A Formalization of Elements of Special Relativity in Coq

Contents

Chapter 1

Introduction

1.1 Motivations

Physics often entails the performance of manual calculation, which benefits from computational support. This often takes the form of e.g. Wolfram Mathematica, which is used widely by physicists and physics students to accelerate computations and algebraic manipulation. These calculations are often more easily carried out in computation rather than on paper. Besides the direct benefits of speed conferred by the use of automation in these tasks, computational support also reduces the rate at which errors are committed, because computational tools typically do not make errors, whereas physics students may be error-prone (in the author's experience).

However, tools such as Mathematica do not go far beyond presenting a computational convenience. They do not capture the higher-level reasoning that guides the computations performed by physicists. The more fundamental portion of a physical theory consists of these higher-level statements regarding the entities described by the theory—rather than the computations, which are informed by these high-level descriptions. Examples of such statements from special relativity include that "two inertial observers should not have a relative velocity faster than the speed of light" or that "the coordinate systems induced by two observers should be related by a Poincaré transform." An example of a computation informed by the latter statement is the computation of the relative position and velocity

of two inertial observers from the coordinates they assign to a set of events, which might proceed by finding the parameters of an appropriate Poincaré transform.

Instead of a calculational tool like Mathematica, one might consider the use of a system which is able to encode these higher-level statements. An example of such a system is Coq[1]. Coq is a formal proof management system which is sufficiently powerful to support statement and proof of sophisticated theorems about programs. As we will show, Coq's programming language is expressive enough to formalize statements occurring in special relativity.

A few features of theorem proving systems such as Coq make them attractive targets of investigation. First, the applicability of a theorem to an entity or situation is unambiguous, and the validity of the conclusion is certain. This may ameliorate uncertainties harbored by a physics student about the applicability of a concept and in particular eliminates the possibility of an erroneous conclusion from an inapplicable statement. Second, Coq supports user-defined automation, which might be exploited to automate proofs based on theorems that the user has identified as useful. Third, Coq gives very high assurance that the conclusions proven are correct, at least insofar as they follow from the axioms that were selected to describe the physical universe to the system; it is encouraging to obtain this level of guarantee regarding a foundational physical theory. These are properties shared by many theorem proving systems; other work has previously demonstrated the use of Isabelle in formally verifying physical reasoning.

In this work, we formalize a preliminary portion of the theory of special relativity, demonstrating that Coq may be used to formalize physical reasoning at the level of introductory physics. Furthermore, we show that the automation mechanism of Coq can significantly reduce programmer effort for formalizing physical reasoning. The following describes the structure of this document.

In Chapter 2, we introduce some background for Coq and special relativity that may assist with understanding the remainder of the document. Readers familiar with either Coq or special relativity may wish to skip these sections. The section on Coq contains an overview of what it is, some examples for how it is used interactively, and samples of proof and automation code with explanations of their contents. It also introduces some aspects of

[1] https://coq.inria.fr/

Coq's logical system that affect the way we carry out proofs in the rest of the document. The section on special relativity contains an overview of what it is, some predictions it makes, and an introduction to a mathematical representation.

In Chapter 3, we discuss an axiom system that gives a rigorous mathematical representation of the relativistic universe. We display the formalization of the axioms in Coq and explain the meaning of each axiom in more intuitive terms. We also give high-level descriptions of the manner in which the axioms are used. The proofs from the rest of the work are based on this axiom system. We begin to discuss our Coq development after Chapter 3.

Chapter 4 describes a portion of the Coq development in which we develop reasoning about the concept of "event" that is presented by these axioms. We need to develop these additional means of reasoning due to a property of Coq's logic—the native use of an intensional rather than extensional equality. The results are important in supporting the use of events in the same manner as they are used in proofs outside of Coq.

Chapter 5 presents a proof that one definition of the causal relation is transitive, which is a first step towards a formalized proof of the Alexandrov-Zeeman theorem. This proof demonstrates one way to conduct reasoning about vectors and real numbers in Coq with the automation we developed as support; we observe the effectiveness of the automation and the importance of the availability of lemmata that permit the use of common steps in reasoning.

Chapter 6 presents a formalization of a proof that an inertial observer will not see another inertial observer to be traveling at a relative velocity equal to the speed of light, an important preliminary fact about movement in a relativistic universe. We use this proof to demonstrate our capability to replicate physical reasoning about events, vectors, and real numbers realistically.

Chapter 7 describes the guts of the real number reasoning automation supporting these proofs, starting from the tactic `real_crush`. We explain the procedures the automation performs in order to solve the goals we encountered. We give observations regarding the development and improvement of the automation.

Chapter 8 concludes with some observations regarding the use of Coq for physical reasoning.

Chapter 2

Background

This chapter is intended to give an informal overview of the tool (Coq) and physical theory (special relativity) I will be discussing in the remainder of this document. Readers familiar with Coq or special relativity may wish to skip the relevant sections.

2.1 Coq

Coq[1] is a formal proof management system. Coq provides a programming language in which users may specify formal theorems. Coq also provides an interface for *interactively* supplying formal proofs of these theorems. Additionally, Coq's programming language is a fully-fledged functional programming language which may be used to write practical programs.

2.1.1 Interactive theorem proving

Broadly, the use of the interactive proof interface consists of the following steps:

1. The user states a theorem in Coq's programming language, which becomes a *proof obligation* or *goal* which Coq will present to the user.
2. Coq presents one of the remaining proof obligations to the user.
3. The user selects a *tactic* to express a piece of reasoning about the presented proof obligation. Tactics implement backwards reasoning, so that when a tactic is applied

[1] https://coq.inria.fr/

to a goal, the hypotheses it requires to prove the goal are generated as new proof obligations to be met [9].

4. Repeat from step 2 until all proof obligations are discharged.

For instance, we may try to prove the following:

```
Lemma foo (A : Prop) (B : Prop) : A → (A → B) → B.
```

We enter the interactive mode with the command `Proof.` Coq will generate a proof obligation which it lists as:

```
A, B : Prop
============================
 A → (A → B) → B
```

The bar may be read as "implies". Currently this may be read as "for some propositions A and B, we have $A \implies (A \implies B) \implies B$." Now invoking the tactic `intros` will transform the goal into one which has the hypotheses of this implication above the line and the conclusion below:

```
A, B : Prop
H : A
H0 : A → B
============================
 B
```

The names `H` and `H0` are automatically generated. Now one may use the tactic `apply`, as in `apply H0`, to reduce this goal:

```
A, B : Prop
H : A
H0 : A → B
============================
 A
```

The reason this has replaced `B` with `A` might be thought of in the terms: "`H0` would be able to prove `B` if it had some `A` available, so applying `H0` will solve `B` while generating an obligation for an `A`." However, we do have `A`, in the form of `H`. The goal is solved by `apply H`, generating the message: `No more subgoals.` The final *proof script* reflecting what the user

entered is below:

```
Lemma foo (A : Prop) (B : Prop) : A → (A → B) → B.
Proof.
  intros.
  apply H0.
  apply H.
Qed.
```

This is an artifact that may be retained and fed into the compiler again to re-verify the proof. During the verification of such a "proof script," the compiler executes the tactics sequentially as though participating in an interactive session with a human.

2.1.2 Tactics and automation

Tactics are a powerful feature of Coq. Tactics may be used to automate reasoning. For example, the Coq tactic `auto` attempts to automatically solve a goal by recursively applying a small set of operations to a certain depth. Coq supports an extensive library of tactics by default, of which `intros`, `apply`, and `auto` are amongst the most primitive. The user may also write their own tactics in Ltac, Coq's tactic language.

Ltac is a programming language that allows its user to automatically apply tactics in an imperative way using the same syntax that is used during interactive theorem proving. Ltac provides pattern matching and function call primitives for controlling the application of these tactics.

We here dissect a representative example of Ltac whose usage and purpose will also be discussed later in the document (in section 7.1.4).

```
Ltac rs_len sum tac :=
  match sum with
    | ?sum' + _ ⇒ rs_len sum' ltac:(fun x ⇒ tac (S x))
    | _ ⇒ tac 0%nat
  end.
```

This defines an Ltac function `rs_len` that accepts two arguments, which will be referred to internally as `sum` and `tac`. The `match` statement pattern matches against `sum`, expecting that it takes the form of some `sum'` added to something.

