconds.
% Length of proof is 85.
% Level of proof is 31.
% Maximum clause weight is 25.
% Given clauses 342.

16 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
18 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
19 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
20 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
21 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
22 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
23 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
prior(f,o_1) & -prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & prior(f,o_3) & -
holds(f,o_3))))) # label(non_clause).  [assumption].
24 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
-prior(f,o_1) & prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & -prior(f,o_3) &
holds(f,o_3))))) # label(non_clause).  [assumption].
26 -(all f all s_1 all s_2 (state(f) & arboreal(s_1) & arboreal(s_2) &
(prior(f,s_1) <-> -prior(f,s_2)) & earlier(s_1,s_2) -> (exists s
(changes(s,f) & arboreal(s))))) # label(non_clause).  [assumption].
35 -earlierEq(x,y) | arboreal(y).  [clausify(16)].
39 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | earlierEq(y,f5(x,y,z)).  [clausify(23)].
41 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | earlierEq(y,f6(x,y,z)).  [clausify(24)].
```

```
43 -holds(x,y) | state(x).  [clausify(18)].
45 -prior(x,y) | state(x).  [clausify(19)].
47 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(20)].
49 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(21)].
50 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | prior(x,f5(x,y,z)).  [clausify(23)].
51 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | -holds(x,f5(x,y,z)).  [clausify(23)].
52 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | -prior(x,f6(x,y,z)).  [clausify(24)].
53 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | holds(x,f6(x,y,z)).  [clausify(24)].
55 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | arboreal(f5(x,y,z)).  [resolve(39,g,35,a)].
58 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | arboreal(f6(x,y,z)).  [resolve(41,g,35,a)].
65 changes(x,y) | -falsifies(x,y).  [clausify(22)].
68 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(47,b,45,b)].
69 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(49,b,43,b)].
73 changes(x,y) | -achieves(x,y).  [clausify(22)].
97 arboreal(c2).  [clausify(26)].
98 arboreal(c3).  [clausify(26)].
99 -prior(c1,c2) | -prior(c1,c3).  [clausify(26)].
100 prior(c1,c2) | prior(c1,c3).  [clausify(26)].
101 earlier(c2,c3).  [clausify(26)].
102 -changes(x,c1) | -arboreal(x).  [clausify(26)].
115 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)) | -prior(z,u).
[resolve(50,a,45,b)].
116 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)) | -holds(z,u).
[resolve(51,a,43,b)].
118 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)) | -prior(z,u).
[resolve(52,a,45,b)].
120 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)) | -prior(z,u).
[resolve(53,a,45,b)].
123 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | arboreal(f5(z,x,y)) | -prior(z,u).
[resolve(55,a,45,b)].
131 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)) | -prior(z,u).
[resolve(58,a,45,b)].
139 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(68,a,65,b)].
140 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(69,a,73,b)].
```

```
150 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)).  [factor(115,c,g)].
151 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)).  [factor(116,f,g)].
152 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)).  [factor(118,d,g)].
153 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)).  [factor(120,d,g)].
154 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | arboreal(f5(z,x,y)).  [factor(123,c,g)].
157 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)).  [factor(131,d,g)].
160 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(139,b,d)].
161 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(140,c,d)].
177 -prior(x,c2) | prior(x,c3) | prior(x,f5(x,c2,c3)).
[resolve(150,e,101,a),unit_del(a,97),unit_del(b,98)].
178 prior(x,c2) | -prior(x,c3) | -prior(x,f6(x,c2,c3)).
[resolve(152,e,101,a),unit_del(a,97),unit_del(b,98)].
179 prior(x,c2) | -prior(x,c3) | holds(x,f6(x,c2,c3)).
[resolve(153,e,101,a),unit_del(a,97),unit_del(b,98)].
180 -prior(x,c2) | prior(x,c3) | arboreal(f5(x,c2,c3)).
[resolve(154,e,101,a),unit_del(a,97),unit_del(b,98)].
183 prior(x,c2) | -prior(x,c3) | arboreal(f6(x,c2,c3)).
[resolve(157,e,101,a),unit_del(a,97),unit_del(b,98)].
285 prior(c1,c2) | arboreal(f6(c1,c2,c3)).
[resolve(183,b,100,b),merge(c)].
288 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | arboreal(f5(c1,c2,c3)).
[resolve(285,a,180,a)].
289 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | prior(c1,f5(c1,c2,c3)).
[resolve(285,a,177,a)].
291 prior(c1,c2) | holds(c1,f6(c1,c2,c3)).
[resolve(179,b,100,b),merge(c)].
292 prior(c1,c2) | -arboreal(f6(c1,c2,c3)) | prior(c1,f6(c1,c2,c3)) |
changes(f6(c1,c2,c3),c1).  [resolve(291,b,161,c)].
330 arboreal(f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)) | -prior(c1,c2).
[resolve(288,b,99,b)].
331 arboreal(f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)).
[resolve(330,c,285,a),merge(c)].
339 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | -arboreal(f5(c1,c2,c3)) |
holds(c1,f5(c1,c2,c3)) | changes(f5(c1,c2,c3),c1).
[resolve(289,c,160,b)].
488 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | changes(f6(c1,c2,c3),c1) |
arboreal(f5(c1,c2,c3)).  [resolve(292,b,331,a)].
552 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)) | -
arboreal(f6(c1,c2,c3)).  [resolve(488,c,102,a)].
571 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | arboreal(f5(c1,c2,c3)).
[resolve(552,d,331,a),merge(d)].
572 prior(c1,c2) | arboreal(f5(c1,c2,c3)) | -prior(c1,c3).
[resolve(571,b,178,c),merge(c)].
573 prior(c1,c2) | arboreal(f5(c1,c2,c3)).
```

