Can We Believe What We Have Proved? – Automating Gödel's Ontological Proof of God's Existence in PowerEpsilon

Ming-Yuan Zhu

CoreTek Systems
11th Floor, 1102-1103, Qingyun Contemporary Building
43 North Third Ring Road
Beijing 100086
People's Republic of China
E-Mail: zhumy@coretek.com.cn

January 13, 2019

Abstract

We formalize Gödel's Ontological Argument, the proof of the existence of God, in **PowerEpsilon** and verify its correctness. The formalization raises delicate questions in formal logic and provides an opportunity to show how these are handled, soundly and efficiently, by the higher-order logic of **PowerEpsilon** and its mechanized support. The simplicity of the Argument, coupled to its bold conclusion, raise interesting issues on the interpretation and application of formal methods in the real world.

Keywords Ontological argument, formal methods; mathematical theorem proof development systems.

Contents

1 Introduction

This work is just for fun. A fun with computer and mathematical logic. It is done during my spare time as a hobby project and has nothing to do with religions.

The Ontological Argument is a proof of the existence of God. This is a topic that almost everyone, believer and unbeliever alike, finds deeply interesting. Formulation and verification of the Argument in a mechanized theorem prover is therefore an excellent candidate for Fun With Formal Methods. Furthermore, the formalization raises subtle issues in mathematical logic and thereby serves as a useful pedagogical vehicle to introduce students to these issues and how they are resolved in a mechanized system such as **PowerEpsilon**.

2 PowerEpsilon

This work was carried out and assisted by a mechanical theorem proof development system **PowerEpsilon** [17]. The semantic specification and analysis can be used as a basis to guide a realistic implementation.

PowerEpsilon [17, 18], is a strongly-typed polymorphic functional programming language based on Martin-Löf's type theory [12] and the Calculus of Constructions [8]. The system can be used as both a programming language with a very rich set of data structures and a metalanguage for formalizing constructive mathematics. The system has been implemented using the software development system **AUTOSTAR** constructed by author [16].

PowerEpsilon is a proof checker much similar to other mechanical proof checkers, such as **LCF** [10] and **Nuprl** [7], which are completely formal user-controlled systems. However, **PowerEpsilon** is more powerful than **LCF** and **Nuprl**, in which the equality and induction rules for arbitrary inductive types are definable. Since **PowerEpsilon** is composed of only a few constructs, it is a useful tool for studying and giving semantics to programming languages.

3 Formal Modal Logic in PowerEpsilon

Formal modal logic were developed to make precise the mathematical properties of differing conceptions of such notions such as possibility, necessity, belief, knowledge and temporal progression which arise in philosophy and natural languages. In the last thirty year modal logics have emerged as useful tools for expressing essential ideas in computer science and artificial intelligence.

Formally, modal logic is an extension of classical propositional or predicate logic. The language of classical logic is enriched by adding of new "modal operators". The standard basic operators are traditionally denoted by $\Box$ and $\Diamond$. Syntactically, they can be viewed as new unary connectives.

3.1 Possible Worlds and Individuals

The semantics for modal logic are usually given as follows: First we define a frame, which consists of a non-empty set, **G**, whose members are generally called possible worlds, and a binary relation, **R**, that holds (or not) between the possible worlds of **G**. This binary relation is called the accessibility relation. For example,`w` **R** `u` means that the world `u` is accessible from world `w`. That is to say, the state of affairs known as `u` is a live possibility for `w`. This gives a pair, $\langle \mathbf{G}, \mathbf{R} \rangle$. Some formulations of modal logic also include a constant term in **G**, conventionally called "the actual world", which is often symbolized as `w*`.

```
dec PWorld : Prop;

dec Indiv : Prop;
```

3.2 Properties of Logical Operators

```
dec ExclMiddleRule0 :
  !(T : Type(0))
   @(Or, T, @(Neg, T));

dec NegNegPredLem :
  !(X : Prop)
```

```
    [@(Neg, @(Neg, X)) -> X];

dec ABCImpNegLem :
  !(A : Type(0), B : Type(0), C : Type(0))
   [[A -> B -> C] -> [@(Neg, C) -> @(Or, @(Neg, A), @(Neg, B))]];
```

3.3 Modal Logical Operators

3.3.1 Modal Logical Null, Truth and Negation

```
def mNull =
  \(x : Indiv, w : PWorld)
   Null;

def mNeg =
  \(P : [Indiv -> PWorld -> Prop])
  \(x : Indiv, w : PWorld)
   @(Neg, @(P, x, w));

def mTrue = @(mNeg, mNull);

dec mNegNegLem :
  !(X : [Indiv -> PWorld -> Prop])
   @(Equal, [Indiv -> PWorld -> Prop], @(mNeg, @(mNeg, X)), X);

dec mNegNullTrueLem : @(Equal, [Indiv -> PWorld -> Prop], @(mNeg, mNull), mTrue);

dec mNegTrueNullLem : @(Equal, [Indiv -> PWorld -> Prop], @(mNeg, mTrue), mNull);
```

3.3.2 Modal Logical $\wedge$, $\vee$, $\rightarrow$ and $\equiv$

```
def mAnd =
  \(x : Indiv, w : PWorld)
  \(P : [Indiv -> PWorld -> Prop], Q : [Indiv -> PWorld -> Prop])
   @(And, @(P, x, w), @(Q, x, w));

def mOr =
  \(w : PWorld)
  \(P : [PWorld -> Prop], Q : [PWorld -> Prop])
   @(Or, @(P, w), @(Q, w));

def mImp =
  \(x : Indiv, w : PWorld)
  \(P : [Indiv -> PWorld -> Prop], Q : [Indiv -> PWorld -> Prop])
   [@(P, x, w) -> @(Q, x, w)];

def mEqv =
  \(w : PWorld)
  \(P : [PWorld -> Prop], Q : [PWorld -> Prop])
   @(And, [@(P, w) -> @(Q, w)], [@(Q, w) -> @(P, w)]);
```

3.4 Possibilist Qualification $\forall$ and $\exists$

```
def mForall =
  \(P : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
   !(x : Indiv)
    @(P, x, w);

def mExist =
  \(P : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
   ?(x : Indiv)
    @(P, x, w);
```

3.5 Actualist Qualification $\forall^E$ and $\exists^E$

```
dec existsAt : [Indiv -> PWorld -> Prop];

def mForallAct =
  \(P : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
   !(x : Indiv)
    [@(existsAt, x, w) -> @(P, x, w)];

def mExistAct =
  \(P : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
   ?(x : Indiv)
    @(And, @(existsAt, x, w), @(P, x, w));
```

3.6 Frame Conditions and Accessibility Relation

The different systems of modal logic are distinguished by the properties of their corresponding accessibility relations. There are several systems that have been espoused (often called frame conditions). An accessibility relation is:

Reflexive iff `w` **R** `w`, for every `w` in **G**.

Symmetric iff `w` **R** `u` implies `u` **R** `w`, for all `w` and `u` in **G**.

Transitive iff `w` **R** `u` and `u` **R** `q` together imply `w` **R** `q`, for all `w`, `u`, `q` in **G**.

Serial iff, for each `w` in **G** there is some `u` in **G** such that `w` **R** `u`.

Euclidean iff, for every `u`, `t`, and `w`, `w` **R** `u` and `w` **R** `t` implies `u` **R** `t` (note that it also implies: `t` **R** `u`)

Functional iff, for every `u`, `v`, and `w`, `w` **R** `u` and `w` **R** `v` implies `u = v`.

Dense iff, for each `w` and `v` in **G** there is some `u` in **G** such that `w` **R** `u` and `u` **R** `v`.

In terms of **PowerEpsilon**, we define **R** as `ActRel`.

```
dec ActRel : [PWorld -> PWorld -> Prop];
```

Assuming that `ActRel` is reflexive functional,

```
dec ARelFunctionalLem :
  !(w : PWorld, v : PWorld, u : PWorld)
   [@(ActRel, w, v) -> @(ActRel, w, u) -> @(Equal, PWorld, v, u)];
```

```
dec ARelReflLem :
  !(w : PWorld)
   @(ActRel, w, w);
```

we will then have the following properties:

```
dec ARelFunEqLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) -> @(Equal, PWorld, w, v)];

dec ARelSymmLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) -> @(ActRel, v, w)];

dec ARelTranLem :
  !(w : PWorld, u : PWorld, v : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, u, v) ->
    @(ActRel, w, v)];

dec ARelSymmTranLem :
  !(w : PWorld, u : PWorld, v : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, w, v) ->
    @(ActRel, u, v)];

dec ARelSerialLem :
  !(w : PWorld)
   ?(u : PWorld)
    @(ActRel, w, u);

dec ARelEuclideanLem :
  !(u : PWorld, t : PWorld, w : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, w, t) ->
    @(ActRel, u, t)];

dec ARelShiftReflLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) -> @(ActRel, v, v)];

dec ARelDenseLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) ->
    ?(u : PWorld) @(And, @(ActRel, w, u), @(ActRel, u, v))];

dec existsAtARelLem :
  !(x : Indiv)
  !(w : PWorld, u : PWorld, v : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, w, v) ->
    @(existsAt, x, u) ->
    @(existsAt, x, v)];

dec existsAtARelLem2 :
  !(x : Indiv)
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) ->
    @(existsAt, x, w) ->
    @(existsAt, x, v)];
```

3.7 Axiomatic Systems

- N, Necessitation Rule: If p is a theorem (of any system invoking N), then $\Box p$ is likewise a theorem.
- K, Distribution Axiom: $\Box(p \to q) \to (\Box p \to \Box q)$.
- T, Reflexivity Axiom: $\Box p \to p$ (If p is necessary, then p is the case.)

Other well-known elementary axioms are:

- D: $\Box p \to \Diamond p$.
- M: $\Box p \to p$.
- 4: $\Box p \to \Box\Box p$.
- B: $p \to \Box\Diamond p$.
- 5: $\Diamond p \to \Box\Diamond p$.
- CD: $\Diamond p \to \Box p$.
- $\Box$M: $\Box(\Box p \to p)$.
- C4: $\Box\Box p \to \Box p$.
- C: $\Diamond\Box p \to \Box\Diamond p$.