We discuss the first branch in detail. The underscore is a wildcard indicating that the

pattern match should disregard the right hand side of the sum for this branch. On the right hand side of the arrow `=>` is the action to be taken in the case that the pattern to the left of the arrow matches. Here, `sum'` becomes defined as whatever was pattern matched. This branch recursively calls `rs_len`, passing in `sum'` and a newly constructed piece of Ltac that implements a continuation-passing way of constructing an argument for the given `tac`.

If the pattern match succeeds, then the execution of the match statement will terminate here. If the pattern match fails *or* the tactics that are then invoked fail, the match statement will execute the next statement. Here, the second branch has a wildcard underscore for a pattern, meaning that it will always be executed when the first branch falls through. Match statements that reach the end without managing to match a pattern will also fail.

This fall-through behavior of attempting a next branch when the invoked tactics in a currently executing branch fail is an important language feature for supporting proof automation. This enables native support of backtracking proof search; when one branch of the proof fails, Ltac is able to automatically attempt another approach. Failures propagate out of recursions, so the backtracking can proceed even if a distant recursive call fails. In this way, a match statement may rather be read as a sequence of operations to attempt in the course of proof search. Explicit invocations of the tactic `fail` can be idiomatic, with a meaning of "now try the next branch." [6]

2.1.3 Logic

Coq implements a constructive logic called the calculus of constructions. Because it is a constructive logic, it does not natively support the *principle of excluded middle*, i.e. that $\forall P, P \vee \neg P$, or that $\forall P, \neg\neg P \implies P$. It also does not support *functional extensionality*, which is the statement that two functions f, g are equal when they are equal on all their arguments: $(\forall x, f(x) = g(x)) \not\Longrightarrow f = g$.

In Coq, two things are equal (without further qualification) when they are syntactically or "intensionally" equal. Intuitively, intensional equality is equality of representation, whereas extensional equality is equality of behavior. In these terms, the reason why functional extensionality is not given in Coq is that the function may be implemented in two different ways. For instance, add may be implemented in either of the two following ways:

```
Fixpoint add1 (x : nat) (y : nat) :=
  match x with
  | O ⇒ y
  | S x' ⇒ S (add1 x' y)
  end.

Fixpoint add2 (x : nat) (y : nat) :=
  match y with
  | O ⇒ x
  | S y' ⇒ S (add2 x y')
  end.
```

These functions can be shown to be extensionally equal, which may be written in Coq as:

```
Lemma add1_extensional_eq_add2 :
  forall n m : nat, add1 n m = add2 n m
```

(Proof by induction on `n` and `m`.) But, `add1` and `add2` are not syntactically equal. Because their representations are not equal, they are not intensionally equal.

The absence of the principle of excluded middle and functional extensionality affects the ease with which certain arguments may be executed. Their absence in the calculus of constructions notwithstanding, the principle of excluded middle and functional extensionality may be assumed to be true in Coq without generating a contradiction. Out of interest in the expressiveness of Coq, we make an attempt in this work to track where these assumptions enter the reasoning.

2.2 Special Relativity

Special relativity is a physical theory expressing a relationship between space and time as they are perceived by observers in relative motion. Roughly, special relativity reflects the implications of the observation that the speed of light appears the same to all observers, *irrespective* of their motion relative to the source of the light. To give a concrete image, we may state this in terms of a hypothetical experimental setup:

1. A laser is placed on a train platform.

2. There is an observer Alice on the platform, who is carrying some sophisticated device that can measure the speed of light passing it.

3. There is also a train speeding by the platform. This train is carrying a second observer Bob, who is also carrying a light speed measuring device.

4. The laser is fired down the platform, parallel to the tracks. Alice and Bob both measure the speed of the laser as it passes them.

5. If the train were moving at, say, 30 mph in the direction of travel of the laser, one might expect Bob to measure a speed 30 mph less than the speed Alice measured. However, this is not the case: both of them measure the same speed.

It is now possible to conduct a series of thought experiments that more or less intuitively demonstrate some of the effects predicted by relativity. Here is a demonstration of time dilation, a phenomenon where observers in relative motion see each others' clocks to be running slow:

1. There is an infinite frictionless horizontal plane (we are keeping things to two dimensions for ease of visualization).

2. Imagine two mirrors oriented horizontally, with one laying upon the plane and the other arranged to be hovering directly above it. A vertically-traveling beam of light might be placed between these mirrors such that the beam bounces vertically up and down forever. The mirrors might be of such a distance apart that the light takes one second to make a round trip. An astronaut—Alice—is sitting atop these mirrors counting the number of times the light has come by; this arrangement is a "light clock". Call this light clock "clock A".

3. Another light clock, "clock B," slides by the first one. To Alice, clock B must be ticking less frequently than her own clock A, because the light in clock B must travel in an apparently diagonal line to stay within the clock as it moves. Correspondingly, if astronaut Bob were sitting atop clock B, Alice should observe him slowed down as compared to how he would appear when they were not moving relative to one another.

The mathematical description of relativity often begins by specifying a coordinate system for locating "events," which assigns a location in space and a moment in time to each event. Intuitively, events may be things like interactions—when a particle encounters another. For instance, consider a universe with one dimension of space and one dimension of time, or a 1+1-dimensional universe. Arbitrarily fixing a coordinate system by selecting an origin, we might indicate the location and moment in time of an event by specifying that it exists "3 meters left and 1 second in the future" relative to the origin. These notions are formalized below.

It is possible to compute the effect of a relative velocity on the coordinates. If (in our 1+1-dimensional universe) an event were located at coordinates (x, t) in the eyes of an observer A, then in the eyes of an observer B colocated with A but moving at relative speed v, the event would be seen at coordinates (x', t'), where:

$$x' = \gamma(x - vt) \tag{2.1}$$

$$t' = \gamma\left(t - \frac{vx}{c^2}\right) \tag{2.2}$$

in which γ, the Lorentz factor, is given by:

$$\gamma = \frac{1}{\sqrt{1 - v^2/c^2}} \tag{2.3}$$

and c is the speed of light, about 3×10^8 meters per second, or some 670,000,000 miles per hour. The Lorentz factor may be seen as the factor by which time appears to be slowed down for something moving at relative velocity v to an observer.

The axioms selected for this development make precise the notion of body, event, and observed coordinates. These are described and discussed in Chapter 3.

We work in a 3+1-dimensional universe, so our coordinates will be given by tuples of four points. Our convention is to write them as (x, y, z, t) so that the fourth entry is the time coordinate.

2.3 Notations

We give an overview of the notation used.

Mathematical notation $A \wedge B$ will be written with *italic* typesetting. Coq code `A /\ B` will be written under a `monospace` font, with some symbols replaced for pretty-printing.

2.3.1 Mathematical notation

Mathematical objects discussed in this work include real numbers, three-dimensional vectors, and four-vectors. Real numbers and four-vectors are both written with a variable, as x; which is in use will be inferred from context. Three-dimensional vectors will be marked with a vector $\vec{x}$ like so. We indicate the zero vector with $\vec{0}$. Real, vector, and four-vector operations will be written with the same symbols $+, -$, etc..

We norm vectors using double bars $||\cdot||$, so one might see $||\vec{x}||$. We use single bars $|\cdot|$ to take the absolute value of real numbers: $|x|$. Tuples are written with parentheses, so for instance the zero four-vector is written $(0, 0, 0, 0)$. We use a convention with time written in the fourth coordinate: (x, y, z, t). Components are written with x_1, x_2, x_3, x_4 indicating the respective position in the tuple.

If $x = (a, b, c, d)$ is a four-vector, then we will take its spatial component using: $\vec{x^s} = (a, b, c)$ and its temporal component using: $x^t = d$.

Propositional relations between objects will be written as $P(a, b, c)$.

2.3.2 Coq notation

We have developed notation to more briefly express the concepts outlined above. Variables are not marked in Coq; the operations enforce type safety.