```
[resolve(572,c,100,b),merge(c)].
580 arboreal(f5(c1,c2,c3)) | prior(c1,c3).
[resolve(573,a,180,a),merge(c)].
592 arboreal(f5(c1,c2,c3)) | -prior(c1,c2).  [resolve(580,b,99,b)].
593 arboreal(f5(c1,c2,c3)).  [resolve(592,b,573,a),merge(b)].
616 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | holds(c1,f5(c1,c2,c3)) |
changes(f5(c1,c2,c3),c1).  [back_unit_del(339),unit_del(c,593)].
625 -changes(f5(c1,c2,c3),c1).  [ur(102,b,593,a)].
626 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | holds(c1,f5(c1,c2,c3)).
[back_unit_del(616),unit_del(d,625)].
642 arboreal(f6(c1,c2,c3)) | prior(c1,c3) | -prior(c1,c2).
[resolve(626,c,151,f),merge(f),unit_del(c,97),unit_del(d,98),unit_del(
f,101)].
647 arboreal(f6(c1,c2,c3)) | prior(c1,c3).
[resolve(642,c,285,a),merge(c)].
652 arboreal(f6(c1,c2,c3)) | -prior(c1,c2).  [resolve(647,b,99,b)].
659 arboreal(f6(c1,c2,c3)).  [resolve(652,b,285,a),merge(b)].
683 prior(c1,c2) | prior(c1,f6(c1,c2,c3)) | changes(f6(c1,c2,c3),c1).
[back_unit_del(292),unit_del(b,659)].
685 -changes(f6(c1,c2,c3),c1).  [ur(102,b,659,a)].
686 prior(c1,c2) | prior(c1,f6(c1,c2,c3)).
[back_unit_del(683),unit_del(c,685)].
694 prior(c1,c2) | -prior(c1,c3).  [resolve(686,b,178,c),merge(b)].
699 prior(c1,c2).  [resolve(694,b,100,b),merge(b)].
700 -prior(c1,c3).  [back_unit_del(99),unit_del(a,699)].
719 prior(c1,f5(c1,c2,c3)).  [resolve(699,a,177,a),unit_del(a,700)].
721 holds(c1,f5(c1,c2,c3)).
[resolve(719,a,160,b),unit_del(a,593),unit_del(c,625)].
723 $F.
[ur(151,a,97,a,b,98,a,c,699,a,d,700,a,e,101,a),unit_del(a,721)].

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

Proposition #02 (Case #02) Proof – With Missing Axiom

This proof corresponds to Ex 4.4.7 (with the missing axiom).