3.7.1 Necessitation Rule

```
dec NecessRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(P, w) -> @(mAlways, P, w)];
```

3.7.2 Distribution Rule

```
dec DistrRule :
  !(P : [PWorld -> Prop], Q : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, \(u : PWorld) [@(P, u) -> @(Q, u)], w) ->
    @(mAlways, P, w) ->
    @(mAlways, Q, w)];
```

3.7.3 Reflexive Rule

```
dec ReflRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, P, w) -> @(P, w)];
```

3.7.4 Transitive Rule

```
dec FourRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, P, w) -> @(mAlways, @(mAlways, P), w)];
```

3.7.5 Symmetric Rule

```
dec BRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(P, w) -> @(mAlways, @(mSometime, P), w)];
```

3.7.6 Serial Rule

```
dec DRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, P, w) -> @(mSometime, P, w)];
```

3.7.7 Euclidean Rule

```
dec FiveRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mSometime, P, w) -> @(mAlways, @(mSometime, P), w)];
```

3.7.8 Functional Rule

```
dec CDRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mSometime, P, w) -> @(mAlways, P, w)];
```

3.7.9 Shift Reflexive Rule

```
dec BMRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   @(mAlways, \(u : PWorld) [@(mAlways, P, u) -> @(P, u)], w);
```

3.7.10 Dense Rule

```
dec C4Rule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, @(mAlways, P), w) -> @(mAlways, P, w)];
```

3.7.11 Convergent Rule

```
dec CRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mSometime, @(mAlways, P), w) -> @(mAlways, @(mSometime, P), w)];
```

3.8 Modal Operators

3.8.1 Definitions of Modal Logic Operators $\Box$ and $\Diamond$

```
def mAlways =
  \(P : [PWorld -> Type(0)])
  \(w : PWorld)
   !(v : PWorld)
    [@(ActRel, w, v) -> @(P, v)];

def mSometime =
  \(P : [PWorld -> Type(0)])
  \(w : PWorld)
   ?(v : PWorld)
    @(And, @(ActRel, w, v), @(P, v));
```

Definition 3.1 *$X \cap Z$ is defined as $\Box\forall^E[X \leftrightarrow \forall Y[Z(Y) \to Y]]$.*

```
def mJoint =
  \(X : [Indiv -> PWorld -> Prop], Z : [[Indiv -> PWorld -> Type(0)] -> PWorld -> Prop])
   !(w : PWorld)
    @(mAlways,
      @(mForall,
        \(x : Indiv, v : PWorld)
         @(And,
           [@(X, x, v) -> !(Y : [Indiv -> PWorld -> Prop]) [@(Z, Y, v) -> @(Y, x, v)]],
           [!(Y : [Indiv -> PWorld -> Prop]) [@(Z, Y, v) -> @(Y, x, v)] -> @(X, x, v)])),
      w);
```

Definition 3.2 *$X \Rightarrow Y$ is defined as $\Box\forall^E[X \to Y]$.*

```
def mArrow =
  \(X : [Indiv -> PWorld -> Prop], Y : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
   @(mAlways,
     @(mForallAct,
       \(x : Indiv, v : PWorld)
        [@(X, x, v) -> @(Y, x, v)]),
     w);
```

3.8.2 Properties of Modal Logic Operators

Lemma 3.1 $\forall\Phi[\Box\Phi \to \Phi]$.

```
dec mAlwaysElimLem :
  !(w : PWorld)
  !(Q : [PWorld -> Type(0)])
   [@(mAlways, Q, w) -> @(Q, w)];
```

Lemma 3.2 $\forall\Phi[\Phi \to \Diamond\Phi]$.

```
dec mSometimeIntrLem :
  !(w : PWorld)
  !(Q : [PWorld -> Type(0)])
   [@(Q, w) -> @(mSometime, Q, w)];
```

Lemma 3.3 $\forall\varphi[\neg\Diamond\exists^E\varphi \to \Box\forall^E\neg\varphi]$.

```
dec SomeTimeNegLem :
  !(X : [Indiv -> PWorld -> Prop])
  !(w : PWorld)
   [@(Neg, @(mSometime, @(mExistAct, X), w)) ->
    @(mAlways, @(mForallAct, @(mNeg, X)), w)];
```

For $\Rightarrow$, we have the following lemmas.

Lemma 3.4 $\forall\varphi,\phi[\neg[\varphi \Rightarrow \phi] \to \Diamond\exists^E(\varphi \wedge \neg\phi)]$.

```
dec mNegArrowLem :
  !(X : [Indiv -> PWorld -> Prop], Y : [Indiv -> PWorld -> Prop])
  !(w : PWorld)
   [@(Neg, @(mArrow, X, Y, w)) ->
    @(mSometime,
      @(mExistAct,
        \(x : Indiv, v : PWorld)
         @(And, @(X, x, v), @(Neg, @(Y, x, v)))),
      w)];
```

Lemma 3.5 $\forall\varphi[\varphi \Rightarrow \varphi]$.

```
dec IdArrowLem :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   @(mArrow, X, X, w);
```

Lemma 3.6 $\forall\varphi[\bot \Rightarrow \varphi]$.

```
dec mNullTrueArwLem :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   @(mArrow, mNull, X, w);
```

Lemma 3.7 $\forall \alpha, \beta, \gamma[\alpha \Rightarrow \beta \wedge \beta \Rightarrow \gamma \rightarrow \alpha \Rightarrow \gamma]$.

```
dec mArrowTranLem :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop], Y : [Indiv -> PWorld -> Prop], Z : [Indiv -> PWorld -> Prop])
   [@(mArrow, X, Y, w) ->
    @(mArrow, Y, Z, w) ->
    @(mArrow, X, Z, w)];
```

Lemma 3.8 $\Box\forall^E\top$.

```
dec mTrueLem :
  !(w : PWorld)
   @(mAlways, @(mForallAct, mTrue), w);
```

3.9 Meta-Logical Predicates

```
def mValid =
  \(P : [PWorld -> Prop])
   !(w : PWorld)
    @(P, w);

def mSat =
  \(P : [PWorld -> Prop])
   ?(w : PWorld)
    @(P, w);

def mCSat =
  \(P : [PWorld -> Prop])
   ?(w : PWorld)
    @(Neg, @(P, w));

def mInvalid =
  \(P : [PWorld -> Prop])
   !(w : PWorld)
    @(Neg, @(P, w));
```

4 Original Version of Gödel’s Ontological Argument

4.1 An Informal Description of Gödel’s Ontological Argument

Gödel’s ontological proof is a formal argument by the mathematician Kurt Gödel (1906C1978) for God’s existence. More precisely, it presupposes the notion of positive and negative properties, and proves the necessary existence of an object which each positive property, but no negative property, applies to.

Gödel defines God as a being who possesses all positive properties and states a few reasonable (but debatable) axioms that such properties should satisfy [9, 14]. The overall idea of Gödel's proof is in the tradition of Anselm's argument, who defined God as an entity of which nothing greater can be conceived. Anselm argued that existence in the actual world would make such an assumed being even greater (more perfect), hence, by definition, God must exist. However, for Anselm existence was treated as a predicate and the possibility of God's existence was assumed as granted. These issues were criticized by Kant and Leibniz, respectively, and they were addressed in the work of Gödel.

The whole proof is outlined in Table 1.

4.2 Gödel's God in PowerEpsilon

Gödel's variant of the ontological argument [9] is a direct descendant of Leibniz's, which in turn derives from Descartes'. All these arguments have a two-part structure:

1. prove that God's existence is possible, and
2. prove that God's existence is necessary, if possible.

The main conclusion (God's necessary existence) then follows from (1) and (2), either by modus ponens (in non-modal contexts) or by invoking some axioms of modal logic (notably, but not necessarily as we will see, the so-called modal logic system).

4.2.1 Proving that God's Existence is Possible

We start out with definition Definition 1 (`Godlike`) and axioms Axiom 1 - Axiom 3.

Definition 1 Being Godlike is equivalent to having all positive properties.

Axiom 1 Exactly one of a property or its negation is positive.

Axiom 2 Any property entailed by a positive property is positive.

Axiom 3 The combination of any collection of positive properties is itself positive.

```
dec Positive : [[Indiv -> PWorld -> Type(0)] -> PWorld -> Prop];

def pos =
  \(Z : [[Indiv -> PWorld -> Prop] -> PWorld -> Prop])
   !(X : [Indiv -> PWorld -> Prop])
   !(w : PWorld)
    [@(Z, X, w) -> @(Positive, X, w)];
```

Definition 4.1 $\mathbf{G}(x) \equiv \forall\varphi[\mathbf{P}(\varphi) \to \varphi(x)]$.

```
def GodLike =
  \(x : Indiv, w : PWorld)
   !(Y : [Indiv -> PWorld -> Prop])
    [@(Positive, Y, w) -> @(Y, x, w)];
```

Axiom 4.1 $\forall\varphi[\mathbf{P}(\neg\varphi) \leftrightarrow \neg\mathbf{P}(\varphi)]$.

```
dec Axiom1 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Positive, @(mNeg, X), w) -> @(Neg, @(Positive, X, w))];

dec Axiom1b :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Neg, @(Positive, X, w)) -> @(Positive, @(mNeg, X), w)];
```

Axiom 1 Either a property or its negation is positive, but not both:

$$\forall\varphi[\mathbf{P}(\neg\varphi) \leftrightarrow \neg\mathbf{P}(\varphi)]$$

Axiom 2 A property necessarily implied by a positive property is positive:

$$\forall\varphi\forall\psi[\mathbf{P}(\varphi) \wedge [\varphi \Rightarrow \psi] \rightarrow \mathbf{P}(\psi)]$$

Theorem 1 Positive properties are possibly exemplified:

$$\forall\varphi[\mathbf{P}(\varphi) \rightarrow \Diamond\exists\varphi(x)]$$

Definition 1 A God-like being possesses all positive properties:

$$\mathbf{G}(x) \equiv \forall\varphi[\mathbf{P}(\varphi) \rightarrow \varphi(x)]$$

Axiom 3 The property of being God-like is positive:

$$\mathbf{P}(\mathbf{G})$$

Theorem 2 Possibly, a God-like being exists:

$$\Diamond\exists x\ \mathbf{G}(x)$$

Axiom 4 Positive properties are necessarily positive:

$$\forall\varphi[\mathbf{P}(\varphi) \rightarrow \Box\mathbf{P}(\varphi)]$$

Definition 2 An essence of an individual is a property possessed by it and necessarily implying any of its properties:

$$\varphi\ \mathbf{ess}\ x \equiv \forall\phi[\phi(x) \rightarrow \Box\forall y[\varphi(y) \rightarrow \phi(y)]]$$

Theorem 3 Being God-like is an essence of any God-like being:

$$\forall x[\mathbf{G}(x) \rightarrow \mathbf{G}\ \mathbf{ess}\ x]$$

Definition 3 Necessary existence of an individual is the necessary exemplification of all its essences:

$$\mathbf{NE}(x) \equiv \forall\varphi[\varphi\ \mathbf{ess}\ x \rightarrow \Box\exists y\ \varphi(y)]$$

Axiom 5 Necessary existence is a positive property:

$$\mathbf{P}(\mathbf{NE})$$

Theorem 4 If a God-like being exists, then necessarily a God-like being exists:

$$\exists x\ \mathbf{G}(x) \rightarrow \Box\exists y\ \mathbf{G}(y)$$

Theorem 5 If possibly a God-like being exists, then necessarily a God-like being exists:

$$\Diamond\exists x\ \mathbf{G}(x) \rightarrow \Box\exists y\ \mathbf{G}(y)$$

Theorem 6 Necessarily, a God-like being exists:

$$\Box\exists x\ \mathbf{G}(x)$$

Table 1: The original version of Gödel's ontological argument

Axiom 4.2 $\forall\varphi,\phi[\mathbf{P}(\varphi) \wedge (\varphi \Rightarrow \phi) \rightarrow \mathbf{P}(\phi)]$.

`Axiom2` indicates that, if φ is positive, then any property that φ necessitates must also be positive.

```
dec Axiom2 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop], Y : [Indiv -> PWorld -> Prop])
   [@(Positive, X, w) ->
    @(mArrow, X, Y, w) ->
    @(Positive, Y, w)];
```

Axiom 4.3 $\forall Z,\varphi[\text{pos}(Z) \wedge (\varphi \cap Z) \rightarrow \mathbf{P}(\varphi)]$.

```
dec Axiom3 :
  !(Z : [[Indiv -> PWorld -> Prop] -> PWorld -> Prop], X : [Indiv -> PWorld -> Prop])
   [@(And, @(pos, Z), @(mJoint, X, Z)) ->
    !(w : PWorld) @(Positive, X, w)];
```

Lemma 4.1 $\forall\varphi[\mathbf{P}(\varphi) \rightarrow \neg\mathbf{P}(\neg\varphi)]$.

```
dec AuxLem1 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Positive, X, w) -> @(Neg, @(Positive, @(mNeg, X), w))];
```

Lemma 4.2 $\forall\varphi[\neg\mathbf{P}(\neg\varphi) \rightarrow \mathbf{P}(\varphi)]$.

```
dec AuxLem2 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Neg, @(Positive, @(mNeg, X), w)) -> @(Positive, X, w)];
```

From Axiom 1 and Axiom 2 follows theorem Theorem 1:

Theorem 1 Every positive property is possibly instantiated (if a property P is positive, then it is possible that some being has property P).