We norm four-vectors using bar-brackets `|[ x ]|`. This norm is the Euclidean norm, i.e. `|[ (x, y, z, t) ]| = sqrt (x * x + y * y + z * z + t * t)`. We do not take the norm of either three-dimensional vectors or real numbers separately, as will be discussed momentarily. Tuples are written with parentheses as before: `(a, b, c, d)`. Components are written `x _x`, `x _y`, `x _z`, `x _t`, indicating the respective position in the tuple.

Real number operations are written with the standard notation: `a + b`, `a - b`. The

syntax `a + - b` indicates `a + (- b)`, or simply `a - b`. One may also explicitly write the name of the functions associated with the notation: `Rplus a b` for `+`, for instance.

Four-vector operations are written with a different notation which includes a `v` in the operator: `x +v+ y`, `x -v- y`. Function names may also be written explicitly: `v4dot x y`.

If `x` is a four-vector `(a, b, c, d)`, then we will take its spatial component using: `x ^s`. We represent three-dimensional vectors as projections of four-dimensional vectors, so in fact `x ^s = (a, b, c, 0)`. Similarly, the temporal component is taken using: `x ^t = (0, 0, 0, d)`.

Propositional relations between objects will be written in Coq's notation `P a b c`.

Chapter 3

Axioms for Special Relativity

In this chapter we describe and explain the axioms we used to represent a relativistic universe to Coq. For this development we adapted a first-order logic axiom system for special relativity, SpecRel, developed by H. Andréka, J.X. Madarász, I. Németi and G. Székely [3] [13] [4]. The original SpecRel axioms allowed for flexibility regarding the set of quantities that may compose a set of coordinates; we have adapted SpecRel to apply specifically to 4-tuples of real numbers, where we have used Coq's library theory of real numbers. After this modification, the axiom system that we use for the development comprises five axioms.

We selected SpecRel because it reflects a common means of expressing situations in textbook problems in special relativity—in terms of bodies and their relative locations in a particular reference frame. There are other axiom systems describing relativistic universes; [2] for instance, Robb developed an early one in 1914 [10]. However, these describe the structure of the universe geometrically, rather than explicitly describing the properties of and relationships between a set of bodies [7] [2].

SpecRel expresses the universe in terms of a set of bodies as well as a coordinatization relation that allows inertial observers to locate the bodies in space. Disadvantageously for formalization in Coq, events take on a second-class role as sets of coincident bodies, defined and equated only extensionally. However, with some development, it is possible to recover a closer-to-textbook notion of events as being identified with points in spacetime.

Because readers may not be familiar with Coq notation, in our descriptions of the axioms

we make a distinction between the mathematical statement of a theorem and the notation used to describe it to Coq.

3.1 Elements

The axioms are assertions about a set of bodies B, a predicate on bodies IB indicating that a body is an inertial body, a predicate on bodies Ph indicating that a body is a photon, and a three-place relation on body, body, and coordinate W which captures the notion of observation. In Coq, we define these as variables heading a module containing the axioms. Their formalizations in Coq are as follows:

```
Variable body : Type.
Variable IB : body → Prop.
Variable Ph : body → Prop.
Variable W : body → body → v4 → Prop.
```

The three-place relation W should be interpreted as follows: if we have $W(b_1, b_2, x)$ for bodies b_1 and b_2 and coordinates x, then b_1 observes b_2 at coordinates x, or b_1 *coordinatizes* b_2 at coordinates x. The set of observations $W(b, ...)$ (which we do not speak of rigorously in Coq) may be referred to as a coordinatization of spacetime.

Andréka et. al. make extensive use of the following abbreviated concepts. We may define another predicate on bodies Ob as follows: $\exists bv, W(m, b, v)$, which is given in Coq as:

```
Definition Ob m := exists b v, W m b v.
```

That is, m is an observer if there exists a body b and a coordinate such that m observes b at these coordinates. In particular, the body and coordinates observed by m are not specially important: m is an observer if it (via W) observes *anything anywhere*. A more specific predicate on bodies IOb captures the additional requirement that this body is inertial: $IOb(m) \equiv IB(m) \wedge Ob(m)$, which is given in Coq as:

```
Definition IOb m := IB m ∧ Ob m.
```

That is, a body is an inertial observer if it is both an inertial body and an observer.

These definitions in place, we will now proceed to describe each of the five axioms con-

straining these predicates in turn.

3.2 AxPh

AxPh is shorthand for axiom of photons. It is given by:

$$\forall m, \exists c, \forall x, y, IOb(m) \implies$$
$$(\exists p, Ph(p) \wedge W(m,p,x) \wedge W(m,p,y) \iff ||\vec{y^s} - \vec{x^s}|| = c|y^t - x^t| \tag{3.1}$$

We formalize it in Coq as:

```
forall m, exists c, forall x y,
  IOb m →
  (exists p, Ph p ∧ W m p x ∧ W m p y) ↔
  |[ y^s -v- x^s ]| = c * |[ y^t -v- x^t ]|
```

The interpretation of this axiom is as follows. We may define two points to be *lightlike* or *null separated* when they satisfy the equality on the right hand side of the biconditional in (3.1). This indicates that they lie along a trajectory that might be taken by an object moving at some speed c. The axiom states that two points with coordinates x, y are null separated iff there is a photon that passes through both points.

This axiom is the crucial axiom for fixing the structure of the spacetime. As will be stipulated by AxEv, a photon p that is seen by an inertial observer m must be seen by all. In particular, some other observer m' will see p in the same two events at some possibly different coordinates. Moreover, m' will also coordinatize the events to be null separated. In other words, AxPh transfers null separation observed in one coordinatization to all others.

In this way, photons serve to indicate a spacetime structure that is invariant between different coordinatizations. Broadly, photons may be used to "carry" information from one inertial observer's coordinatization to another. As is done in our proof of "no luminal inertial observers," it is possible to indicate a set of photons known to intersect at certain points, and reason about the properties of this set across different coordinatizations.

Another consequence of AxPh is that space is filled with photons: given some coordinates x, one could find any number of coordinates y, each of which result in a photon passing

through x, so every point in space is perforated by a hail of photons from every direction. Thus photonic bodies should not be thought to correspond to anything physical. Rather, their importance is in the consistency of their trajectories across multiple coordinatizations, which serves to lend the spacetime structure as discussed.

3.3 AxEv

AxEv is shorthand for axiom of events. It is given by:

$$\forall m, k, IOb(m) \wedge IOb(k) \implies \forall x, \exists y, ev(m, x) = ev(k, y) \tag{3.2}$$

Andréka et. al. use $ev(m, x)$ to denote the set of bodies coordinatized at point x by body m. This requires interpretation before it can be accepted by Coq. In order to formalize this in Coq, we define a new four-place relation *eveq* as follows:

```
Definition eveq b1 v1 b2 v2 := forall b, W b1 b v1 ↔ W b2 b v2.
```

This expresses the equality of events in terms of the following: b is in the event seen by b_1 at v_1 iff b is in the event seen by b_2 at v_2. This is an extensional equality between sets defined in terms of propositions whose proof demonstrates belonging to the set. We will eliminate this in the course of development by creating independent, explicit statements of extensional equality. Now using *eveq*, we formalize AxEv in Coq as follows:

```
forall m k,
  IOb m ∧ IOb k →
  forall x, exists y, eveq m x k y
```

AxEv is a consistency axiom that states that if an inertial observer m observes an event at some set of coordinates, then another inertial observer k must also observe the event at some set of coordinates. This ensures that there is a single consistent universe and that all inertial observers observe it.

3.4 AxSf

AxSf is shorthand for axiom of self. It is given by:

$$\forall m, IOb(m) \implies (\forall v, W(m, m, v) \iff \vec{v^s} = \vec{0}) \tag{3.3}$$

which may be formalized directly in Coq as:

```
forall m,
  IOb m →
  (forall v, W m m v ↔ v _x = 0 ∧ v _y = 0 ∧ v _z = 0)
```

This axiom captures a property of the worldview of an inertial observer under relativity: an inertial observer cannot distinguish its motion from rest with all other bodies moving around it. For convenience, we may then place the body at rest at the origin.