Goal:

```
(all f all s_1 all s_2
     ((state(f)
     & arboreal(s_1)
     & arboreal(s_2)
     & (prior(f,s_1) <-> -prior(f,s_2))
     & earlier(s_2,s_1))
     ->
     (exists s
          (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.28 (+ 0.03) seconds.
% Length of proof is 75.
% Level of proof is 24.
% Maximum clause weight is 25.
% Given clauses 393.

16 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
18 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
19 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
20 (all f all o (falsifies(o,f) <-> state(f) & arboreal(o) &
prior(f,o) & -holds(f,o))) # label(non_clause).  [assumption].
21 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
22 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
23 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
prior(f,o_1) & -prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & prior(f,o_3) & -
holds(f,o_3))))) # label(non_clause).  [assumption].
24 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
-prior(f,o_1) & prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & -prior(f,o_3) &
holds(f,o_3))))) # label(non_clause).  [assumption].
26 -(all f all s_1 all s_2 (state(f) & arboreal(s_1) & arboreal(s_2) &
(prior(f,s_1) <-> -prior(f,s_2)) & earlier(s_2,s_1) -> (exists s
(changes(s,f) & arboreal(s))))) # label(non_clause).  [assumption].
35 -earlierEq(x,y) | arboreal(y).  [clausify(16)].
41 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | earlierEq(y,f6(x,y,z)).  [clausify(24)].
43 -holds(x,y) | state(x).  [clausify(18)].
45 -prior(x,y) | state(x).  [clausify(19)].
```

```
47 falsifies(x,y) | -state(y) | -arboreal(x) | -prior(y,x) |
holds(y,x).  [clausify(20)].
49 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(21)].
50 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | prior(x,f5(x,y,z)).  [clausify(23)].
51 -state(x) | -arboreal(y) | -arboreal(z) | -prior(x,y) | prior(x,z)
| -earlier(y,z) | -holds(x,f5(x,y,z)).  [clausify(23)].
52 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | -prior(x,f6(x,y,z)).  [clausify(24)].
53 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | holds(x,f6(x,y,z)).  [clausify(24)].
58 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | arboreal(f6(x,y,z)).  [resolve(41,g,35,a)].
65 changes(x,y) | -falsifies(x,y).  [clausify(22)].
68 falsifies(x,y) | -arboreal(x) | -prior(y,x) | holds(y,x) | -
prior(y,z).  [resolve(47,b,45,b)].
69 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(49,b,43,b)].
73 changes(x,y) | -achieves(x,y).  [clausify(22)].
95 -prior(x,y) | arboreal(y).  [clausify(19)].
97 arboreal(c2).  [clausify(26)].
98 arboreal(c3).  [clausify(26)].
99 -prior(c1,c2) | -prior(c1,c3).  [clausify(26)].
100 prior(c1,c2) | prior(c1,c3).  [clausify(26)].
101 earlier(c3,c2).  [clausify(26)].
102 -changes(x,c1) | -arboreal(x).  [clausify(26)].
115 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)) | -prior(z,u).
[resolve(50,a,45,b)].
116 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)) | -holds(z,u).
[resolve(51,a,43,b)].
118 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)) | -prior(z,u).
[resolve(52,a,45,b)].
120 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)) | -prior(z,u).
[resolve(53,a,45,b)].
131 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)) | -prior(z,u).
[resolve(58,a,45,b)].
139 -arboreal(x) | -prior(y,x) | holds(y,x) | -prior(y,z) |
changes(x,y).  [resolve(68,a,65,b)].
140 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(69,a,73,b)].
150 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | prior(z,f5(z,x,y)).  [factor(115,c,g)].
151 -arboreal(x) | -arboreal(y) | -prior(z,x) | prior(z,y) | -
earlier(x,y) | -holds(z,f5(z,x,y)).  [factor(116,f,g)].
152 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f6(z,x,y)).  [factor(118,d,g)].
```

```
153 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f6(z,x,y)).  [factor(120,d,g)].
157 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f6(z,x,y)).  [factor(131,d,g)].
160 -arboreal(x) | -prior(y,x) | holds(y,x) | changes(x,y).
[factor(139,b,d)].
161 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(140,c,d)].
177 -prior(x,c3) | prior(x,c2) | prior(x,f5(x,c3,c2)).
[resolve(150,e,101,a),unit_del(a,98),unit_del(b,97)].
178 prior(x,c3) | -prior(x,c2) | -prior(x,f6(x,c3,c2)).
[resolve(152,e,101,a),unit_del(a,98),unit_del(b,97)].
179 prior(x,c3) | -prior(x,c2) | holds(x,f6(x,c3,c2)).
[resolve(153,e,101,a),unit_del(a,98),unit_del(b,97)].
183 prior(x,c3) | -prior(x,c2) | arboreal(f6(x,c3,c2)).
[resolve(157,e,101,a),unit_del(a,98),unit_del(b,97)].
282 prior(c1,c2) | prior(c1,f5(c1,c3,c2)).
[resolve(177,a,100,b),merge(c)].
286 prior(c1,c2) | -arboreal(f5(c1,c3,c2)) | holds(c1,f5(c1,c3,c2)) |
changes(f5(c1,c3,c2),c1).  [resolve(282,b,160,b)].
287 prior(c1,c2) | arboreal(f5(c1,c3,c2)).  [resolve(282,b,95,a)].
291 prior(c1,c3) | arboreal(f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)).
[resolve(183,b,287,a)].
292 prior(c1,c3) | holds(c1,f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)).
[resolve(179,b,287,a)].
330 arboreal(f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)) | -prior(c1,c2).
[resolve(291,a,99,b)].
331 arboreal(f6(c1,c3,c2)) | arboreal(f5(c1,c3,c2)).
[resolve(330,c,287,a),merge(c)].
339 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | -arboreal(f6(c1,c3,c2)) |
prior(c1,f6(c1,c3,c2)) | changes(f6(c1,c3,c2),c1).
[resolve(292,b,161,c)].
705 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | prior(c1,f6(c1,c3,c2)) |
changes(f6(c1,c3,c2),c1).  [resolve(339,c,331,a),merge(e)].
715 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | prior(c1,f6(c1,c3,c2)) | -
arboreal(f6(c1,c3,c2)).  [resolve(705,d,102,a)].
753 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | prior(c1,f6(c1,c3,c2)).
[resolve(715,d,331,a),merge(d)].
754 prior(c1,c3) | arboreal(f5(c1,c3,c2)) | -prior(c1,c2).
[resolve(753,c,178,c),merge(c)].
759 prior(c1,c3) | arboreal(f5(c1,c3,c2)).
[resolve(754,c,287,a),merge(c)].
765 arboreal(f5(c1,c3,c2)) | -prior(c1,c2).  [resolve(759,a,99,b)].
770 arboreal(f5(c1,c3,c2)).  [resolve(765,b,287,a),merge(b)].
802 prior(c1,c2) | holds(c1,f5(c1,c3,c2)) | changes(f5(c1,c3,c2),c1).
[back_unit_del(286),unit_del(b,770)].
804 -changes(f5(c1,c3,c2),c1).  [ur(102,b,770,a)].
805 prior(c1,c2) | holds(c1,f5(c1,c3,c2)).
[back_unit_del(802),unit_del(c,804)].
839 prior(c1,c2) | -prior(c1,c3).
[resolve(805,b,151,f),merge(e),unit_del(b,98),unit_del(c,97),unit_del(
e,101)].
```

```
840 prior(c1,c2).  [resolve(839,b,100,b),merge(b)].
842 -prior(c1,c3).  [back_unit_del(99),unit_del(a,840)].
861 arboreal(f6(c1,c3,c2)).  [resolve(840,a,183,b),unit_del(a,842)].
862 holds(c1,f6(c1,c3,c2)).  [resolve(840,a,179,b),unit_del(a,842)].
864 -prior(c1,f6(c1,c3,c2)).  [ur(178,a,842,a,b,840,a)].
869 -changes(f6(c1,c3,c2),c1).  [ur(102,b,861,a)].
877 $F.
[resolve(862,a,161,c),unit_del(a,861),unit_del(b,864),unit_del(c,869)]
.

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

Proposition #02 (Case #03a) Proof – With Missing Axiom

This proof corresponds to Ex 4.4.15 (with the missing axiom).

Goal:

```
(all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlier(s_3,s_1)
      & -prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 1.89 (+ 0.19) seconds.
% Length of proof is 42.
% Level of proof is 9.
% Maximum clause weight is 22.
% Given clauses 2209.

20 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
21 (all s (initial(s) <-> arboreal(s) & -(exists sp earlier(sp,s)))) #
label(non_clause).  [assumption].
23 (all f all occ (holds(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
24 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
28 (all f all o (achieves(o,f) <-> state(f) & arboreal(o) & -
prior(f,o) & holds(f,o))) # label(non_clause).  [assumption].
29 (all f all o (changes(o,f) <-> achieves(o,f) | falsifies(o,f))) #
label(non_clause).  [assumption].
31 (all f all o_1 all o_2 (state(f) & arboreal(o_1) & arboreal(o_2) &
-prior(f,o_1) & prior(f,o_2) & earlier(o_1,o_2) -> (exists o_3
(earlierEq(o_1,o_3) & earlierEq(o_3,o_2) & -prior(f,o_3) &
holds(f,o_3))))) # label(non_clause).  [assumption].
35 -(all f all s_1 all s_2 all s_3 (state(f) & arboreal(s_1) &
arboreal(s_2) & prior(f,s_1) & -prior(f,s_2) & initial(s_3) &
earlier(s_3,s_1) & -prior(f,s_3) & -earlier(s_1,s_2) & -
earlier(s_2,s_1) -> (exists s (changes(s,f) & arboreal(s))))) #
label(non_clause).  [assumption].
46 -earlierEq(x,y) | arboreal(y).  [clausify(20)].
52 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
```

```
| -earlier(y,z) | earlierEq(y,f8(x,y,z)).  [clausify(31)].
55 -holds(x,y) | state(x).  [clausify(23)].
57 -prior(x,y) | state(x).  [clausify(24)].
61 achieves(x,y) | -state(y) | -arboreal(x) | prior(y,x) | -
holds(y,x).  [clausify(28)].
64 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | -prior(x,f8(x,y,z)).  [clausify(31)].
65 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | holds(x,f8(x,y,z)).  [clausify(31)].
75 -state(x) | -arboreal(y) | -arboreal(z) | prior(x,y) | -prior(x,z)
| -earlier(y,z) | arboreal(f8(x,y,z)).  [resolve(52,g,46,a)].
87 achieves(x,y) | -arboreal(x) | prior(y,x) | -holds(y,x) | -
holds(y,z).  [resolve(61,b,55,b)].
91 changes(x,y) | -achieves(x,y).  [clausify(29)].
119 -initial(x) | arboreal(x).  [clausify(21)].
131 arboreal(c2).  [clausify(35)].
133 prior(c1,c2).  [clausify(35)].
135 initial(c4).  [clausify(35)].
136 earlier(c4,c2).  [clausify(35)].
137 -prior(c1,c4).  [clausify(35)].
140 -changes(x,c1) | -arboreal(x).  [clausify(35)].
166 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f8(z,x,y)) | -prior(z,u).
[resolve(64,a,57,b)].
168 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f8(z,x,y)) | -prior(z,u).
[resolve(65,a,57,b)].
189 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f8(z,x,y)) | -prior(z,u).
[resolve(75,a,57,b)].
200 -arboreal(x) | prior(y,x) | -holds(y,x) | -holds(y,z) |
changes(x,y).  [resolve(87,a,91,b)].
213 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | -prior(z,f8(z,x,y)).  [factor(166,d,g)].
214 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | holds(z,f8(z,x,y)).  [factor(168,d,g)].
223 -arboreal(x) | -arboreal(y) | prior(z,x) | -prior(z,y) | -
earlier(x,y) | arboreal(f8(z,x,y)).  [factor(189,d,g)].
228 -arboreal(x) | prior(y,x) | -holds(y,x) | changes(x,y).
[factor(200,c,d)].
236 arboreal(c4).  [resolve(135,a,119,a)].
254 prior(x,c4) | -prior(x,c2) | -prior(x,f8(x,c4,c2)).
[resolve(213,e,136,a),unit_del(a,236),unit_del(b,131)].
255 prior(x,c4) | -prior(x,c2) | holds(x,f8(x,c4,c2)).
[resolve(214,e,136,a),unit_del(a,236),unit_del(b,131)].
264 prior(x,c4) | -prior(x,c2) | arboreal(f8(x,c4,c2)).
[resolve(223,e,136,a),unit_del(a,236),unit_del(b,131)].
590 holds(c1,f8(c1,c4,c2)).  [resolve(255,b,133,a),unit_del(a,137)].
693 arboreal(f8(c1,c4,c2)).  [resolve(264,b,133,a),unit_del(a,137)].
1118 prior(c1,f8(c1,c4,c2)) | changes(f8(c1,c4,c2),c1).
[resolve(590,a,228,c),unit_del(a,693)].
16887 prior(c1,f8(c1,c4,c2)).
```