Theorem 4.1 $\forall\varphi[\mathbf{P}(\varphi) \rightarrow \Diamond\exists\varphi]$.

```
dec Theorem1 :
  !(X : [Indiv -> PWorld -> Prop])
  !(w : PWorld)
   [@(Positive, X, w) ->
    @(mSometime, @(mExistAct, X), w)];
```

If a property φ is positive, it must be possibly instantiated, since a property that is not possibly instantiated necessitates every property (vacuously), and then, if this impossible property were positive, by `Axiom2`, it would follow that every property is positive, which is clearly ruled out by `Axiom1`.

From Definition 1 and Axiom 3 follows theorem Theorem 2:

Theorem 2 Being `Godlike` is a positive property.

Theorem 4.2 **P(G)**.

```
dec Theorem2 :
  !(w : PWorld)
   @(Positive, GodLike, w);
```

`Theorem2` asserts that **G** is positive. This makes good sense under both interpretations of positivity: if no positive property entails any negativity or privation, then G (which is, in effect, the infinitary conjunction or intersection of all the positive properties) must also be potentially negation-free. Similarly, to possess all the properties compatible with perfection is surely itself compatible with perfection.

To prove `Theorem2`, we need the following lemmas.

```
dec PosGodLikeAuxLem :
  !(w : PWorld)
   let Z = \(X : [Indiv -> PWorld -> Prop], v : PWorld)
            @(Neg, @(Positive, @(mNeg, X), v)),
       X = GodLike in
     @(Or,
       @(Positive, GodLike, w),
       @(And, @(And, @(pos, Z), @(mJoint, X, Z)), @(Neg, @(Positive, X, w))));

dec PosGodLikeLem :
  !(w : PWorld)
   ?(Z : [[Indiv -> PWorld -> Prop] -> PWorld -> Prop], X : [Indiv -> PWorld -> Prop])
    @(Or,
      @(Positive, GodLike, w),
      @(And, @(And, @(pos, Z), @(mJoint, X, Z)), @(Neg, @(Positive, X, w))));
```

From Theorem 1 and Theorem 2 follows theorem Theorem 3:

Theorem 3 Being Godlike is possibly instantiated.

Theorem 4.3 $\Diamond\exists x\ \mathbf{G}(x)$.

```
dec Theorem3 :
  !(w : PWorld)
   @(mSometime, @(mExistAct, GodLike), w);
```

4.2.2 Proving that God's Existence is Necessary, If Possible

Definition 2 A property E is the essence of an individual x iff x has E and all of x's properties are entailed by E.

Axiom 4 Positive (negative) properties are necessarily positive (negative).

```
dec Axiom4 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Positive, X, w) -> @(mAlways, @(Positive, X), w)];
```

Definition 4.2 φ **ess** $x \ \equiv\ \forall\phi[\phi(x) \to (\varphi \Rightarrow \phi)]$.

```
def essInd =
  \(X : [Indiv -> PWorld -> Prop])
  \(x : Indiv)
  \(w : PWorld)
   !(Y : [Indiv -> PWorld -> Prop])
    [@(Y, x, w) -> @(mArrow, X, Y, w)]);
```

From Axiom 1 and Axiom 4 (using definitions Definition 1 and Definition 2) follows:

Theorem 4 Being `Godlike` is an essential property of any `Godlike` individual.

Definition 3 Necessary existence of an individual is the necessary instantiation of all its essences.

Axiom 5 Necessary existence is a positive property.

Theorem 4.4 $\forall x\, \mathbf{G}(x) \rightarrow \mathbf{G}(x)$ **ess** x.

```
dec Theorem4 :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) -> @(essInd, GodLike, x, w)];
```

Lemma 4.3 $\forall x[\mathbf{G}(x) \rightarrow \forall\phi[\phi(x) \rightarrow (G \Rightarrow \phi)]]$.

```
dec essIndLem :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) ->
    !(Y : [Indiv -> PWorld -> Prop])
     [@(Y, x, w) -> @(mArrow, GodLike, Y, w)]];
```

Definition 4.3 $\mathbf{NE}(x) \equiv \forall\phi[\phi$ **ess** $x \rightarrow (\Box\exists^E\phi)]$.

```
def NE =
  \(x : Indiv)
  \(w : PWorld)
   @(And,
     !(X : [Indiv -> PWorld -> Prop])
      [@(essInd, X, x, w) -> @(mAlways, @(mExistAct, X), w)],
     True);
```

Finally, Gödel assumes that necessary existence in this sense is a positive property, from which it follows that Godlikeness is necessarily instantiated, and, if we assume axiom T of modal logic, that Godlikeness is actually instantiated.

Axiom 4.4 $\mathbf{P}(\mathbf{NE})$.

```
dec Axiom5 :
  !(w : PWorld)
   @(Positive, NE, w);
```

From Theorem 4 and Axiom 5 (using Definition 1, Definition 2, Definition 3) follows theorem Theorem 5:

Theorem 5 (The property of) being `Godlike`, if instantiated, is necessarily instantiated.

Theorem 4.5 $\Diamond\exists^E x\, \mathbf{G}(x) \rightarrow \Box\exists^E x\, \mathbf{G}(x)$.

```
dec Theorem5 :
  !(w : PWorld)
   [@(mSometime, @(mExistAct, GodLike), w) -> @(mAlways, @(mExistAct, GodLike), w)];
```

And finally from Theorem 3, Theorem 5 (together with some implicit modal axioms) the existence of (at least a) God follows:

Theorem 6 Being `Godlike` is actually instantiated.

Theorem 4.6 $\Box\exists^E x\,\mathbf{G}(x)$.

```
dec Theorem6 :
  !(w : PWorld)
   @(mAlways, @(mExistAct, GodLike), w);
```

Corollary 4.1 $\forall \mathtt{E}, \mathtt{P}\ \forall x\ \mathtt{E}\ \mathbf{ess}\ x \wedge \mathtt{P}(x) \rightarrow E \Rightarrow \mathtt{P}$.

```
dec Corollary1 :
  !(E : [Indiv -> PWorld -> Prop], P : [Indiv -> PWorld -> Prop])
  !(x : Indiv, w : PWorld)
   [@(essInd, E, x, w) ->
    @(P, x, w) ->
   @(mArrow, E, P, w)];
```

4.2.3 Modal Collapse

The second part of Gödel's argument features some philosophically controversial, implicit commitments, such as modal collapse (everything that is the case is necessarily the case), which directly attacks our self-understanding as agents having a free will; and the assertion that whatever is possibly necessarily the case is thereby actually the case.

The modal collapse, originally detected by Sobel [15], has been confirmed as a side-effect of Gödel's argument in various previous experiments with automated theorem provers. In still other, more controversial, words, there is no free will. Roughly speaking, the idea of Sobel's proof is this. God, having all positive properties, must have the property of having any given truth be the case. Since God's existence is necessary, anything that is a truth must necessarily be a truth.

The modal collapse formula is represented as follows:

```
dec ModalCollapsLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(Q, w) -> @(mAlways, Q, w)];
```

It reads as: "everything that is the case is so necessarily". This entails the following statement as a corollary: "for every property `P`, any individual having property `P` has `P necessarily`". Modal collapse is not only a quite un-intuitive assertion, but also has many profound metaphysical repercussions. As an example, if it is the case that it is sunny in Berlin on August 16, 2018, then this is so necessarily, i.e., it is impossible that it could have been otherwise (and the same would apply to any other state of affairs). Other corollaries also seem intuitively quite controversial: if I happen to own a blue car, then my owning a blue car is a necessary fact, i.e., I could never have owned just a red car instead; somehow, owning a blue car is an essential trait of mine, it is part of my identity. Modal collapse implies that this happens for every single property we happen to exemplify (like owning a blue car, being brown eyed, being married to x, having done such and such, etc.). In particular, modal collapse implies that we have no agency whatsoever to choose our actions: anything we (seem to) have done so far has merely been imposed by providence upon us (simply because things could not have been otherwise). This is the reason why modal collapse has been associated with lack of free will.

Lemma 4.4 $\varphi \rightarrow \Box\varphi$.

To prove `ModalCollapsLem`, we will need the following lemma:

Lemma 4.5 $\forall Q\ (\exists^E \mathbf{G} \wedge Q) \rightarrow (G \Rightarrow Q)$.

```
dec MCAuxLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mExistAct, GodLike, w) ->
    @(Q, w) ->
    !(u : PWorld)
     [@(ActRel, w, u) -> @(mForallAct, \(z : Indiv, v : PWorld) [@(GodLike, z, v) -> @(Q, v)], u)]];
```

Lemma 4.6 $\varphi \rightarrow \Box\varphi$.

```
dec ModalCollapsLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(Q, w) -> @(mAlways, Q, w)];
```

Theorem 4.7 $\varphi \leftrightarrow \Box\varphi$.

```
dec ModalCollapsThm1 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(Q, w)]);
```

Theorem 4.8 $\varphi \leftrightarrow \Diamond\varphi$.

```
dec ModalCollapsThm2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(Q, w) -> @(mSometime, Q, w)],
     [@(mSometime, Q, w) -> @(Q, w)]);
```

Lemma 4.7 $\Diamond\varphi \rightarrow \varphi$.

```
dec MCAuxLem2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mSometime, Q, w) -> @(Q, w)];
```

Theorem 4.9 $\Diamond\varphi \leftrightarrow \Box\varphi$.

```
dec ModalCollapsThm3 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(mSometime, Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(mSometime, Q, w)]);
```

4.2.4 Inconsistency in Gödel's Ontological Argument

The inconsistency was found in the original version of Gödel's Ontological Argument [5]. It is described as follows:

Lemma 4.8 $\forall x \perp$ **ess** x.

```
dec EmptyEssLem :
  !(x : Indiv, w : PWorld)
   @(essInd, mNull, x, w);
```

Lemma 4.9 $\Box\exists\perp$.

```
dec InconsistLem :
  !(w : PWorld)
   @(mAlways, @(mExistAct, mNull), w);
```

Lemma 4.10 $\perp$.

```
dec InconsistThm :
  !(w : PWorld)
   Null;
```

5 Scott's Version of Gödel's Ontological Argument

5.1 Gödel's God Revised in PowerEpsilon

The inconsistency was eliminated in Scott's version of Gödel's Ontological Argument [14] by introducing an additional condition $\varphi(x)$ in φ **ess** x. The whole proof is then outlined in Table 2.

5.2 Proving that God's Existence is Possible

We start out with definition Definition 1 (Godlike) and axioms Axiom 1 - Axiom 3.

Definition 1 Being Godlike is equivalent to having all positive properties.

Axiom 1 Exactly one of a property or its negation is positive.

Axiom 2 Any property entailed by a positive property is positive.