3.5 AxSm1

AxSm1 is shorthand for the first symmetry axiom. It is given by:

$$\begin{aligned}
&\forall mk, IOb(m) \wedge IOb(k) \implies \\
&\qquad \forall x, y, x', y', x^t = y^t \wedge x'^t = y'^t \wedge \\
&\qquad\qquad ev(m, x) = ev(k, x') \wedge ev(m, y) = ev(k, y') \implies \\
&\qquad\qquad ||\vec{x^s} - \vec{y^s}|| = ||\vec{x'^s} - \vec{y'^s}||
\end{aligned} \tag{3.4}$$

This may be formalized in Coq as:

```
forall m k,
  IOb m ∧ IOb k →
  forall x y x' y', x^t = y^t ∧ x'^t = y'^t ∧
                    eveq m x k x' ∧ eveq m y k y' →
                    |[ x^s -v- y^s ]| = |[ x'^s -v- y'^s ]|
```

This axiom ensures that if two observers agree that two events are simultaneous, then the observers will agree on the distance between the two events. This is used to ensure that the "units of measurement" used by each observer are the same. Otherwise, coordinatizations

of two different bodies might be related by some dilation factor other than 1.

3.6 AxSm2

AxSm2 is shorthand for the second symmetry axiom. It is given by:

$$\forall m, IOb(m) \implies \exists p, Ph(p) \wedge W(m, p, (0, 0, 0, 0)) \wedge W(m, p, (1, 0, 0, 1)) \tag{3.5}$$

In Coq, we write:

```
forall m,
  IOb m → exists p, Ph p ∧ W m p (0, 0, 0, 0) ∧ W m p (1, 0, 0, 1)
```

This axiom fixes the speed of light (c, as it appeared in AxPh) to be 1.

Andréka et. al. are concerned with avoiding the use of this axiom as part of the process of logical analysis of relativity, in order to see that many properties of relativity manifest themselves irrespective of the speed of light [3]. In this development, we are not concerned with this logical analysis, so a first proof is to merge AxSm1 and AxPh into an AxPh' where the speed of light c has been removed. The statement of AxPh' is as follows:

```
Lemma ax_ph':
  forall m, forall x y,
    IOb m →
    (exists p, Ph p ∧ W m p x ∧ W m p y) ↔
    |[ y^s -v- x^s ]| = |[ y^t -v- x^t ]|.
```

The proof proceeds as follows.

1. Before attempting the goal, we analyze AxSm2 and AxPh, solving for c to show that it must be 1.

2. Substituting this, we obtain a version of AxPh where the multiplication is by 1 instead of by c.

3. We use this version to demonstrate the goal, which elides the multiplication.

4. This demonstration proceeds by demonstrating that the multiplication by 1 in the intermediate version of AxPh leaves the expression the same as in the AxPh' statement,

which is done by our `real_crush` automation.

From this point forth, we will take "null separated" to implicitly include that $c = 1$.

Chapter 4

Events

4.1 Extensional equality between events

The axioms given by Andréka et. al. discuss events in terms of extensionally defined sets, where an element is a member of a set when it satisfies a predicate. In this case, we have that a body b is an element of the event $ev(m, x)$ when it satisfies the predicate:

```
fun b ⇒ W m b x
```

In the course of constructing results from these axioms, Andréka et. al. reason using equalities between events. This is not so convenient to do in Coq. Because this "equality" is extensional, we cannot conclude from it that "equal" events must be at the same location. Instead, we define a new extensional equality, which we denote with `=e=`, as in the following:

```
ev b x =e= ev b y
```

We demonstrate that this is an equivalence relation; this allows us to perform equational reasoning over events when we are able to demonstrate that the equations hold. This equality is not as general as Coq's native equality `=`; what this means in particular is that we are obliged to explicitly demonstrate to Coq under which situations it is appropriate to use it to perform rewrites.

4.2 Event equality and coordinate equality

Ultimately, we would like to connect this notion of event equality to coordinate equality, i.e.:

```
Lemma ev_diff :
  forall b x y, IOb b → ev b x =e= ev b y ↔ x = y
```

Such a result would allow us to consider events as first-class points in spacetime, in close association with the coordinates at which they may be found. The reverse implication follows from a rewrite: since $x = y$ we may simply replace x with y, and the result is true by reflexivity. The forward implication seems difficult to prove directly because it would require us to construct an explicit equality from an extensional one. Intuitively, because the hypothesis is quite weak (leaving x and y fairly unconstrained), this would entail the construction of the values of x and y directly from the contents of the events, which we are not even given to be nonempty.

Instead, we demonstrate a modified theorem `ev_diff'`, which is the statement that different events exist at different coordinates and vice versa:

```
Lemma ev_diff' :
  forall b x y, IOb b → (ev b x <e> ev b y) ↔ x <> y.
```

In particular, `ev_diff'` is used in Andréka et. al.'s proof of "no faster than light inertial observers." There, its statement is that different events exist at different coordinates [4]. They briefly note that this may be demonstrated by constructing different photons for different coordinates. Our proof then proceeds in this manner. We first prove `ph_distinct`, the statement that if two coordinates are different, then there exists a photon present at the event of the first coordinate but not the event of the second:

```
Lemma ph_distinct :
  forall x y, x <> y → exists z,
                          |[ x^s -v- z^s ]| = |[ x^t -v- z^t ]| ∧
                          |[ y^s -v- z^s ]| <> |[ y^t -v- z^t ]|.
```

Our proof of `ph_distinct` proceeds by indicating a point z which is null separated from x but not from y. Intuitively, we can select a z that shares spatial coordinates with y but is at a different time from y, which certainly means that it and z are not null separated.

Furthermore we can select the time at which z exists to make it null separated from x.

1. We use Coq's axiom of total ordering over the reals to inform as to which z to construct. The time coordinate of x could be before or after the coordinate of y:

$$x^t < y^t \vee x^t = y^t \vee x^t > y^t \tag{4.1}$$

2. Our proof proceeds to select z to be temporally after x when y is before or simultaneous with x, and is temporally before x when y is after x.

3. In the first case ($x^t \leq y^t$), we then pick z to be:

$$z = (y_1, y_2, y_3, x^t + ||\vec{y^s} - \vec{x^s}||) \tag{4.2}$$

   By inspection, we see that:

$$||\vec{x^s} - \vec{z^s}|| = ||\vec{y^s} - \vec{x^s}|| = |x^t - z^t| \tag{4.3}$$

   which satisfies the first condition on z.

   Likewise, we see that:

$$||\vec{y^s} - \vec{z^s}|| = 0 \tag{4.4}$$

$$|y^t - z^t| = |y^t - x^t| + ||\vec{y^s} - \vec{x^s}|| \tag{4.5}$$

   However, since the latter is nonzero from the stipulation $x \neq y$, the second condition holds for this z as well.

   In the second case ($x^t > y^t$), we pick z to be:

$$z = (y_1, y_2, y_3, x^t - ||\vec{y^s} - \vec{x^s}||) \tag{4.6}$$

   The proof that this z satisfies the desired conditions proceeds likewise.

The arithmetic necessary to demonstrate that z is null separated from x and not null separated from y is handled in large part by the real number automation, whose presence is

important for minimizing the size of the proof. We did not complete a proof without automation for comparison. However, since our first success at proving this result, improvements in our automation have reduced the size of the proof from 111 lines to 32.

Given `ph_distinct`, we prove the forward implication of `ev_diff'` by directly constructing a contradiction: if the events were equal, then the photon constructed by `ph_distinct` would be found at both coordinates, which is not possible. As before, the reverse implication is trivial.

4.3 Classical reasoning

The proof of `ph_distinct` has introduced non-constructive reasoning by the use of the trichotomy of the total ordering over the real numbers: this is not decidable in general, and the assumption that it is is classical. Thus `ev_diff'` depends in some way on classical reasoning. Additionally, it is possible to prove the originally desired `ev_diff` from `ev_diff'` by using decidability of equality between coordinates. However, this is obtained from the decidability of equality over the real numbers, which is also a classical assumption.

A priori, because Coq does not include the principle of excluded middle, the contrapositive cannot be used to remove a negation, and `ev_diff'` should not directly imply `ev_diff` (though the reverse implication does hold). However, we are able to prove this implication (ultimately resulting in a proof of `ev_diff`) via the use of classical reasoning. We have highlighted intrusions of classical reasoning into our proofs as they were presented here; however, we have not demonstrated that the original axioms used to encode relativity did not somehow introduce classical reasoning, in which case we should not be concerned with further introductions of classical reasoning via the means we have pointed out.