```
[resolve(1118,b,140,a),unit_del(b,693)].
16888 $F.  [resolve(16887,a,254,c),unit_del(a,137),unit_del(b,133)].

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

Proposition #02 (Case #03b) Proof – With Missing Axiom

This proof corresponds to Ex 4.4.16 (with the missing axiom).

Goal:

```
((all f all s_1 all s_2 all s_3
        ((state(f)
        & arboreal(s_1)
        & arboreal(s_2)
        & prior(f,s_1) & -prior(f,s_2)
      & initial(s_3)
      & earlier(s_3,s_1)
      & prior(f,s_3)
        & -earlier(s_1,s_2)
        & -earlier(s_2,s_1))
        ->
        (exists s
                (changes(s,f) & arboreal(s))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 28.97 (+ 1.03) seconds.
% Length of proof is 70.
% Level of proof is 12.
% Maximum clause weight is 20.
% Given clauses 11584.

6 (all occ (activity_occurrence(occ) -> (exists a (activity(a) &
occurrence_of(occ,a))))) # label(non_clause).  [assumption].
8 (all s (arboreal(s) -> activity_occurrence(s))) # label(non_clause).
[assumption].
13 (all s1 all s2 (earlier(s1,s2) -> (exists sp (initial(sp) &
earlierEq(sp,s1))))) # label(non_clause).  [assumption].
16 (all s all a (occurrence_of(s,a) -> (arboreal(s) <->
generator(a)))) # label(non_clause).  [assumption].
19 (all a all s1 all s2 (generator(a) -> (earlier(s1,successor(a,s2))
<-> earlierEq(s1,s2)))) # label(non_clause).  [assumption].
20 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
21 (all s (initial(s) <-> arboreal(s) & -(exists sp earlier(sp,s)))) #
label(non_clause).  [assumption].
24 (all f all occ (prior(f,occ) -> state(f) & arboreal(occ))) #
label(non_clause).  [assumption].
25 (all occ1 all occ2 all f (initial(occ1) & initial(occ2) ->
(prior(f,occ1) <-> prior(f,occ2)))) # label(non_clause).
[assumption].
32 (all s1 (arboreal(s1) -> (exists s2 (initial(s2) &
earlierEq(s2,s1))))) # label(non_clause).  [assumption].
33 (all f all s_1 all s_2 (state(f) & prior(f,s_1) & arboreal(s_2) & -
prior(f,s_2) & earlier(s_1,s_2) -> (exists s (changes(s,f) &
arboreal(s))))) # label(non_clause).  [assumption].
```

```
35 -(all f all s_1 all s_2 all s_3 (state(f) & arboreal(s_1) &
arboreal(s_2) & prior(f,s_1) & -prior(f,s_2) & initial(s_3) &
earlier(s_3,s_1) & prior(f,s_3) & -earlier(s_1,s_2) & -
earlier(s_2,s_1) -> (exists s (changes(s,f) & arboreal(s))))) #
label(non_clause).  [assumption].
42 -generator(x) | earlier(y,successor(x,z)) | -earlierEq(y,z).
[clausify(19)].
43 -earlier(x,y) | earlierEq(f2(x,y),x).  [clausify(13)].
45 -earlierEq(x,y) | arboreal(x).  [clausify(20)].
47 -earlierEq(x,y) | earlier(x,y) | y = x.  [clausify(20)].
54 -arboreal(x) | earlierEq(f9(x),x).  [clausify(32)].
57 -prior(x,y) | state(x).  [clausify(24)].
66 -state(x) | -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z)
| changes(f10(x,y,z),x).  [clausify(33)].
67 -state(x) | -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z)
| arboreal(f10(x,y,z)).  [clausify(33)].
101 -activity_occurrence(x) | occurrence_of(x,f1(x)).  [clausify(6)].
103 -arboreal(x) | activity_occurrence(x).  [clausify(8)].
113 -occurrence_of(x,y) | -arboreal(x) | generator(y).
[clausify(16)].
120 -initial(x) | -earlier(y,x).  [clausify(21)].
124 -initial(x) | -initial(y) | -prior(z,x) | prior(z,y).
[clausify(25)].
128 -arboreal(x) | initial(f9(x)).  [clausify(32)].
132 arboreal(c3).  [clausify(35)].
134 -prior(c1,c3).  [clausify(35)].
135 initial(c4).  [clausify(35)].
137 prior(c1,c4).  [clausify(35)].
140 -changes(x,c1) | -arboreal(x).  [clausify(35)].
149 arboreal(f2(x,y)) | -earlier(x,y).  [resolve(45,a,43,b)].