Axiom 3 The combination of any collection of positive properties is itself positive.

```
dec Positive : [[Indiv -> PWorld -> Type(0)] -> PWorld -> Prop];

def pos =
  \(Z : [[Indiv -> PWorld -> Prop] -> PWorld -> Prop])
   !(X : [Indiv -> PWorld -> Prop])
   !(w : PWorld)
    [@(Z, X, w) -> @(Positive, X, w)];
```

Axiom 1 Either a property or its negation is positive, but not both:

$$\forall\varphi[\mathbf{P}(\neg\varphi) \leftrightarrow \neg\mathbf{P}(\varphi)]$$

Axiom 2 A property necessarily implied by a positive property is positive:

$$\forall\varphi\forall\psi[\mathbf{P}(\varphi) \wedge [\varphi \Rightarrow \psi] \rightarrow \mathbf{P}(\psi)]$$

Theorem 1 Positive properties are possibly exemplified:

$$\forall\varphi[\mathbf{P}(\varphi) \rightarrow \Diamond\exists\varphi(x)]$$

Definition 1 A God-like being possesses all positive properties:

$$\mathbf{G}(x) \equiv \forall\varphi[\mathbf{P}(\varphi) \rightarrow \varphi(x)]$$

Axiom 3 The property of being God-like is positive:

$$\mathbf{P}(\mathbf{G})$$

Theorem 2 Possibly, a God-like being exists:

$$\Diamond\exists x\ \mathbf{G}(x)$$

Axiom 4 Positive properties are necessarily positive:

$$\forall\varphi[\mathbf{P}(\varphi) \rightarrow \Box\mathbf{P}(\varphi)]$$

Definition 2 An essence of an individual is a property possessed by it and necessarily implying any of its properties:

$$\varphi\ \mathbf{ess}\ x \equiv \varphi(x) \wedge \forall\phi[\phi(x) \rightarrow \Box\forall y[\varphi(y) \rightarrow \phi(y)]]$$

Theorem 3 Being God-like is an essence of any God-like being:

$$\forall x[\mathbf{G}(x) \rightarrow \mathbf{G}\ \mathbf{ess}\ x]$$

Definition 3 Necessary existence of an individual is the necessary exemplification of all its essences:

$$\mathbf{NE}(x) \equiv \forall\varphi[\varphi\ \mathbf{ess}\ x \rightarrow \Box\exists y\ \varphi(y)]$$

Axiom5 Necessary existence is a positive property:

$$\mathbf{P}(\mathbf{NE})$$

Theorem 4 If a God-like being exists, then necessarily a God-like being exists:

$$\exists x\ \mathbf{G}(x) \rightarrow \Box\exists y\ \mathbf{G}(y)$$

Theorem 5 If possibly a God-like being exists, then necessarily a God-like being exists:

$$\Diamond\exists x\ \mathbf{G}(x) \rightarrow \Box\exists y\ \mathbf{G}(y)$$

Theorem 6 Necessarily, a God-like being exists:

$$\Box\exists x\ \mathbf{G}(x)$$

Table 2: Scott's version of Gödel's ontological argument

Definition 5.1 $\mathbf{G}(x) \equiv \forall\varphi[\mathbf{P}(\varphi) \rightarrow \varphi(x)]$.

```
def GodLike =
  \(x : Indiv, w : PWorld)
   !(Y : [Indiv -> PWorld -> Prop])
    [@(Positive, Y, w) -> @(Y, x, w)];
```

Axiom 5.1 $\forall\varphi[\mathbf{P}(\neg\varphi) \leftrightarrow \neg\mathbf{P}(\varphi)]$.

```
dec Axiom1 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Positive, @(mNeg, X), w) -> @(Neg, @(Positive, X, w))];

dec Axiom1b :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Neg, @(Positive, X, w)) -> @(Positive, @(mNeg, X), w)];
```

Axiom 5.2 $\forall\varphi,\phi[\mathbf{P}(\varphi) \wedge (\varphi \Rightarrow \phi) \rightarrow \mathbf{P}(\phi)]$.

```
dec Axiom2 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop], Y : [Indiv -> PWorld -> Prop])
   [@(Positive, X, w) ->
    @(mArrow, X, Y, w) ->
    @(Positive, Y, w)];
```

Axiom 5.3 $\forall Z,\varphi[pos(Z) \wedge (\varphi \cap Z) \rightarrow \mathbf{P}(\varphi)]$.

```
dec Axiom3 :
  !(Z : [[Indiv -> PWorld -> Prop] -> PWorld -> Prop], X : [Indiv -> PWorld -> Prop])
   [@(And, @(pos, Z), @(mJoint, X, Z)) ->
    !(w : PWorld) @(Positive, X, w)];
```

Lemma 5.1 $\forall\varphi[\mathbf{P}(\varphi) \rightarrow \neg\mathbf{P}(\neg\varphi)]$.

```
dec AuxLem1 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Positive, X, w) -> @(Neg, @(Positive, @(mNeg, X), w))];
```

Lemma 5.2 $\forall\varphi[\neg\mathbf{P}(\neg\varphi) \rightarrow \mathbf{P}(\varphi)]$.

```
dec AuxLem2 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Neg, @(Positive, @(mNeg, X), w)) -> @(Positive, X, w)];
```

From Axiom 1 and Axiom 2 follows theorem Theorem 1:

Theorem 1 Every positive property is possibly instantiated (if a property P is positive, then it is possible that some being has property P).

Theorem 5.1 $\forall\varphi[\mathbf{P}(\varphi) \rightarrow \Diamond\exists\varphi]$.

```
dec Theorem1 :
  !(X : [Indiv -> PWorld -> Prop])
  !(w : PWorld)
   [@(Positive, X, w) ->
    @(mSometime, @(mExistAct, X), w)];
```

From Definition 1 and Axiom 3 follows theorem Theorem 2:

Theorem 2 Being Godlike is a positive property.

Theorem 5.2 $\mathbf{P}(\mathbf{G})$.

```
dec Theorem2 :
  !(w : PWorld)
   @(Positive, GodLike, w);
```

To prove `Theorem2`, we need the following lemmas.

```
dec PosGodLikeAuxLem :
  !(w : PWorld)
   let Z = \(X : [Indiv -> PWorld -> Prop], v : PWorld)
            @(Neg, @(Positive, @(mNeg, X), v)),
       X = GodLike in
     @(Or,
       @(Positive, GodLike, w),
       @(And, @(And, @(pos, Z), @(mJoint, X, Z)), @(Neg, @(Positive, X, w))));

dec PosGodLikeLem :
  !(w : PWorld)
   ?(Z : [[Indiv -> PWorld -> Prop] -> PWorld -> Prop], X : [Indiv -> PWorld -> Prop])
    @(Or,
      @(Positive, GodLike, w),
      @(And, @(And, @(pos, Z), @(mJoint, X, Z)), @(Neg, @(Positive, X, w))));
```

From Theorem 1 and Theorem 2 follows theorem Theorem 3:

Theorem 3 Being Godlike is possibly instantiated.

Theorem 5.3 $\Diamond\exists x\ \mathbf{G}(x)$.

```
dec Theorem3 :
  !(w : PWorld)
   @(mSometime, @(mExistAct, GodLike), w);
```

5.3 Proving that God's Existence is Necessary, If Possible

Definition 2 A property E is the essence of an individual x iff x has E and all of x's properties are entailed by E.

Axiom 4 Positive (negative) properties are necessarily positive (negative).

```
dec Axiom4 :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   [@(Positive, X, w) -> @(mAlways, @(Positive, X), w)];
```

In order to avoid the inconsistency, the condition $\varphi(x)$ is introduced. We will see then that $\perp$**ess**x will never hold. This is the only difference in between the original version and Scott's version of Gödel's Ontological Argument.

Definition 5.2 φ **ess** $x \;\equiv\; \varphi(x) \wedge \forall\phi[\phi(x) \rightarrow (\varphi \Rightarrow \phi)]$.

```
def essInd =
  \(X : [Indiv -> PWorld -> Prop])
  \(x : Indiv)
  \(w : PWorld)
   @(And,
     @(X, x, w),
     !(Y : [Indiv -> PWorld -> Prop])
      [@(Y, x, w) -> @(mArrow, X, Y, w)]);
```

From Axiom 1 and Axiom 4 (using definitions Definition 1 and Definiotn 2) follows:

Theorem 4 Being Godlike is an essential property of any Godlike individual.

Definition 3 Necessary existence of an individual is the necessary instantiation of all its essences.

Axiom 5 Necessary existence is a positive property.

Theorem 5.4 $\forall x\ \mathbf{G}(x) \rightarrow \mathbf{G}(x)$ **ess** x.

```
dec Theorem4 :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) -> @(essInd, GodLike, x, w)];
```

Lemma 5.3 $\forall x[\mathbf{G}(x) \rightarrow \forall\phi[\phi(x) \rightarrow (G \Rightarrow \phi)]]$.

```
dec essIndLem :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) ->
    !(Y : [Indiv -> PWorld -> Prop])
     [@(Y, x, w) -> @(mArrow, GodLike, Y, w)]];
```

Definition 5.3 $\mathbf{NE}(x) \equiv \forall\phi[\phi$ **ess** $x \rightarrow (\Box\exists^E\phi)]$.

```
def NE =
  \(x : Indiv)
  \(w : PWorld)
   @(And,
     !(X : [Indiv -> PWorld -> Prop])
      [@(essInd, X, x, w) -> @(mAlways, @(mExistAct, X), w)],
     True);
```

Axiom 5.4 $\mathbf{P}(\mathbf{NE})$.

```
dec Axiom5 :
  !(w : PWorld)
   @(Positive, NE, w);
```

From Theorem 4 and Axiom 5 (using Definition 1, Definition 2, Definition 3) follows theorem Theorem 5:

Theorem 5 (The property of) being `Godlike`, if instantiated, is necessarily instantiated.

Theorem 5.5 $\Diamond\exists^E x\, \mathbf{G}(x) \to \Box\exists^E x\, \mathbf{G}(x)$.

```
dec Theorem5 :
  !(w : PWorld)
   [@(mSometime, @(mExistAct, GodLike), w) -> @(mAlways, @(mExistAct, GodLike), w)];
```

And finally from Theorem 3, Theorem 5 (together with some implicit modal axioms) the existence of (at least a) God follows:

Theorem 6 Being `Godlike` is actually instantiated.

Theorem 5.6 $\Box\exists^E x\, \mathbf{G}(x)$.

```
dec Theorem6 :
  !(w : PWorld)
   @(mAlways, @(mExistAct, GodLike), w);
```

Corollary 5.1 $\forall \mathrm{E}, \mathtt{P}\ \forall x\ \mathrm{E}\ \mathbf{ess}\ x \wedge \mathtt{P}(x) \to E \Rightarrow \mathtt{P}$.

```
dec Corollary1 :
  !(E : [Indiv -> PWorld -> Prop], P : [Indiv -> PWorld -> Prop])
  !(x : Indiv, w : PWorld)
   [@(essInd, E, x, w) ->
    @(P, x, w) ->
    @(mArrow, E, P, w)];
```

5.4 Modal Collapse

The modal collapse can not been avoided in Scott's version of ontological proof.

Lemma 5.4 $\varphi \to \Box\varphi$.