4.4 Rephrasing axioms

This established, it is possible to rephrase AxEv and AxSm1 in terms of `=e=` instead of `eveq`:

```
Lemma ax_ev' :
  forall m k : body,
    IOb m → IOb k → forall x : v4, exists y : v4, ev m x =e= ev k y.

Lemma ax_sm1' :
  forall m k,
    IOb m → IOb k →
    forall x y x' y', x^t = y^t →
                      x'^t = y'^t →
                      ev m x =e= ev k x' →
                      ev m y =e= ev k y' →
                      |[ x^s -v- y^s ]| = |[ x'^s -v- y'^s ]|.
```

The proofs of these proceed easily by unfolding the definitions of `ev`, `=e=`, and `eveq` in the original axiom statements. The result is dispatched by `auto` after applying the original axiom statements.

Chapter 5

Causality

An important structural aspect of the relativistic universe is the causality relation. Physically, an event x can precede another event y causally iff the difference in its time coordinate is less than or equal to the difference in its space components, and x precedes y in time:

$$x << y \equiv ||\vec{y^s} - \vec{x^s}|| \leq |y^t - x^t| \wedge x^t < y^t \tag{5.1}$$

where we have denoted causality using $<<$. We may further break this relation into two weaker notions of causality:

$$x < y \equiv ||\vec{y^s} - \vec{x^s}|| < |y^t - x^t| \wedge x^t < y^t \tag{5.2}$$

$$x < \cdot y \equiv ||\vec{y^s} - \vec{x^s}|| = |y^t - x^t| \wedge x^t < y^t \tag{5.3}$$

which might be termed strict causality and photon causality. The latter notion is especially relevant because it is reflected directly by the axiom system used in this development. In particular, when a photon exists between two points x and y, they are related by this photon causality $< \cdot$. It can be shown that these two forms of causality, however, are equivalent [14].

5.1 Alexandrov-Zeeman theorem

The Alexandrov-Zeeman theorem [14] explains that the ways in which observers' viewpoints may differ arises from the causal structure of spacetime. This is important in justifying (from SpecRel) the higher-order description of the relativistic universe in terms of worldlines, relative velocities, and Poincaré transforms which is often given in textbooks [1].

The properties of and relationships between the two forms of causality `cause` and `causeP` are used to prove the Alexandrov-Zeeman theorem. Explicitly, it states that the light-ray preserving bijections on the coordinate space consist of Poincaré transforms, possibly composed with a dilation of space. Zeeman's result gives that a bijection f which preserves light rays, and whose inverse f^{-1} also preserves light rays, is a Poincaré transform [14]. Cacciafesta showed that a minimal set of assumptions that can recover the same result only needs that f preserves light rays, since it is possible to prove that its inverse also preserves light rays [5]. Finally, Székely showed that due to AxSm1, the bijection f must be a Poincaré transform *without* a dilation [13]. So, one might begin with functions f that respect the light ray structure induced by AxPh, and obtain that f must be a Poincaré transform.

5.2 Transitivity of strict causality

As a first step towards a formalization of this, we have demonstrated that strict causality is a transitive relation. (Photon causality is not transitive.)

5.2.1 Definitions

AxPh specifies that two points lie on a light ray if their spatial separation equals their temporal separation. Recall the condition:

$$||\vec{y^s} - \vec{x^s}|| = |y^t - x^t| \tag{5.4}$$

In contrast, textbook material often instead discusses the invariant interval:

$$s(v) = (v^t)^2 - ||v^s||^2 \tag{5.5}$$

Two points x, y are specified to lie on a light ray when $s(y - x) = 0$, or they have null invariant interval. We also see that they may only be causally related under the definitions given above when $0 \leq s(y - x)$. For this segment, we switch to a language that discusses point pairs in terms of the invariant interval between them:

```
Definition interval (v : v4) : R := (v _t)^2 - (v _x)^2 - (v _y)^2 - (v _z)^2.
Definition lightlike (v : v4) : Prop := interval v = 0.
Definition timelike (v : v4) : Prop := interval v > 0.
Definition spacelike (v : v4) : Prop := interval v < 0.
```

We may connect `lightlike` in particular with photons in the axiom system, via the following two lemmata:

```
Lemma lightlike_eq :
  forall x y, lightlike (y -v- x) ↔ |[ y^s -v- x^s ]| = |[ y^t -v- x^t ]|.

Lemma lightlike_has_ph :
  forall m, forall x y,
    IOb m →
    (exists p, Ph p ∧ W m p x ∧ W m p y) ↔ lightlike (y -v- x).
```

We are able to prove the first lemma immediately using our real number automation. The second, `lightlike_has_ph`, also follows immediately from the first, which gives the criterion required to demonstrate the existence of a photon under AxPh.

In these terms, we define $<$, `cause`, and $< \cdot$, `causeP`, as follows:

```
Definition cause (x y : v4) := timelike (y -v- x) ∧ x _t < y _t.
Definition causeP (x y : v4) := lightlike (y -v- x) ∧ x _t < y _t.
```

5.2.2 cause_trans

Using this, the theorem statement is:

```
Lemma cause_trans :
  forall x y, cause x y → forall z, cause y z → cause x z
```

Given that $0 < s(y - x) \wedge x^t < y^t \wedge 0 < s(y - z) \wedge y^t < z^t$, we would like to show $0 < s(z - x) \wedge x^t < z^t$. The second obligation comes immediately. The proof of the first part

begins by rewriting the obligation to make use of the triangle inequality:

$$||\vec{y^s} - \vec{x^s}|| < y^t - x^t \wedge \tag{5.6}$$

$$||\vec{z^s} - \vec{y^s}|| < z^t - y^t \tag{5.7}$$

$$\implies$$

$$||\vec{z^s} - \vec{x^s}|| < z^t - x^t \tag{5.8}$$

(5.7) gives an inequality on y^t. We may use it to eliminate y^t in (5.8), obtaining the hypothesis:

$$||\vec{y^s} - \vec{x^s}|| + ||\vec{z^s} - \vec{x^s}|| < z^t - x^t \tag{5.9}$$

This substitution requires the fact that $b < a \wedge c + a < d \implies c + b < d$ for any a, b, c, d, which we prove separately. Overall we observe that some common intuitive reasoning steps when working with inequalities, such as this one, are not represented in the default Coq library. For an interactive proof system intended to support physical reasoning, it is important for convenience that lemmas enabling these reasoning steps—or some other means of accomplishing quickly accomplishing these steps—are available.

Our ability to rewrite the goals in the manner described is supported by the presence of real number automation. These rewrites generate auxiliary goals that would otherwise need to be proved separately, and after the fact; this disrupts the flow of the proof, increasing proof effort and difficulty of comprehension. With real number automation, these goals may be dispatched immediately and without distracting from the main portion of the proof.

To complete the proof, it suffices to show that:

$$||\vec{y^s} - \vec{x^s}|| + ||\vec{z^s} - \vec{x^s}|| \leq ||\vec{z^s} - \vec{x^s}|| \tag{5.10}$$

This follows from the triangle inequality. Our development's proof of the triangle inequality is incomplete; it still takes as assumed the Cauchy-Schwarz inequality, whose proof we leave to future work.

Chapter 6

No Relative Motion at c

An important fact relating inertial observers in the relativistic universe is that inertial observers should not observe themselves to be moving at faster than light speed with respect to one another. That is, we should never have m, b such that:

$$IOb(m) \wedge W(m, b, x) \wedge W(m, b, y) \wedge s(y - x) = 0 \tag{6.1}$$

This might be expressed in Coq as:

```
Lemma no_ftl_iob :
  forall m k, forall x y, W m k x ∧ W m k y →
                          x <> y ∧ IOb m ∧ IOb k →
                          |[ y^s -v- x^s ]| < |[ y^t -v- x^t ]|.
```

Andréka et. al. demonstrated that this result follows from SpecRel [4]. The proof proceeds by constructing sets of photons in k's reference frame which imply contradictions in m's reference frame. Their proof requires total ordering to hold over the real numbers, since it proceeds explicitly by construction; we might remove this initial use of total ordering by modifying the theorem statement slightly:

```
Lemma no_ftl_iob' :
  forall m k, forall x y, W m k x ∧ W m k y →
                          x <> y ∧ IOb m ∧ IOb k →
                          |[ y^s -v- x^s ]| >= |[ y^t -v- x^t ]| → False.
```

though decidable equality of real numbers is used later in their proof.