151 earlier(f2(x,y),x) | x = f2(x,y) | -earlier(x,y).
[resolve(47,a,43,b)].
152 earlier(f2(x,y),x) | f2(x,y) = x | -earlier(x,y).
[copy(151),flip(b)].
156 -arboreal(x) | -generator(y) | earlier(f9(x),successor(y,x)).
[resolve(54,b,42,c)].
157 -arboreal(x) | arboreal(f9(x)).  [resolve(54,b,45,a)].
158 -arboreal(x) | earlier(f9(x),x) | x = f9(x).
[resolve(54,b,47,a)].
159 -arboreal(x) | earlier(f9(x),x) | f9(x) = x.  [copy(158),flip(c)].
170 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
changes(f10(x,y,z),x) | -prior(x,u).  [resolve(66,a,57,b)].
172 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
arboreal(f10(x,y,z)) | -prior(x,u).  [resolve(67,a,57,b)].
215 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
changes(f10(x,y,z),x).  [factor(170,a,f)].
216 -prior(x,y) | -arboreal(z) | prior(x,z) | -earlier(y,z) |
arboreal(f10(x,y,z)).  [factor(172,a,f)].
232 initial(f9(c3)).  [resolve(132,a,128,a)].
234 activity_occurrence(c3).  [resolve(132,a,103,a)].
245 -initial(x) | prior(c1,x).
[resolve(137,a,124,c),unit_del(a,135)].
```

```
250 arboreal(f9(c3)).  [resolve(157,a,132,a)].
252 earlier(f9(c3),c3) | f9(c3) = c3.  [resolve(159,a,132,a)].
275 occurrence_of(c3,f1(c3)).  [resolve(234,a,101,a)].
287 activity_occurrence(f9(c3)).  [resolve(250,a,103,a)].
323 occurrence_of(f9(c3),f1(f9(c3))).  [resolve(287,a,101,a)].
404 generator(f1(c3)).  [resolve(275,a,113,a),unit_del(a,132)].
408 -arboreal(x) | earlier(f9(x),successor(f1(c3),x)).
[resolve(404,a,156,b)].
441 prior(c1,f9(c3)).  [resolve(245,a,232,a)].
529 f9(c3) = c3 | -prior(x,f9(c3)) | prior(x,c3) |
arboreal(f10(x,f9(c3),c3)).  [resolve(252,a,216,d),unit_del(c,132)].
530 f9(c3) = c3 | -prior(x,f9(c3)) | prior(x,c3) |
changes(f10(x,f9(c3),c3),x).  [resolve(252,a,215,d),unit_del(c,132)].
726 generator(f1(f9(c3))).  [resolve(323,a,113,a),unit_del(a,250)].
731 -arboreal(x) | earlier(f9(x),successor(f1(f9(c3)),x)).
[resolve(726,a,156,b)].
2661 earlier(f9(c3),successor(f1(c3),c3)).  [resolve(408,a,132,a)].
5029 f9(c3) = c3 | arboreal(f10(c1,f9(c3),c3)).
[resolve(529,b,441,a),unit_del(b,134)].
5067 earlier(f2(f9(c3),successor(f1(c3),c3)),f9(c3)) |
f2(f9(c3),successor(f1(c3),c3)) = f9(c3).  [resolve(2661,a,152,c)].
5077 f9(c3) = c3 | changes(f10(c1,f9(c3),c3),c1).
[resolve(530,b,441,a),unit_del(b,134)].
8195 earlier(f9(c3),successor(f1(f9(c3)),c3)).
[resolve(731,a,132,a)].
14102 arboreal(f2(f9(c3),successor(f1(f9(c3)),c3))).
[resolve(8195,a,149,b)].
25413 initial(f9(f2(f9(c3),successor(f1(f9(c3)),c3)))).
[resolve(14102,a,128,a)].
36440 prior(c1,f9(f2(f9(c3),successor(f1(f9(c3)),c3)))).
[resolve(25413,a,245,a)].
41807 f2(f9(c3),successor(f1(c3),c3)) = f9(c3).
[resolve(5067,a,120,b),unit_del(b,232)].
41854 f9(c3) = c3 | -arboreal(f10(c1,f9(c3),c3)).
[resolve(5077,b,140,a)].
42443 f9(c3) = c3.  [resolve(41854,b,5029,b),merge(b)].
42457 f2(c3,successor(f1(c3),c3)) = c3.
[back_rewrite(41807),rewrite([42443(2),42443(8)])].
42490 $F.
[back_rewrite(36440),rewrite([42443(3),42443(4),42457(7),42443(3)]),un
it_del(a,134)].

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

Inertia Principle (Part #01) Proof – With Missing Axiom

This proof corresponds to Ex 4.5.1.4 (with the missing axiom).

Goal:

```
(all f all o_1 all o
    ((state(f)
& arboreal(o_1)
& holds(f,o_1)
& -prior(f,o)
& earlier(o,o_1))
    ->
    (exists o_3
(earlierEq(o_3,o_1)
& -prior(f,o_3)
& holds(f,o_3))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.14 (+ 0.06) seconds.
% Length of proof is 28.
% Level of proof is 9.
% Maximum clause weight is 20.
% Given clauses 436.