To prove `ModalCollapsLem`, we will need the following lemma:

Lemma 5.5 $\forall Q\ (\exists^E \mathbf{G} \wedge Q) \to (G \Rightarrow Q)$.

```
dec MCAuxLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mExistAct, GodLike, w) ->
    @(Q, w) ->
    !(u : PWorld)
     [@(ActRel, w, u) -> @(mForallAct, \(z : Indiv, v : PWorld) [@(GodLike, z, v) -> @(Q, v)], u)]];
```

Lemma 5.6 $\varphi \to \Box\varphi$.

```
dec ModalCollapsLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(Q, w) -> @(mAlways, Q, w)];
```

Theorem 5.7 $\varphi \leftrightarrow \Box\varphi$.

```
dec ModalCollapsThm1 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(Q, w)]);
```

Theorem 5.8 $\varphi \leftrightarrow \Diamond\varphi$.

```
dec ModalCollapsThm2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(Q, w) -> @(mSometime, Q, w)],
     [@(mSometime, Q, w) -> @(Q, w)]);
```

Lemma 5.7 $\Diamond\varphi \to \varphi$.

```
dec MCAuxLem2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mSometime, Q, w) -> @(Q, w)];
```

Theorem 5.9 $\Diamond\varphi \leftrightarrow \Box\varphi$.

```
dec ModalCollapsThm3 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(mSometime, Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(mSometime, Q, w)]);
```

6 Discussions

6.1 Related Works

This work was inspired by the work of Benzmüller, Weber and Woltzenlogel Paleo [3, 2, 4]. They have utilized the proof assistant Isabelle/HOL together with the external automated higher-order provers Leo-II and Satallax for computer-assisted analysis of the Anderson-Hájek's Ontological Controversy.

C.A. Anderson [1], Hájek [11] and Bjørdal [6] have proposed emendations of Gödel's axioms and definitions. They require neither comprehension restrictions nor more complex semantics and are therefore technically simpler to analyze with computer support.

Rushby has represented a work on Ontological Argument in PVS [13], I have had time to investigate the work.

6.2 Lessons Learned

There were two lessons learned from this practice:

1. You can prove something which you don't believe and you are not going to believe in future.
2. Another lesson learned from this work is an implication for computer science and information technology. When we introduce a new instruction or a new function into the computer systems in order to obtain more expressive power or to increase the performance, there will be a risk to introduce bugs and backdoors as we have done for the Ontological Argumentation. Beside of the inconsistency, we are not able to predict the risk until we found them finally, since we are not able to define them in advance.

References

[1] C. A. Anderson. Some emendations of Gödel's ontological proof. *Faith and Philosophy*, 7(3), 1990.

[2] C. Benzmüller and B. Woltzenlogel Paleo. Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers. In *Frontiers in Artificial Intelligence and Applications*, volume 263, pages 93–98, Prague, 2014. 21st European Conference on Artifial Intelligence, IOS Press.

[3] C. Benzmüller, L. Weber, and B. Woltzenlogel Paleo. Computer-Assisted Analysis of the Anderson–Hájek Ontological Controversy. *Logica Universalis*, 11(1):139–151, March 2017.

[4] Christoph Benzmüller and David Fuenmayor. Can Computers Help to Sharpen our Understanding of Ontological Arguments? In *Mathematics and Reality: Science and Spiritual Quest*, Bhubaneswar, India, 2018. 11th AISSQ.

[5] Christoph Benzmüller and Bruno Woltzenlogel Paleo. The Inconsistency in Gödels Ontological Argument: A Success Story for AI in Metaphysics. In S. Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, volume 1-3, pages 936–942, New York, 2016. IJCAI, AAAI Press.

[6] F. Bjørdal. Understanding Gödel's Ontological Argument. In T. Childers, editor, *The Logica Yearbook 1998*. Filosofia, Praha, 1999.

[7] R. L. Constable and et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1986.

[8] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3), 1988.

[9] K. Gödel. Appx. A: Notes in Kurt Gödel's Hand. In J. H. Sobel, editor, *Logic and Theism: Arguments for and Against Beliefs in God*, pages 144–145. Cambridge University Press, 2004.

[10] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF.* LNCS 78. Springer-Verlag, 1979.

[11] P. Hájek. Magari and others on Gödel's ontological proof. In A. Ursini and P. Agliano, editors, *Logic and Algebra*, pages 125–135. Marcel Dekker, New York, 1996.

[12] P. Martin-Löf. *Intuitionistic Type Theory.* Studies in Proof Theory, Vol. 1. Bibliopolis, Naples, 1984.

[13] J. Rushby. The Ontological Argument in PVS. In *Proceedings of CAV Workshop "Fun With Formal Methods"*, St. Petersburg, Russia, 2013.

[14] D. Scott. Appx. B: Notes in Dana Scott's Hand. In J.H. Sobel, editor, *Logic and Theism: Arguments for and Against Beliefs in God*, pages 145–146. Cambridge University Press, 2004.

[15] J.H. Sobel. Gödel's ontological proof. In *On Being and Saying. Essays for Richard Cartwright*, pages 241–261. MIT Press, 1987.

[16] M.-Y. Zhu. AUTOSTAR - a software development system. *ACM SIGPLAN Notices*, 23(3), 1989.

[17] M.-Y. Zhu and C.-W. Wang. A higher-order lambda calculus: PowerEpsilon. Technical report, Beijing Institute of Systems Engineering, Beijing, 1991.

[18] M.-Y. Zhu and C.-W. Wang. Program derivation in PowerEpsilon. In *Proceedings of COMPSAC'92*, Chicago, September 1992.

A Appendix: The Proofs in PowerEpsilon for Modal Logic

A.1 The Proof for ActRel

```
dec ARelFunEqLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) -> @(Equal, PWorld, w, v)];

def ARelFunEqLem =
  \(w : PWorld, v : PWorld)
  \(h : @(ActRel, w, v))
   let prf1 = @(ARelReflLem, w) in
   let prf2 = @(ARelFunctionalLem, w, w, v, prf1, h) in
   lec prf : @(Equal, PWorld, w, v) in
   let prf = prf2 in
     prf;

dec ARelSymmLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) -> @(ActRel, v, w)];

def ARelSymmLem =
  \(w : PWorld, v : PWorld)
  \(h : @(ActRel, w, v))
   let Q = \(z : PWorld)
            @(ActRel, z, w) in
   let prf = @(ARelFunEqLem, w, v, h, Q, @(ARelReflLem, w)) in
     prf;

dec ARelTranLem :
  !(w : PWorld, u : PWorld, v : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, u, v) ->
    @(ActRel, w, v)];

def ARelTranLem =
  \(w : PWorld, u : PWorld, v : PWorld)
  \(h1 : @(ActRel, w, u),
    h2 : @(ActRel, u, v))
   let prf = @(ARelSymmLem, w, u, h1) in
   let Q = \(z : PWorld)
            @(ActRel, z, v) in
     @(ARelFunEqLem, u, w, prf, Q, h2);

dec ARelSymmTranLem :
  !(w : PWorld, u : PWorld, v : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, w, v) ->
    @(ActRel, u, v)];

def ARelSymmTranLem =
  \(w : PWorld, u : PWorld, v : PWorld)
  \(h1 : @(ActRel, w, u),
    h2 : @(ActRel, w, v))
   let prf1 = @(ARelSymmLem, w, u, h1) in
   let prf2 = @(ARelTranLem, u, w, v, prf1, h2) in
   lec prf : @(ActRel, u, v) in
   let prf = prf2 in
     prf;

dec ARelSerialLem :
  !(w : PWorld)
   ?(u : PWorld)
    @(ActRel, w, u);

def ARelSerialLem =
  \(w : PWorld)
   let u = w in
   let prf1 = @(ARelReflLem, w) in
   let mkSerial = \(z : PWorld, p : @(ActRel, w, z))
                   <z, p> in
   let prf2 = @(mkSerial, u, prf1) in
   let prf = prf2 in
     prf;

dec ARelEuclideanLem :
  !(u : PWorld, t : PWorld, w : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, w, t) ->
    @(ActRel, u, t)];

def ARelEuclideanLem =
  \(u : PWorld, t : PWorld, w : PWorld)
  \(h1 : @(ActRel, w, u),
    h2 : @(ActRel, w, t))
   let prf1 = @(ARelSymmLem, w, u, h1) in
   let prf2 = @(ARelTranLem, u, w, t, prf1, h2) in
   lec prf : @(ActRel, u, t) in
   let prf = prf2 in
     prf;

dec ARelShiftReflLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) -> @(ActRel, v, v)];

def ARelShiftReflLem =
  \(w : PWorld, v : PWorld)
  \(h : @(ActRel, w, v))
   @(ARelReflLem, v);

dec ARelDenseLem :
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) ->
    ?(u : PWorld) @(And, @(ActRel, w, u), @(ActRel, u, v))];

def ARelDenseLem =
  \(w : PWorld, v : PWorld)
  \(h : @(ActRel, w, v))
   let u = v in
   let mkDense = \(ux : PWorld, p : @(And, @(ActRel, w, ux), @(ActRel, ux, v)))
                  <ux, p> in
   let prf = @(mkDense, u, @(ANDS, @(ActRel, w, u), @(ActRel, u, v), h, @(ARelReflLem, u))) in
     prf;
```

```
dec existsAtARelLem2 :
  !(x : Indiv)
  !(w : PWorld, v : PWorld)
   [@(ActRel, w, v) ->
    @(existsAt, x, w) ->
    @(existsAt, x, v)];

def existsAtARelLem2 =
  \(x : Indiv)
  \(w : PWorld, v : PWorld)
  \(h1 : @(ActRel, w, v),
    h2 : @(existsAt, x, w))
   let prf1 = @(ARelFunEqLem, w, v, h1) in
   let Q = \(Z : PWorld)
            @(existsAt, x, Z) in
   let prf2 = @(prf1, Q, h2) in
   lec prf : @(existsAt, x, v) in
   let prf = prf2 in
     prf;

dec existsAtARelLem :
  !(x : Indiv)
  !(w : PWorld, u : PWorld, v : PWorld)
   [@(ActRel, w, u) ->
    @(ActRel, w, v) ->
    @(existsAt, x, u) ->
    @(existsAt, x, v)];

def existsAtARelLem =
  \(x : Indiv)
  \(w : PWorld, u : PWorld, v : PWorld)
  \(h1 : @(ActRel, w, u),
    h2 : @(ActRel, w, v),
    h3 : @(existsAt, x, u))
   let Q = \(Z : PWorld)
            @(existsAt, x, Z) in
   let prf1 = @(ARelSymmLem, w, u, h1) in
   let prf2 = @(ARelTranLem, u, w, v, prf1, h2) in
   let prf3 = @(existsAtARelLem2, x, u, v, prf2, h3) in
   lec prf : @(existsAt, x, v) in
   let prf = prf3 in
     prf;
```

A.2 The Proof of Axiomatic Systems

A.2.1 The Proof of DistrRule

```
dec DistrRule :
  !(P : [PWorld -> Prop], Q : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, \(u : PWorld) [@(P, u) -> @(Q, u)], w) ->
    @(mAlways, P, w) ->
    @(mAlways, Q, w)];

def DistrRule =
  \(P : [PWorld -> Prop], Q : [PWorld -> Prop])
  \(w : PWorld)
  \(h1 : @(mAlways, \(u : PWorld) [@(P, u) -> @(Q, u)], w),
    h2 : @(mAlways, P, w))
  \(u : PWorld)
  \(h : @(ActRel, w, u))
   let prf1 = @(h1, u, h) in
   let prf2 = @(h2, u, h) in
   let prf3 = @(prf1, prf2) in
   lec prf : @(Q, u) in
   let prf = prf3 in
     prf;
```