Below, we present the proof of the equality branch of `no_ftl_iob`, i.e. we prove that:

```
Lemma no_luminal_iob :
  forall m k, forall x y, W m k x ∧ W m k y →
                          x <> y ∧ IOb m ∧ IOb k →
                          |[ y^s -v- x^s ]| <> |[ y^t -v- x^t ]|.
```

Intuitively, this is the statement that an inertial observer will not coordinatize another inertial observer to be moving at the speed of light relative to itself. More precisely, since we have not yet demonstrated that bodies must travel on lines, this is the statement that an inertial observer will not coordinatize another at two null separated points.

We have formalized this branch of the proof by Andréka et. al. (ibid) in order to demonstrate the capacity of our event reasoning and real number automation for handling practical proofs. The proof proceeds as follows:

1. Suppose m coordinatizes k at two different x and y, which are null separated. By AxPh, m will also find a photon p at x and y.

2. We find p in both $ev(m, x) = ev(k, x')$ and $ev(m, y) = ev(k, y')$, so k will find that x', y' are null separated.

3. Because $k \in ev(m, x)$ and $k \in ev(m, y)$, we also have $W(k, k, x')$ and $W(k, k, y')$. By AxSf, this means $\vec{x'^s} = \vec{0}$ and $\vec{y'^s} = \vec{0}$. But since x' and y' are null separated, this can only mean that they are both $(0, 0, 0, 0)$. This latter step requires some reasoning over expressions of real numbers, which is handled by `real_crush`.

4. Because x and y are different, the events $ev(m, x)$ and $ev(m, y)$ are different. We have $ev(m, x) = ev(k, x')$ and $ev(m, y) = ev(k, y')$ for some x', y' by AxEv, so we also have $ev(k, x') \neq ev(k, y')$ (under our extensional equality of events).

5. This contradicts what we found in step 3, that $x' = y'$.

The second branch of the proof, that we should not have $||\vec{y^s} - \vec{x^s}|| > |y^t - x^t|$, uses geometric reasoning that we have not yet formalized, so we leave this to future work.

Chapter 7

Real Number Automation

In order to support this development, we constructed automation that handled some goals involving real numbers. In this chapter we describe the automation and approximate its capabilities. We also discuss the process of its development and improvement.

7.1 Automation

7.1.1 real_crush

`real_crush` is an Ltac tactic for solving goals involving equalities and inequalities of real numbers. The operation of `real_crush` may be divided into stages, which proceed as follows:

1. Simplification. Iteratively call:

   (a) `cbn in *`, which invokes an improved version of Coq's `simpl` tactic for reducing portions that reduce computationally under the calculus of constructions.

   (b) `subst`, which is intended to substitute any equalities that render variables redundant.

   (c) `real_simpl`, which reduces real number expressions according to a set of particular equalities. The operation of this tactic is described further in section 7.1.2.

   (d) `real_normalize_bal; real_sum_cycsimpl_g`, which places the expressions in the goal into a particular "balanced" normal form, then attempts to simplify

resulting equalities of sums in the conclusion by removing shared terms from both sides. The operation of `real_normalize_bal` is described further in section 7.1.3; `real_sum_cycsimpl_g` is described further in section 7.1.4.

2. Basic goal solve attempts. These include:

   (a) `repeat (real_normalize_field; subst; real_simpl); field` repeatedly normalizes the expressions in the goal to attempt to isolate monomials, substitutes on the results, and reduces. Once this iterative simplification is done, we see if an invocation of `field` will deal with what's left. This is useful in the proof of the triangle inequality from Cauchy-Schwarz.

   (b) We also attempt `discrR` and `fourier` for solving real number inequalities.

   If these fail to discharge the obligation, we return to a normalized form and attempt `auto` once, before moving onto the inequality proof search portion.

3. `real_ineq_crush`, which normalizes and breaks down inequalities to search for proofs of the original goal. The operation of `real_ineq_crush` is described further in section 7.1.5.

The body of `real_crush` is reproduced below.

```
Ltac real_crush' :=
  repeat (cbn in *;
          subst;
          real_simpl;
          try (real_normalize_bal; real_sum_cycsimpl_g));
  try (repeat (real_normalize_field; subst; real_simpl); field);
  try discrR; try fourier;
  real_normalize_bal; auto.

Ltac real_crush := real_crush'; try real_ineq_crush; auto.
```

Capabilities

We approximate the capabilities of `real_crush`.

`real_crush` can easily dispatch goals where a hypothesis reduces to the goal after normalization. Whether this occurs is up to the capabilities of `real_simpl` and the normalization

tactics (described below) to commute variables appropriately; the user may still have to manually commute when expressions are long and happen to fall into the wrong ordering. Alternatively, expressing the hypothesis and goals in the right forms initially may avoid this. In this development we did not observe having to exercise a great deal of attention as to our expression.

The step described in preparation for `real_crush` can minimally solve simple polynomial systems encountered in our proofs of statements surrounding the triangle inequality.

Our `real_crush`

7.1.2 real_simpl

`real_simpl` reduces expressions of real numbers by applying appropriate rewrites upon matching certain patterns. The following is a list of examples of patterns that are matched and simplified by `real_simpl`:

1. $-0 \rightarrowtail 0$
2. $1 * x \rightarrowtail x$, $x * 1 \rightarrowtail x$
3. sqrt $0 \rightarrowtail 0$
4. $x + -x \rightarrowtail 0$, $-x + x \rightarrowtail 0$
5. $- - x \rightarrowtail x$

This simplification is iterated to bring the expressions to a small form.

As can be seen, `real_simpl` is used to reduce fairly simple patterns. Despite the simplicity of the situations it handles, `real_simpl` affords a great deal of convenience. A single application subsumes what otherwise would be a long manual process of pattern-matching simple observations.

The tactic does not commute expressions, which means that its simplification is local in nature. For instance, $x + y - x$ will not be simplified because it does not contain $x + -x$ syntactically—it would be necessary to commute $y + -x = -x + y$ to obtain this. Improvements in automation could be made to address this in a case-specific manner.

vec_simpl

`real_simpl` also calls upon `vec_simpl` to unfold vector operations appropriately when the vectors have already been `destruct`ed into their components, i.e. are written explicitly in the form `(_, _, _, _)` rather than as a single variable, viz. `v`. `vec_simpl` performs targeted unfolding on explicitly written four-tuples of reals, viz. `v4neg (_, _, _, _)`, under either vector operations or equality.

7.1.3 real_normalize

A number of `real_normalize` tactics are used to bring equalities and inequalities into "normalized" forms that have particular properties. A list of the normalizers used in this development, their intended effects, and their manner of usage follows:

1. `real_normalize_bal` shifts equalities and inequalities so that both sides are sums of terms with no negations, i.e. $x + -y + -z = -w + v + u$ becomes $x + w = v + u + y + z$. This is used in particular to prepare for applications of `real_sum_cycsimpl_g` (section 7.1.4), which is expected to help to simplify sums where identical terms appear on both sides. This way, an equation like $a = x + b - x$ may be simplified to $a = b$.

2. `real_normalize_right` shifts the contents of the equality/inequality to the right-hand side (RHS), leaving a 0 on the left. This is used to prepare for applications of the inequality-solving automation, which performs simplifications that rely on the left-hand side (LHS) of the inequality being 0 (i.e. $0 \leq r^2$ is true).

3. `real_normalize_field` shifts the contents of equalities in hypotheses to the RHS, *excluding a single monomial*, which remains on the left. It also shifts the contents of equalities in the goal to the RHS, leaving a 0 on the left. This is done to permit subst to reduce the number of variables; the ultimate intention of this application is to produce an equality of a form amenable to solution by the `field` tactic.