24 (all s1 all s2 (earlierEq(s1,s2) <-> arboreal(s1) & arboreal(s2) &
(earlier(s1,s2) | s1 = s2))) # label(non_clause).  [assumption].
38 (all f all o (always(f,o) <-> (all o1 (earlier(o1,o) ->
prior(f,o1))))) # label(non_clause).  [assumption].
39 (all f all o1 (holds(f,o1) -> always(f,o1) | (exists o2
(earlierEq(o2,o1) & -prior(f,o2) & (all o3 (earlierEq(o2,o3) &
earlierEq(o3,o1) -> holds(f,o3)))))))) # label(non_clause).
[assumption].
41 -(all f all o_1 all o (state(f) & arboreal(o_1) & holds(f,o_1) & -
prior(f,o) & earlier(o,o_1) -> (exists o_3 (earlierEq(o_3,o_1) & -
prior(f,o_3) & holds(f,o_3))))) # label(non_clause).  [assumption].
101 -always(x,y) | -earlier(z,y) | prior(x,z).  [clausify(38)].
103 -holds(x,y) | always(x,y) | earlierEq(f11(x,y),y).
[clausify(39)].
104 -holds(x,y) | always(x,y) | -prior(x,f11(x,y)).  [clausify(39)].
105 -holds(x,y) | always(x,y) | -earlierEq(f11(x,y),z) | -
earlierEq(z,y) | holds(x,z).  [clausify(39)].
146 -earlierEq(x,y) | arboreal(x).  [clausify(24)].
150 earlierEq(x,y) | -arboreal(x) | -arboreal(y) | y != x.
[clausify(24)].
167 holds(c1,c2).  [clausify(41)].
168 -prior(c1,c3).  [clausify(41)].
169 earlier(c3,c2).  [clausify(41)].
170 -earlierEq(x,c2) | prior(c1,x) | -holds(c1,x).  [clausify(41)].
218 -holds(x,y) | earlierEq(f11(x,y),y) | -earlier(z,y) | prior(x,z).
[resolve(103,b,101,a)].
```

```
219 -holds(x,y) | -prior(x,f11(x,y)) | -earlier(z,y) | prior(x,z).
[resolve(104,b,101,a)].
220 -holds(x,y) | -earlierEq(f11(x,y),z) | -earlierEq(z,y) |
holds(x,z) | -earlier(u,y) | prior(x,u).  [resolve(105,b,101,a)].
231 earlierEq(x,x) | -arboreal(x).  [factor(150,b,c),xx(c)].
263 -holds(x,c2) | earlierEq(f11(x,c2),c2) | prior(x,c3).
[resolve(218,c,169,a)].
264 -holds(x,c2) | -prior(x,f11(x,c2)) | prior(x,c3).
[resolve(219,c,169,a)].
265 -holds(x,c2) | -earlierEq(f11(x,c2),y) | -earlierEq(y,c2) |
holds(x,y) | prior(x,c3).  [resolve(220,e,169,a)].
604 earlierEq(f11(c1,c2),c2).  [resolve(263,a,167,a),unit_del(b,168)].
608 prior(c1,f11(c1,c2)) | -holds(c1,f11(c1,c2)).
[resolve(604,a,170,a)].
611 arboreal(f11(c1,c2)).  [resolve(604,a,146,a)].
613 earlierEq(f11(c1,c2),f11(c1,c2)).  [resolve(611,a,231,b)].
1524 holds(c1,f11(c1,c2)).
[resolve(613,a,265,b),unit_del(a,167),unit_del(b,604),unit_del(d,168)]
.
1527 prior(c1,f11(c1,c2)).  [back_unit_del(608),unit_del(b,1524)].
1542 $F.  [resolve(1527,a,264,b),unit_del(a,167),unit_del(b,168)].

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

Inertia Principle (Part #02) Proof – With Missing Axiom

This proof corresponds to Ex 4.5.2.12 (with the missing axiom).