A.2.2 The Proof of `ReflRule` (Reflexive Rule)

```
dec ReflRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, P, w) -> @(P, w)];

def ReflRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(mAlways, P, w))
   lec prf : @(P, w) in
   let prf = @(h, w, @(ARelReflLem, w)) in
     prf;
```

A.2.3 The Proof of `FourRule` (Transitive Rule)

```
dec FourRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, P, w) -> @(mAlways, @(mAlways, P), w)];

def FourRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(mAlways, P, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
  \(v : PWorld)
  \(hyp : @(ActRel, u, v))
   lec prf : @(P, v) in
   let prf = @(h, v, @(ARelTranLem, w, u, v, hy, hyp)) in
     prf;
```

A.2.4 The Proof of `BRule` (Symmetric Rule)

```
dec BRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
```

```
   [@(P, w) -> @(mAlways, @(mSometime, P), w)];

def BRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(P, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
   let v = u in
   let prf1 = @(ARelReflLem, v) in
   let Q = \(Z : PWorld)
            @(P, Z) in
   let prf2 = @(ARelFunEqLem, w, u, hy, Q, h) in
   let mkDiamond = \(vx : PWorld, px : @(And, @(ActRel, u, vx), @(P, vx)))
                    <vx, px> in
   let prf3 = @(ANDS, @(ActRel, u, v), @(P, v), prf1, prf2) in
   let prf4 = @(mkDiamond, v, prf3) in
   lec prf : @(mSometime, P, u) in
   let prf = prf4 in
     prf;
```

A.2.5 The Proof of `DRule` (Serial Rule)

```
dec DRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, P, w) -> @(mSometime, P, w)];

def DRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(mAlways, P, w))
   let u = w in
   let prf1 = @(h, w, @(ARelReflLem, w)) in
   let mkDiamond = \(ux : PWorld, px : @(And, @(ActRel, w, ux), @(P, ux)))
                    <ux, px> in
   let prf2 = @(mkDiamond, u, @(ANDS, @(ActRel, w, u), @(P, u), @(ARelReflLem, w), prf1)) in
   lec prf : @(mSometime, P, w) in
   let prf = prf2 in
     prf;
```

A.2.6 The Proof of `FiveRule` (Euclidean Rule)

```
dec FiveRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mSometime, P, w) -> @(mAlways, @(mSometime, P), w)];

def FiveRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(mSometime, P, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
   let Q = \(Z : PWorld)
            @(And, @(   mSometime, P, Z), True) in
   let prf1 = @(ARelFunEqLem, w, u, hy, Q, @(ANDS, @(mSometime, P, w), True, h, TRUE)) in
   let prf2 = @(PJ1, @(mSometime, P, u), True, prf1) in
   lec prf : @(mSometime, P, u) in
   let prf = prf2 in
     prf;
```

A.2.7 The Proof of `CDRule` (Functional Rule)

```
dec CDRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mSometime, P, w) -> @(mAlways, P, w)];

def CDRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(mSometime, P, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
   let v = @(FST, h) in
   let prf1 = @(SND, h) in
   let P1 = @(ActRel, w, v),
       P2 = @(P, v) in
   let p1 = @(PJ1, P1, P2, prf1),
       p2 = @(PJ2, P1, P2, prf1) in
   let prf2 = @(ARelSymmLem, w, v, p1) in
   let prf3 = @(ARelTranLem, v, w, u, prf2, hy) in
   let Q = \(Z : PWorld)
            @(P, Z) in
   let prf4 = @(ARelFunEqLem, v, u, prf3, Q, p2) in
   lec prf : @(P, u) in
   let prf = prf4 in
     prf;
```

A.2.8 The Proof of `BMRule` (Shift Reflexive Rule)

```
dec BMRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   @(mAlways, \(u : PWorld) [@(mAlways, P, u) -> @(P, u)], w);

def BMRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(v : PWorld)
  \(h : @(ActRel, w, v))
  \(hy : @(mAlways, P, v))
   lec prf : @(P, v) in
   let prf = @(hy, v, @(ARelReflLem, v)) in
```

```
      prf;
```

A.2.9 The Proof of `C4Rule` (Dense Rule)

```
dec C4Rule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mAlways, @(mAlways, P), w) -> @(mAlways, P, w)];

def C4Rule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(mAlways, @(mAlways, P), w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
   lec prf : @(P, u) in
   let prf = @(h, w, @(ARelReflLem, w), u, hy) in
     prf;
```

A.2.10 The Proof of `CRule` (Convergent Rule)

```
dec CRule :
  !(P : [PWorld -> Prop])
  !(w : PWorld)
   [@(mSometime, @(mAlways, P), w) -> @(mAlways, @(mSometime, P), w)];

def CRule =
  \(P : [PWorld -> Prop])
  \(w : PWorld)
  \(h : @(mSometime, @(mAlways, P), w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
   let v = @(FST, h) in
   let prf1 = @(SND, h) in
   let P1 = @(ActRel, w, v),
       P2 = @(mAlways, P, v) in
   let p1 = @(PJ1, P1, P2, prf1),
       p2 = @(PJ2, P1, P2, prf1) in
   let prf2 = @(ARelSymmLem, w, u, hy) in
   let prf3 = @(ARelTranLem, u, w, v, prf2, p1) in
   lec prf4 : @(P, v) in
   let prf4 = @(p2, v, @(ARelReflLem, v)) in
   let mkDiamond = \(ux : PWorld, px : @(And, @(ActRel, u, ux), @(P, ux)))
                   <ux, px> in
   let prf5 = @(mkDiamond, v, @(ANDS, @(ActRel, u, v), @(P, v), prf3, prf4)) in
   lec prf : @(mSometime, P, u) in
   let prf = prf5 in
     prf;
```

A.3 The Proof of mNegNullTrueLem and mNegTrueNullLem for ⊤ and ⊥

```
dec mNegNullTrueLem : @(Equal, [Indiv -> PWorld -> Prop], @(mNeg, mNull), mTrue);

def mNegNullTrueLem = @(Refl_Eq, [Indiv -> PWorld -> Prop], @(mNeg, mNull));

dec mNegTrueNullLem : @(Equal, [Indiv -> PWorld -> Prop], @(mNeg, mTrue), mNull);

def mNegTrueNullLem =
  let prf1 = @(Refl_Eq, [Indiv -> PWorld -> Prop], @(mNeg, mTrue)) in
  let prf2 = @(mNegNegLem, mNull) in
  let Q = \(Z : [Indiv -> PWorld -> Prop])
           @(Equal, [Indiv -> PWorld -> Prop], @(mNeg, mTrue), Z) in
  let prf3 = @(prf2, Q, prf1) in
    prf3;
```

A.4 The Proof of mAlwaysElimLem for □-Elimination

```
dec mAlwaysElimLem :
  !(w : PWorld)
  !(Q : [PWorld -> Type(0)])
   [@(mAlways, Q, w) -> @(Q, w)];

def mAlwaysElimLem =
  \(w : PWorld)
  \(Q : [PWorld -> Type(0)])
  \(h : @(mAlways, Q, w))
   let prf0 = @(ARelReflLem, w) in
   let prf1 = @(h, w, prf0) in
   lec prf : @(Q, w) in
   let prf = prf1 in
     prf;
```

A.5 The Proof of mSometimeIntrLem for ◇-Introduction

```
dec mSometimeIntrLem :
  !(w : PWorld)
  !(Q : [PWorld -> Type(0)])
   [@(Q, w) -> @(mSometime, Q, w)];

def mSometimeIntrLem =
  \(w : PWorld)
  \(Q : [PWorld -> Type(0)])
  \(h : @(Q, w))
   let prf1 = @(ARelReflLem, w) in
   let prf2 = @(ANDS, @(ActRel, w, w), @(Q, w), prf1, h) in
   let mkSometime = \(v : PWorld, p : @(And, @(ActRel, w, v), @(Q, v)))
                     <v, p> in
   let prf3 = @(mkSometime, w, prf2) in
   lec prf : @(mSometime, Q, w) in
   let prf = prf3 in
     prf;
```

A.6 The Proof of SomeTimeNegLem for $\neg\Diamond\exists^E$

```
dec SomeTimeNegLem :
  !(X : [Indiv -> PWorld -> Prop])
  !(w : PWorld)
   [@(Neg, @(mSometime, @(mExistAct, X), w)) ->
    @(mAlways, @(mForallAct, @(mNeg, X)), w)];

def SomeTimeNegLem =
  \(X : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
  \(h : @(Neg, @(mSometime, @(mExistAct, X), w)))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
  \(x : Indiv)
  \(hyp : @(existsAt, x, u))
  \(H : @(X, x, u))
   let prf1 = @(ANDS, @(existsAt, x, u), @(X, x, u), hyp, H) in
   let prf2 = <x, prf1> in
   let prf3 = @(ANDS, @(ActRel, w, u), @(mExistAct, X, u), hy, prf2) in
   let prf4 = <u, prf3> in
   let prf5 = @(h, prf4) in
   lec prf : Null in
   let prf = prf in
     prf;
```

A.7 The Proof of mNullTrueArwLem for $\Rightarrow$ and $\perp$

```
dec mNullTrueArwLem :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   @(mArrow, mNull, X, w);

def mNullTrueArwLem =
  \(w : PWorld)
  \(X : [Indiv -> PWorld -> Prop])
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
  \(x : Indiv)
  \(hyp1 : @(existsAt, x, u))
  \(hyp2 : @(mNull, x, u))
   lec prf : @(X, x, u) in
   let prf = @(hyp2, @(X, x, u)) in
     prf;
```

A.8 The Proof of IdArrowLem for $\Rightarrow$

```
dec IdArrowLem :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop])
   @(mArrow, X, X, w);

def IdArrowLem =
  \(w : PWorld)
  \(X : [Indiv -> PWorld -> Prop])
  \(u : PWorld)
  \(h : @(ActRel, w, u))
  \(x : Indiv)
  \(hy : @(existsAt, x, u))
  \(hyp : @(X, x, u))
   hyp;
```

A.9 The Proof of `mArrowTranLem` for Transitivity of ⇒

```
dec mArrowTranLem :
  !(w : PWorld)
  !(X : [Indiv -> PWorld -> Prop], Y : [Indiv -> PWorld -> Prop], Z : [Indiv -> PWorld -> Prop])
   [@(mArrow, X, Y, w) ->
    @(mArrow, Y, Z, w) ->
    @(mArrow, X, Z, w)];

def mArrowTranLem =
  \(w : PWorld)
  \(X : [Indiv -> PWorld -> Prop], Y : [Indiv -> PWorld -> Prop], Z : [Indiv -> PWorld -> Prop])
  \(h1 : @(mArrow, X, Y, w), h2 : @(mArrow, Y, Z, w))
  \(u : PWorld)
  \(h : @(ActRel, w, u))
  \(x : Indiv)
  \(hy1 : @(existsAt, x, u),
    hy2 : @(X, x, u))
   lec prf1 : @(Y, x, u) in
   let prf1 = @(h1, u, h, x, hy1, hy2) in
   let prf2 = @(h2, u, h, x, hy1, prf1) in
   lec prf : @(Z, x, u) in
   let prf = prf2 in
     prf;
```

A.10 The Proof of `mAlwaysWorldLem` for World Changing of □

```
dec mAlwaysWorldLem :
  !(P : [PWorld -> Type(0)])
  !(w : PWorld, u : PWorld)
   [@(ActRel, w, u) -> @(mAlways, P, u) -> @(mAlways, P, w)];

def mAlwaysWorldLem =
  \(P : [PWorld -> Type(0)])
  \(w : PWorld, u : PWorld)
  \(h1 : @(ActRel, w, u),
    h2 : @(mAlways, P, u))
  \(v : PWorld)
  \(hy : @(ActRel, w, v))
   let prf1 = @(ARelTranLem, u, w, v, @(ARelSymmLem, w, u, h1), hy) in
   let prf2 = @(h2, v, prf1) in
   lec prf : @(P, v) in
   let prf = prf2 in
```

```
      prf;
```