Broadly, all of the normalizers proceed by pattern matching on sums on either side of an equality or inequality and applying "real sum regularity" lemmas to move terms across the equality or inequality. Real sum regularity lemmas allow the user add the same term to both sides of an equality/inequality.

We illustrate this general methodology with an example. One branch of `real_normalize_right` recognizes equalities where a term other than 0 is on the LHS using the following match statement:

```
match goal with
  ...
  | [ ⊢ ?x = _ ] ⇒
    match x with
      | 0 ⇒ fail 1
      | _ ⇒ apply (Rplus_eq_reg_r (- x));
            rewrite Rplus_opp_r;
            try rewrite Rplus_0_r;
            real_normalize_right
    end
  ...
end.
```

In the case that the LHS is 0, this will take the `fail` branch, which in this instance leaves the inner match statement and moves to try the next case in the overall match. In the case that the LHS x is not actually 0, it will move on to subtract x from both sides using `Rplus_eq_reg_r`, a lemma from the Coq standard library. The statement of `Rplus_eq_reg_r` is:

```
Rplus_eq_reg_r : forall r r1 r2 : R, r1 + r = r2 + r → r1 = r2
```

That is, given a goal of the form $r_1 = r_2$, we can replace it with a goal where r has been added to both sides.

7.1.4 real_sum_cycsimpl_g

`real_sum_cycsimpl_g` is used to simplify equalities and inequalities of sums that have been normalized by `real_normalize_bal`. Specifically, if the same term is found on both sides of an equality or inequality, it is stripped away: $a + c = b + a$ becomes $c = b$ after application of this tactic.

This tactic commutes sums to achieve its effect. In Coq, `a + b` is a convenience notation that unfolds to `Rplus a b`. This notation is left associative, so `a + b + c + d` is implicitly `((a + b) + c) + d`. In this sum, we refer to `a` as the "tail", since it is the deepest embedded, and to `d` as the "front" or "head", since it is exposed for operations like the use of

compatibility lemmas (as described below).

With this understanding, we may summarize the overall operation of the tactic as follows:

1. Find the number of terms n in the RHS of the sum and repeat the following operations n times.

2. Bring the tail of the sum on the RHS to the head. If it exists in both sides of the equality/inequality, it will be brought to the head of both the RHS and left hand side (LHS).

3. Then we can attempt to use a "real sum compatibility" lemma, which allows the user to *remove* the same term from both sides of an equality/inequality. If the term brought to the head of the RHS existed in the LHS, then it will be removed from both sides by this attempted application.

We named this tactic "cycsimpl," short for "cyclical simplification," because the operation of this tactic proceeds by repeatedly "cycling" the tail of the RHS sum to the head.

The body of `real_sum_cycsimpl_g` is reproduced below:

```
Ltac rs_cycsimpl_g sz :=
  let iter := match goal with
                | [ ⊢ context [ _ <= ?x ] ] ⇒ rs_cycsimpl_fronter_g x;
                                               try apply Rplus_le_compat_r; auto;
                                               rs_cycsimpl_cyc_g x
                | [ ⊢ context [ _ < ?x ] ] ⇒ rs_cycsimpl_fronter_g x;
                                             try apply Rplus_lt_compat_r; auto;
                                             rs_cycsimpl_cyc_g x
                | [ ⊢ context [ _ = ?x ] ] ⇒ rs_cycsimpl_fronter_g x;
                                             try apply Rplus_eq_compat_r; auto;
                                             rs_cycsimpl_cyc_g x
                | _ ⇒ idtac
              end in
  match sz with
    | S ?sz' ⇒ iter; rs_cycsimpl_g sz'
    | _ ⇒ iter
  end.

Ltac real_sum_cycsimpl_g :=
  match goal with
    | [ ⊢ context [ _ <= ?sum ] ] ⇒ rs_len sum rs_cycsimpl_g
    | [ ⊢ context [ _ < ?sum ] ] ⇒ rs_len sum rs_cycsimpl_g
    | [ ⊢ context [ _ = ?sum ] ] ⇒ rs_len sum rs_cycsimpl_g
    | _ ⇒ idtac
  end.
```

Because this tactic invokes commutativity to perform reasoning beyond local syntactic comparison, it is relatively more complex than the other tactics discussed, relying on a number of subtactics to achieve its effects.

Our `real_sum_cycsimpl_g` effectively simplifies goals with terms that should cancel, which can sharply reduce proof complexity: once terms are introduced which are shared on both sides, this simplifier removes them both, which enables options for simplification by transformations such as substitution.

The rest of this section describes the construction of its subtactics.

rs_len

`rs\_len` receives a sum term and a tactic. It calls the tactic it is provided with the length it finds in standard continuation-passing style. We are obliged to use continuation-passing style to transfer intermediate computational results in Ltac because Ltac does not support return values. We repeat `rs\_len` below to illustrate precisely what the use of continuation-passing

style in Ltac entails:

```
Ltac rs_len sum tac :=
  match sum with
    | ?sum' + _ ⇒ rs_len sum' ltac:(fun x ⇒ tac (S x))
    | _ ⇒ tac 0%nat
  end.
```

In this way, further recursive calls of `rs_len` are induced to pass to the original tactic a successor. The final call passes in a 0, which undergoes some number of successors equal to the number of recursive calls performed.

rs_cycsimpl_fronter_g

`rs_cycsimpl_fronter_g` is used to move a term from the tail to the head of a sum. It does this by finding the tail of the sum and then passing it to another subtactic `front_Rplus_g`. This tactic operates by detecting when the term it is given appears outside of the head position in a sum expression and applies a series of rewrites based on associativity and commutativity of Rplus to move it one position forwards, leaving the ordering of the rest of the expression intact. Because `front_Rplus_g` embodies the core logic of this tactic, we reproduce it below:

```
Ltac front_Rplus_g x :=
  repeat rewrite ← Rplus_assoc;
  match goal with
    | [ ⊢ context [x + ?y] ] ⇒ rewrite (Rplus_comm x y); front_Rplus_g x
    | [ ⊢ context [(?y + x) + ?z] ] ⇒
      rewrite (Rplus_assoc y x z);
        rewrite (Rplus_comm x z);
        rewrite ← (Rplus_assoc y z x); front_Rplus_g x
    | _ ⇒ idtac
  end.
```

In the latter branch, the operations performed transform the expression as follows:

$$\begin{aligned} & (y + x) + z \\ \rightarrow \quad & y + (x + z) \\ \rightarrow \quad & y + (z + x) \\ \rightarrow \quad & (y + z) + x \end{aligned}$$

thereby shifting x one to the right, while leaving the ordering of y and z in the sum intact.

7.1.5 real_ineq_crush

`real_ineq_crush` is used to solve goals involving inequalities. The tactic relies on the in-built backtracking capability of Ltac to try different approaches to breaking down an inequality.

`real_ineq_crush` returns to the first half of `real_crush` to perform simplifications in the course of backtracking. After this runs, it invokes `real_normalize_right` to prepare inequalities for solving. Then it pattern matches on the goal to select a lemma to apply for backtracking proof search. The following is a list of examples of actions it might take:

1. The goal takes the form $0 < x * x$ for some x: attempt to prove $0 < x$.
2. The goal takes the form $0 \leq x * y$ for some x, y: attempt to prove $0 \leq x$ and $0 \leq y$.
3. The goal takes the form $0 < x + y$ for some x, y: attempt to prove $0 \leq x$ and $0 < y$.

`real_ineq_crush` is a tactic that searches for a proof and may fail. Some cases result in an immediate solution, such as when the inequality takes the form $0 \leq x * x$ for some x. In other cases, such as when the inequality includes a sum, `real_ineq_crush` proceeds by breaking the inequality down into others which might be immediately soluble in similar manner. We have found that inequalities generated by unfolding vector expressions often yield in this way, because these often take the form of sums of square roots or squares, which are particularly amenable to reasoning under this heuristic.

7.1.6 destruct_coords

We also developed a piece of automation, `destruct_coords`, for reducing expressions regarding vectors to expressions regarding real numbers. This tends to increase the complexity of the expressions, but we have found that real number automation often suffices to handle the goals produced. This spares us from writing a redundant set of automation to directly handle expressions regarding vectors and additionally hides the complexity of the expressions generated by unfolding.