Goal:

```
(all f all o2
     ((state(f)
     & arboreal(o2)
     & -initial(o2)
     & -prior(f,o2))
     ->
     (exists o3 exists a
          ((o2 = successor(a,o3))
          & -holds(f,o3))))).
```

```
============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.20 (+ 0.05) seconds.
% Length of proof is 48.
% Level of proof is 8.
% Maximum clause weight is 12.
% Given clauses 353.

6 (all occ (activity_occurrence(occ) -> (exists a (activity(a) &
occurrence_of(occ,a))))) # label(non_clause).  [assumption].
7 (all o all a1 all a2 (occurrence_of(o,a1) & occurrence_of(o,a2) ->
a1 = a2)) # label(non_clause).  [assumption].
8 (all s (arboreal(s) -> activity_occurrence(s))) # label(non_clause).
[assumption].
9 (all s1 all s2 (earlier(s1,s2) -> arboreal(s1) & arboreal(s2))) #
label(non_clause).  [assumption].
16 (all s all a (occurrence_of(s,a) -> (arboreal(s) <->
generator(a)))) # label(non_clause).  [assumption].
17 (all a all o (occurrence_of(successor(a,o),a) <-> generator(a) &
arboreal(o))) # label(non_clause).  [assumption].
26 (all s (initial(s) <-> arboreal(s) & -(exists sp earlier(sp,s)))) #
label(non_clause).  [assumption].
31 (all a all occ all f (prior(f,successor(a,occ)) <-> holds(f,occ) &
generator(a))) # label(non_clause).  [assumption].
43 (all o2 (arboreal(o2) & -initial(o2) -> (exists o3 exists a o2 =
successor(a,o3)))) # label(non_clause).  [assumption].
44 (all a all o (arboreal(successor(a,o)) -> generator(a) &
arboreal(o))) # label(non_clause).  [assumption].
46 -(all f all o2 (state(f) & arboreal(o2) & -initial(o2) & -
prior(f,o2) -> (exists o3 exists a (o2 = successor(a,o3) & -
holds(f,o3))))) # label(non_clause).  [assumption].
53 -occurrence_of(x,y) | -arboreal(x) | generator(y).  [clausify(16)].
56 -occurrence_of(x,y) | arboreal(x) | -generator(y).  [clausify(16)].
58 occurrence_of(successor(x,y),x) | -generator(x) | -arboreal(y).
[clausify(17)].
62 prior(x,successor(y,z)) | -holds(x,z) | -generator(y).
```

```
[clausify(31)].
63 -arboreal(successor(x,y)) | generator(x).  [clausify(44)].
181 -activity_occurrence(x) | occurrence_of(x,f1(x)).  [clausify(6)].
182 -occurrence_of(x,y) | -occurrence_of(x,z) | z = y.  [clausify(7)].
183 -arboreal(x) | activity_occurrence(x).  [clausify(8)].
184 -earlier(x,y) | arboreal(x).  [clausify(9)].
203 initial(x) | -arboreal(x) | earlier(f6(x),x).  [clausify(26)].
215 -arboreal(x) | initial(x) | successor(f15(x),f14(x)) = x.
[clausify(43)].
216 -arboreal(successor(x,y)) | arboreal(y).  [clausify(44)].
217 arboreal(c2).  [clausify(46)].
218 -initial(c2).  [clausify(46)].
219 -prior(c1,c2).  [clausify(46)].
220 successor(x,y) != c2 | holds(c1,y).  [clausify(46)].
229 -occurrence_of(x,y) | arboreal(x) | -occurrence_of(z,y) | -
arboreal(z).  [resolve(56,c,53,c)].
251 -arboreal(successor(x,y)) | occurrence_of(successor(x,z),x) | -
arboreal(z).  [resolve(63,b,58,b)].
254 -arboreal(successor(x,y)) | prior(z,successor(x,u)) | -holds(z,u).
[resolve(63,b,62,c)].
298 successor(f15(c2),f14(c2)) = c2.
[resolve(217,a,215,a),unit_del(a,218)].
299 earlier(f6(c2),c2).  [resolve(217,a,203,b),unit_del(a,218)].
302 activity_occurrence(c2).  [resolve(217,a,183,a)].
307 holds(c1,f14(c2)).  [resolve(298,a,220,a)].
310 arboreal(f14(c2)).  [para(298(a,1),216(a,1)),unit_del(a,217)].
325 occurrence_of(successor(f15(c2),x),f15(c2)) | -arboreal(x).
[para(298(a,1),251(a,1)),unit_del(a,217)].
329 occurrence_of(c2,f1(c2)).  [resolve(302,a,181,a)].
360 arboreal(f6(c2)).  [resolve(299,a,184,a)].
374 -arboreal(successor(x,y)) | prior(c1,successor(x,f14(c2))).
[resolve(307,a,254,c)].
386 -occurrence_of(x,f1(c2)) | arboreal(x).
[resolve(329,a,229,c),unit_del(c,217)].
391 -occurrence_of(c2,x) | f1(c2) = x.  [resolve(329,a,182,b)].
555 occurrence_of(successor(f15(c2),f6(c2)),f15(c2)).
[resolve(325,b,360,a)].
556 occurrence_of(c2,f15(c2)).
[resolve(325,b,310,a),rewrite([298(5)])].
677 f15(c2) = f1(c2).  [resolve(391,a,556,a),flip(a)].
679 occurrence_of(successor(f1(c2),f6(c2)),f1(c2)).
[back_rewrite(555),rewrite([677(2),677(7)])].
688 successor(f1(c2),f14(c2)) = c2.
[back_rewrite(298),rewrite([677(2)])].
885 arboreal(successor(f1(c2),f6(c2))).  [resolve(679,a,386,a)].
1434 $F.  [resolve(374,a,885,a),rewrite([688(6)]),unit_del(a,219)].

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