A.11 The Proof of `mTrueLem` for $\Box, \forall^E, \top$

```
dec mTrueLem :
  !(w : PWorld)
   @(mAlways, @(mForallAct, mTrue), w);

def mTrueLem =
  \(w : PWorld)
  \(u : PWorld)
  \(h : @(ActRel, w, u))
  \(x : Indiv)
  \(hy : @(existsAt, x, u))
  \(hyp : @(mNull, x, u))
   lec prf : @(mNull, x, u) in
   let prf = hyp in
     prf;
```

B Appendix: The Proofs in PowerEpsilon for Original Version of Gödel's Ontological Argument

B.1 The Proof of `Theorem1`

```
dec Theorem1 :
  !(X : [Indiv -> PWorld -> Prop])
  !(w : PWorld)
   [@(Positive, X, w) ->
    @(mSometime, @(mExistAct, X), w)];
```

To prove `Theorem1`, we will need the following lemma:

```
dec mNullNegPosLem :
  !(w : PWorld)
   @(Neg, @(Positive, mNull, w));

def mNullNegPosLem =
  \(w : PWorld)
  \(h : @(Positive, mNull, w))
   lec prf1 : @(mArrow, mNull, mTrue, w) in
   let prf1 = \(v : PWorld)
              \(hy : @(ActRel, w, v))
              \(x : Indiv)
              \(hyp : @(existsAt, x, v))
              \(H : @(mNull, x, v))
               @(H, @(mTrue, x, v)) in
   lec prf2 : @(Positive, mTrue, w) in
   let prf2 = @(Axiom2, w, mNull, mTrue, h, prf1) in
   let prf3 = @(Axiom1, w, mNull, prf2) in
   let prf4 = @(prf3, h) in
   lec prf : Null in
   let prf = prf4 in
     prf;
```

We then have the proof of `Theorem1` as follows:

```
def Theorem1 =
  \(X : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
  \(h : @(Positive, X, w))
   let Q = @(mSometime, @(mExistAct, X), w) in
     @(WHEN0,
       Q,
       @(Neg, Q),
       Q,
       @(ExclMiddleRule0, Q),
       \(q : Q)
        q,
       \(q : @(Neg, Q))
        lec prf1 : @(Positive, mNull, w) in
        let prf1 = @(Axiom2, w, X, mNull) in
        lec prf2 : @(mAlways, @(mForallAct, @(mNeg, X)), w) in
        let prf2 = @(SomeTimeNegLem, X, w, q) in
        lec prf3 : @(Positive, mNull, w) in
        let prf3 = @(prf1, h, prf2) in
        let prf4 = @(mNullNegPosLem, w, prf3) in
        let prf5 = @(prf4, @(And, Q, True)) in
        let prf6 = @(PJ1, Q, True, prf5) in
        lec prf : Q in
        let prf = prf6 in
          prf)
```

B.2 The Proof of `Theorem2`

```
dec Theorem2 :
  !(w : PWorld)
   @(Positive, GodLike, w);

def Theorem2 =
  \(w : PWorld)
   let prf1 = @(PosGodLikeLem, w) in
   let Z = @(FST, prf1),
       X = @(FST, @(SND, prf1)) in
   let prf2 = @(SND, @(SND, prf1)) in
   let Q1 = @(Positive, GodLike, w),
       Q2 = @(And, @(And, @(pos, Z), @(mJoint, X, Z)), @(Neg, @(Positive, X, w))) in
     @(WHEN,
       Q1,
       Q2,
       Q1,
       prf2,
       \(q1 : Q1)
        q1,
       \(q2 : Q2)
        let P1 = @(And, @(pos, Z), @(mJoint, X, Z)),
            P2 = @(Neg, @(Positive, X, w)) in
        let p1 = @(PJ1, P1, P2, q2),
            p2 = @(PJ2, P1, P2, q2) in
        let prf3 = @(Axiom3, Z, X, p1, w) in
        let prf4 = @(p2, prf3, @(Positive, GodLike, w)) in
        lec prf : @(Positive, GodLike, w) in
        let prf = prf4 in
          prf)
```

B.3 The Proof of Theorem3

```
dec Theorem3 :
  !(w : PWorld)
   @(mSometime, @(mExistAct, GodLike), w);

def Theorem3 =
  \(w : PWorld)
   let prf1 = @(Theorem2, w) in
   let prf2 = @(Theorem1, GodLike, w, prf1) in
     prf2;
```

B.4 The Proof of Theorem4

```
dec Theorem4 :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) -> @(essInd, GodLike, x, w)];
```

To prove Theorem4, we will need the lemma essIndLem:

```
dec essIndLem :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) ->
    !(Y : [Indiv -> PWorld -> Prop])
     [@(Y, x, w) -> @(mArrow, GodLike, Y, w)]];
```

To prove essIndLem, we will need another auxiliary lemma ActRelAtLem:

```
dec ActRelAtLem :
  !(w : PWorld, v : PWorld)
  !(Y : [Indiv -> PWorld -> Prop])
   [@(Positive, Y, v) ->
    @(ActRel, w, v) ->
    @(Positive, Y, w)];

def ActRelAtLem =
  \(w : PWorld, v : PWorld)
  \(Y : [Indiv -> PWorld -> Prop])
  \(h1 : @(Positive, Y, v),
    h2 : @(ActRel, w, v))
   lec prf1 : @(mAlways, @(Positive, Y), v) in
   let prf1 = @(Axiom4, v, Y, h1) in
   let prf2 = @(ARelSymmLem, w, v, h2) in
   let prf3 = @(prf1, w, prf2) in
   lec prf : @(Positive, Y, w) in
```

```
    let prf = prf3 in
      prf;
```

The proof of `essIndLem` is obtained by applying the exclusive middle rule, `Axiom1b` and `ActRelAtLem`.

```
def essIndLem =
  \(x : Indiv, w : PWorld)
  \(h : @(GodLike, x, w))
  \(Y : [Indiv -> PWorld -> Prop])
  \(hy : @(Y, x, w))
  \(v : PWorld)
  \(hyp : @(ActRel, w, v))
  \(y : Indiv)
  \(H1 : @(existsAt, y, v),
    H2 : @(GodLike, y, v))
   @(WHEN,
     @(Positive, Y, v),
     @(Neg, @(Positive, Y, v)),
     @(Y, y, v),
     @(ExclMiddleRule0, @(Positive, Y, v)),
     \(p : @(Positive, Y, v))
      lec prf : @(Y, y, v) in
      let prf = @(H2, Y, p) in
        prf,
     \(p : @(Neg, @(Positive, Y, v)))
      let nY = @(mNeg, Y) in
      lec prfX1 : @(Positive, nY, v) in
      let prfX1 = @(Axiom1b, v, Y, p) in
      lec prfX2 : @(Positive, nY, w) in
      let prfX2 = @(ActRelAtLem, w, v, nY, prfX1, hyp) in
      lec prfX3 : @(nY, x, w) in
      let prfX3 = @(h, nY, prfX2) in
      let prfX4 = @(prfX3, hy, @(Y, y, v)) in
      lec prf : @(Y, y, v) in
      let prf = prfX4 in
        prf);
```

The proof of `Theorem4` is obtained as the following.

```
def Theorem4 =
  \(x : Indiv, w : PWorld)
  \(h : @(GodLike, x, w))
   lec prf : @(essInd, GodLike, x, w) in
   let prf = @(essIndLem, x, w, h) in
     prf;
```

B.5 The Proof of `Theorem5`

```
dec Theorem5 :
  !(w : PWorld)
   [@(mSometime, @(mExistAct, GodLike), w) -> @(mAlways, @(mExistAct, GodLike), w)];

def Theorem5 =
  \(w : PWorld)
  \(h : @(mSometime, @(mExistAct, GodLike), w))
```

```
    let u = @(FST, h) in
    lec prf1 : @(And, @(ActRel, w, u), @(mExistAct, GodLike, u)) in
    let prf1 = @(SND, h) in
    let Q1 = @(ActRel, w, u),
        Q2 = @(mExistAct, GodLike, u) in
    let q1 = @(PJ1, Q1, Q2, prf1),
        q2 = @(PJ2, Q1, Q2, prf1) in
    let x = @(FST, q2) in
    let prf2 = @(SND, q2) in
    let R1 = @(existsAt, x, u),
        R2 = @(GodLike, x, u) in
    let r1 = @(PJ1, R1, R2, prf2),
        r2 = @(PJ2, R1, R2, prf2) in
    lec prf3 : @(Positive, NE, u) in
    let prf3 = @(Axiom5, u) in
    lec prf4 : @(essInd, GodLike, x, u) in
    let prf4 = @(Theorem4, x, u, r2) in
    lec prf5 : @(NE, x, u) in
    let prf5 = @(r2, NE, prf3) in
    let U1 = !(X : [Indiv -> PWorld -> Prop])
               [@(essInd, X, x, u) -> @(mAlways, @(mExistAct, X), u)],
        U2 = True in
    let u1 = @(PJ1, U1, U2, prf5),
        u2 = @(PJ2, U1, U2, prf5) in
    lec prf6 : @(mAlways, @(mExistAct, GodLike), u) in
    let prf6 = @(u1, GodLike, prf4) in
    let prf7 = \(v : PWorld)
               \(hy : @(ActRel, w, v))
                let prfX0 = @(ARelSymmTranLem, w, u, v, q1, hy) in
                lec prfX1 : @(existsAt, x, v) in
                let prfX1 = @(existsAtARelLem, x, w, u, v, q1, hy, r1) in
                lec prfX2 : @(mExistAct, GodLike, v) in
                let prfX2 = @(prf6, v, prfX0) in
                lec prf : @(mExistAct, GodLike, v) in
                let prf = prfX2 in
                  prf in
    lec prf : @(mAlways, @(mExistAct, GodLike), w) in
    let prf = prf7 in
      prf;
```

B.6 The Proof of Theorem6

```
dec Theorem6 :
  !(w : PWorld)
   @(mAlways, @(mExistAct, GodLike), w);

def Theorem6 =
  \(w : PWorld)
   let prf1 = @(Theorem3, w) in
   let prf2 = @(Theorem5, w, prf1) in
     prf2;
```

B.7 The Proof of `Corollary1`

```
dec Corollary1 :
  !(E : [Indiv -> PWorld -> Prop], P : [Indiv -> PWorld -> Prop])
  !(x : Indiv, w : PWorld)
   [@(essInd, E, x, w) ->
    @(P, x, w) ->
    @(mArrow, E, P, w)];

def Corollary1 =
  \(E : [Indiv -> PWorld -> Prop], P : [Indiv -> PWorld -> Prop])
  \(x : Indiv, w : PWorld)
  \(h1 : @(essInd, E, x, w),
    h2 : @(P, x, w))
   let Q1 = @(E, x, w),
       Q2 = !(Y : [Indiv -> PWorld -> Prop])
             [@(Y, x, w) -> @(mArrow, E, Y, w)] in
   let q1 = @(PJ1, Q1, Q2, h1),
       q2 = @(PJ2, Q1, Q2, h1) in
   let prf = @(q2, P, h2) in
     prf;
```