`destruct_coords` also uses `vec_simpl` to reduce the results it produces. This tactic was described in section 7.1.2.

7.2 Development

In this section we discuss at a high level our experience with the process of developing and improving automation. Overall, this took the form of iterative improvements to subsume behaviors encountered during manual proof exploration. That is, we would typically produce a proof partially manually with the cooperation of automation; then, we would introduce capabilities to solve the manual section of the proof to the automation.

Generally, we proceed by obtaining requirements for behavior and producing automation that filled out these requirements. Here are a few examples of this process in action. When we observe an expression that should have been simplified automatically, we may introduce the appropriate patterns into `real_simpl`, after which the problem will be solved in all cases. In more complex situations—for instance, when we observed a need to simplify terms from both sides of an equality—we developed larger pieces of automation, viz. `real_sum_cycsimpl_g`. When we observe that our manual proofs typically arrive via the use of a sequence of certain lemmas, we can place these into a backtracking solver; generally, as long as the options taken manually can eventually be taken by the solver, the solver will suffice to handle this class of problems. Effective use of automation requires effective rewrite methods for placing expressions into a normal form, as we developed; this is because it is otherwise difficult to predict whether a lemma can be applicable.

Chapter 8

Related Work and Conclusion

8.1 Related Work

There has been previous work in formally verifying physical reasoning. Most proximately, Stannett and Németi have produced an Isabelle development verifying that there are no faster-than-light inertial observers under SpecRel, towards the end of reasoning about hypercomputation [12]. Isabelle has also been used to formalize optics, in particular geometrical optics [11] and electromagnetic optics [8]. These developments were motivated by safety concerns surrounding optical equipment, for instance as used in laser surgery. These developments used Isabelle to express logical reasoning, exporting computations to Mathematica.

In contrast, our work is a formalization of SpecRel under a different logic—that of Coq—which uses custom automation rather than external or library solvers. We have demonstrated that Coq's logic and automation mechanism are also usable as a basis for formalized physical reasoning.

8.2 Summary

In this section, we summarize some measurements on the Coq development we have presented. The Coq development comprises three files:

`PropSet.v` occupies 48 lines with a formalization of the extensionally defined proposi-

tional sets used to represent spacetime events throughout the development. `PropSet.v` also contains the proof that the extensional equality over these sets constitutes an equivalence relation, and registers the relation with Coq for use with the tactic `rewrite`.

`FourVector.v` occupies 555 lines. `FourVector.v` contains the majority of the proof automation, including the tactic `real_crush` and its subtactics as described in Chapter 7; this occupies 391 lines of the file. The remainder consists of definitions for four-vectors, whose type in our development is denoted `v4`, and proofs of lemmas regarding `v4`s, such as decidable equality and the proof that triangle inequality follows from Cauchy-Schwarz.

`SpecRel.v` occupies 415 lines, containing the definitions of the SpecRel axioms in Coq as well as the proofs of the theorems discussed in this document. The whole development occupies 1018 lines. On a computer running an Intel Core i7-4710HQ at 2.50 GHz, verification of the entire development takes somewhat under one minute.

The tactic `real_crush` is used in 30 locations, supplying goal solutions at every location. We have instrumented `real_simpl` to take an observation on the extent to which it assists the user during theorem proving. In the course of verifying the proof script, `real_simpl` is invoked 3553 times, successfully making 1590 simplifications (as measured by patterns matched). These would otherwise have to be manually entered by the programmer. This gives an impression of the amount of programmer effort saved by our automation. Further engineering may yet increase the ability of automation to reason about our goals.

8.3 Conclusion

We have presented a Coq development based on a set of axioms SpecRel from Andréka et. al. describing the relativistic universe, towards the goal of demonstrating the usability of Coq to formalize physical reasoning. In particular, we have demonstrated that it is possible to adapt Coq's logic to permit reasoning about axioms and representing proofs from the literature in Coq. We have also discussed the effectiveness of Coq's proof automation towards lowering programmer effort.

We would like to conclude with a few high level observations about our use of Coq in physical reasoning.

1. Because both textbooks and literature make use of classical logic in physical reasoning, including the use of extensionality to define objects and the use of excluded middle in proof, proof efforts using Coq for formalizing physical reasoning should best assume classical logic (barring a direct interest in the use of constructive mathematics in physics).

2. The theory we have formalized in this work is not sufficient to solve typical textbook problems. We lack, for instance, foundational results such as Poincaré invariance. The proofs automation required to solve problems at a higher level of abstraction after these results are introduced and become the main objects of theoretical discussion may appear quite different from what we have presented here.

3. Relatedly, the axiom system selected is central to informing the reasoning performed; proving properties about objects not explicitly represented in the axiom system requires work in order to construct means of reasoning about these objects.

4. Analytical geometry, i.e. geometry done in terms of coordinates and equations, seems generally easier than non-analytical geometry to automatically reason about, though this may appear to be the case because it was eased by our real number automation.

5. Examining the proofs we formalized in our development, written physical reasoning often uses a forward reasoning which takes the form: "We have A. Because A, B. Because B, C..." It is difficult to formalize proofs of this form in Coq because Coq's tactic language proceeds by backwards reasoning. This is reflected by the extensive use of `assert` in our proof scripts. Another way to observe this is that physical reasoning requires the use of ingenuity in finding an object or set of objects having the appropriate properties to demonstrate a desired fact, as when structures of photons are constructed to demonstrate a particular property. It is hard to cleanly express these externally introduced pieces of information in Coq because `assert` disrupts the flow of a proof. It is also difficult to write automation to cope with this.

6. As such, we should not see the hand-written automation explored in this work as mechanisms for accelerating theorem proving proper, beyond the level of assistance

supplied by calculational tools. However, Coq's automation mechanism remains important in assisting with formalization, which would be significantly more difficult without automation. The goals solved by automation could be solved by hand, but producing their rigorous proofs to Coq would require significantly more effort. In this way, we might see the role of Coq's automation more to be as an assistive mechanism for facilitating the transfer of proofs from paper to computer certification.

Bibliography

[1] AD Aleksandrov. The space-time of the theory of relativity. *Helvetica Physica Acta*, 4(Supplement):44–45, 1956.

[2] Hajnal Andréka, Judit X Madarász, and István Németi. Logical axiomatizations of space-time. samples from the literature. In *Non-Euclidean geometries*, pages 155–185. Springer, 2006.

[3] Hajnal Andréka, Judit X Madarász, and István Németi. Logic of space-time and relativity theory. *Handbook of spatial logics*, pages 607–711, 2007.

[4] Hajnal Andréka, Judit X Madarász, István Németi, and Gergely Székely. A logic road from special relativity to general relativity. *Synthese*, 186(3):633–649, 2012.

[5] Fabrizio Cacciafesta. An observation about a theorem of ad alexandrov concerning lorentz transformations. *Journal of Geometry*, 18(1):5–8, 1982.

[6] Adam Chlipala. *Certified Programming with Dependent Types*. MIT Press, 2011.

[7] Robert Goldblatt. *Orthogonality and spacetime geometry*. Springer Science & Business Media, 2012.

[8] Sanaz Khan-Afshar, Umair Siddique, Mohamed Yousri Mahmoud, Vincent Aravantinos, Ons Seddiki, Osman Hasan, and Sofiène Tahar. Formal analysis of optical systems. *Mathematics in Computer Science*, 8(1):39–70, 2014.

[9] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[10] Alfred Arthur Robb. *A theory of time and space*. University Press Cambridge, 1914.

[11] Muhammad Umair Siddique. *Formal Analysis of Geometrical Optics using Theorem Proving*. PhD thesis, Concordia University Montréal, Québec, Canada, 2015.

[12] Mike Stannett and István Németi. Using isabelle to verify special relativity, with application to hypercomputation theory. *arXiv preprint arXiv:1211.6468*, 2012.

[13] Gergely Székely. First-order logic investigation of relativity theory with an emphasis on accelerated observers, 2010.

[14] Erik Christopher Zeeman. Causality implies the lorentz group. *Journal of Mathematical Physics*, 5(4):490–493, 1964.