B.8 The Proof of Modal Collapse

B.8.1 The proof of `ModalCollapsLem`

```
dec ModalCollapsLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(Q, w) -> @(mAlways, Q, w)];
```

To prove `ModalCollapsLem`, we will need to prove lemma `MCAuxLem`:

```
dec MCAuxLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mExistAct, GodLike, w) ->
    @(Q, w) ->
    @(mAlways,
      \(u : PWorld)
       @(mForallAct, \(z : Indiv, v : PWorld) [@(GodLike, z, v) -> @(Q, v)], u), w)];

def MCAuxLem =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
  \(h1 : @(mExistAct, GodLike, w),
    h2 : @(Q, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
  \(x : Indiv)
  \(hyp1 : @(existsAt, x, u),
    hyp2 : @(GodLike, x, u))
   let z = @(FST, h1) in
   let prf1 = @(SND, h1) in
   let Q1 = @(existsAt, z, w),
       Q2 = @(GodLike, z, w) in
   let q1 = @(PJ1, Q1, Q2, prf1),
       q2 = @(PJ2, Q1, Q2, prf1) in
   let Z = \(z : Indiv, v : PWorld)
            @(Q, v) in
     @(WHEN,
       @(Positive, Z, u),
       @(Neg, @(Positive, Z, u)),
       @(Q, u),
       @(ExclMiddleRule0, @(Positive, Z, u)),
       \(q : @(Positive, Z, u))
        @(hyp2, Z, q),
       \(q : @(Neg, @(Positive, Z, u)))
        lec prfX1 : @(Positive, @(mNeg, Z), u) in
        let prfX1 = @(Axiom1b, u, Z, q) in
        lec prfX2 : @(mNeg, Z, x, u) in
        let prfX2 = @(hyp2, @(mNeg, Z), prfX1) in
        lec prfX3 : @(mAlways, @(Positive, @(mNeg, Z)), u) in
        lec prfX3 : !(vx : PWorld)
                     [@(ActRel, u, vx) -> @(Positive, @(mNeg, Z), vx)] in
        let prfX3 = @(Axiom4, u, @(mNeg, Z), prfX1) in
        let prfX4 = @(ARelSymmLem, w, u, hy) in
        lec prfX5 : @(Positive, @(mNeg, Z), w) in
        let prfX5 = @(prfX3, w, prfX4) in
        let prfX6 = @(q2, @(mNeg, Z), prfX5, h2) in
        let prfX7 = @(prfX6, @(Q, u)) in
        lec prf : @(Q, u) in
        let prf = prfX7 in
          prf);
```

We then have the proof of ModalCollapsLem as the following.

```
def ModalCollapsLem =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
  \(h : @(Q, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
   let prfC1 = @(Theorem6, w) in
   let prfC2 = @(mAlwaysElimLem, w, @(mExistAct, GodLike), prfC1) in
   let prfA1 = @(Theorem3, w) in
   let v = @(FST, prfA1) in
   let prfA2 = @(SND, prfA1) in
   let P1 = @(ActRel, w, v),
       P2 = @(mExistAct, GodLike, v) in
   let p1 = @(PJ1, P1, P2, prfA2),
       p2 = @(PJ2, P1, P2, prfA2) in
   let x = @(FST, p2) in
   let prfA3 = @(SND, p2) in
   let U1 = @(existsAt, x, v),
       U2 = @(GodLike, x, v) in
   let u1 = @(PJ1, U1, U2, prfA3),
       u2 = @(PJ2, U1, U2, prfA3) in
   lec prfB0 : @(ActRel, v, u) in
   let prfB0 = @(ARelSymmTranLem, w, v, u, p1, hy) in
   let prf1 = @(Theorem6, v) in
   let prfB1 = @(prf1, u, prfB0) in
```

```
    let z = @(FST, prfB1) in
    let prfB2 = @(SND, prfB1) in
    let V1 = @(existsAt, z, u),
        V2 = @(GodLike, z, u) in
    let v1 = @(PJ1, V1, V2, prfB2),
        v2 = @(PJ2, V1, V2, prfB2) in
    lec prf3 : @(Q, u) in
    let prf3 = @(MCAuxLem, w, Q, prfC2, h, u, hy, z, v1, v2) in
    lec prf : @(Q, u) in
    let prf = prf3 in
      prf;
```

B.8.2 The Proof of `ModalCollapsThm1`

```
dec ModalCollapsThm1 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(Q, w)]);

def ModalCollapsThm1 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
   let P1 = [@(Q, w) -> @(mAlways, Q, w)],
       P2 = [@(mAlways, Q, w) -> @(Q, w)] in
   let p1 = @(ModalCollapsLem, w, Q),
       p2 = @(mAlwaysElimLem, w, Q) in
     @(ANDS, P1, P2, p1, p2);
```

B.8.3 The Proof of `ModalCollapsThm2`

```
dec ModalCollapsThm2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(Q, w) -> @(mSometime, Q, w)],
     [@(mSometime, Q, w) -> @(Q, w)]);

dec MCAuxLem2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mSometime, Q, w) -> @(Q, w)];

def MCAuxLem2 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
  \(h : @(mSometime, Q, w))
   let u = @(FST, h) in
   let prf1 = @(SND, h) in
   let P1 = @(ActRel, w, u),
       P2 = @(Q, u) in
   let p1 = @(PJ1, P1, P2, prf1),
        p2 = @(PJ2, P1, P2, prf1) in
    let prf2 = @(ModalCollapsLem, u, Q, p2) in
    let prf3 = @(prf2, w, @(ARelSymmLem, w, u, p1)) in
    lec prf : @(Q, w) in
    let prf = prf3 in
      prf;

def ModalCollapsThm2 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
   let P1 = [@(Q, w) -> @(mSometime, Q, w)],
       P2 = [@(mSometime, Q, w) -> @(Q, w)] in
   let p1 = @(mSometimeIntrLem, w, Q),
       p2 = @(MCAuxLem2, w, Q) in
     @(ANDS, P1, P2, p1, p2);
```

B.8.4 The Proof of `ModalCollapsThm3`

```
dec ModalCollapsThm3 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(mSometime, Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(mSometime, Q, w)]);

def ModalCollapsThm3 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
   let Q1 = [@(mSometime, Q, w) -> @(mAlways, Q, w)],
       Q2 = [@(mAlways, Q, w) -> @(mSometime, Q, w)] in
   let q1 = \(h : @(mSometime, Q, w))
              let prf1 = @(MCAuxLem2, w, Q, h) in
              let prf2 = @(ModalCollapsLem, w, Q, prf1) in
              lec prf : @(mAlways, Q, w) in
              let prf = prf2 in
                prf in
   let q2 = \(h : @(mAlways, Q, w))
              let prf1 = @(mAlwaysElimLem, w, Q, h) in
              let prf2 = @(mSometimeIntrLem, w, Q, prf1) in
              lec prf : @(mSometime, Q, w) in
              let prf = prf2 in
                prf in
     @(ANDS, Q1, Q2, q1, q2);
```

B.9 The Proof of Inconsistency Theorem `InconsistThm`

```
dec EmptyEssLem :
  !(x : Indiv, w : PWorld)
   @(essInd, mNull, x, w);

def EmptyEssLem =
  \(x : Indiv, w : PWorld)
  \(Y : [Indiv -> PWorld -> Prop])
  \(h : @(Y, x, w))
   @(mNullTrueArwLem, w, Y);

dec InconsistLem :
  !(w : PWorld)
   @(mAlways, @(mExistAct, mNull), w);

def InconsistLem =
  \(w : PWorld)
   lec prf2 : @(Positive, NE, w) in
   let prf2 = @(Axiom5, w) in
   lec prf3 : @(mSometime, @(mExistAct, NE), w) in
   let prf3 = @(Theorem1, NE, w, prf2) in
   let u = @(FST, prf3) in
   let prf4 = @(SND, prf3) in
   let Q1 = @(ActRel, w, u),
       Q2 = @(mExistAct, NE, u) in
   let q1 = @(PJ1, Q1, Q2, prf4),
       q2 = @(PJ2, Q1, Q2, prf4) in
   let z = @(FST, q2) in
   let prf5 = @(SND, q2) in
   let W1 = @(existsAt, z, u),
       W2 = @(NE, z, u) in
   let w1 = @(PJ1, W1, W2, prf5),
       w2 = @(PJ2, W1, W2, prf5) in
   let W1 = !(X : [Indiv -> PWorld -> Prop])
              [@(essInd, X, z, u) -> @(mAlways, @(mExistAct, X), u)] in
   let prf6 = @(PJ1, W1, True, w2) in
   lec prf7 : @(essInd, mNull, z, u) in
   let prf7 = @(EmptyEssLem, z, u) in
   lec prf8 : @(mAlways, @(mExistAct, mNull), u) in
   let prf8 = @(prf6, mNull, prf7) in
   let prf9 = @(mAlwaysWorldLem, @(mExistAct, mNull), w, u, q1, prf8) in
     prf9;

dec InconsistThm :
  !(w : PWorld)
   Null;

def InconsistThm =
  \(w : PWorld)
   let prf1 = @(InconsistLem, w) in
   let prf2 = @(prf1, w, @(ARelReflLem, w)) in
   let x = @(FST, prf2) in
   let prf3 = @(SND, prf2) in
   let W1 = @(existsAt, x, w),
       W2 = @(mNull, x, w) in
   let w1 = @(PJ1, W1, W2, prf3),
       w2 = @(PJ2, W1, W2, prf3) in
     w2;
```

C Appendix: The Proofs in PowerEpsilon for Scott's Version of Gödel's Ontological Argument

C.1 The Proof of `Theorem1`

```
dec Theorem1 :
  !(X : [Indiv -> PWorld -> Prop])
  !(w : PWorld)
   [@(Positive, X, w) ->
    @(mSometime, @(mExistAct, X), w)];
```

To prove `Theorem1`, we will need the following lemma:

```
dec mNullNegPosLem :
  !(w : PWorld)
   @(Neg, @(Positive, mNull, w));

def mNullNegPosLem =
  \(w : PWorld)
  \(h : @(Positive, mNull, w))
   lec prf1 : @(mArrow, mNull, mTrue, w) in
   let prf1 = \(v : PWorld)
              \(hy : @(ActRel, w, v))
              \(x : Indiv)
              \(hyp : @(existsAt, x, v))
              \(H : @(mNull, x, v))
               @(H, @(mTrue, x, v)) in
   lec prf2 : @(Positive, mTrue, w) in
   let prf2 = @(Axiom2, w, mNull, mTrue, h, prf1) in
   let prf3 = @(Axiom1, w, mNull, prf2) in
   let prf4 = @(prf3, h) in
   lec prf : Null in
   let prf = prf4 in
     prf;
```

We then have the proof of `Theorem1` as follows:

```
def Theorem1 =
  \(X : [Indiv -> PWorld -> Prop])
  \(w : PWorld)
  \(h : @(Positive, X, w))
   let Q = @(mSometime, @(mExistAct, X), w) in
     @(WHEN0,
       Q,
       @(Neg, Q),
       Q,
       @(ExclMiddleRule0, Q),
       \(q : Q)
        q,
       \(q : @(Neg, Q))
        lec prf1 : @(Positive, mNull, w) in
        let prf1 = @(Axiom2, w, X, mNull) in
        lec prf2 : @(mAlways, @(mForallAct, @(mNeg, X)), w) in
        let prf2 = @(SomeTimeNegLem, X, w, q) in
        lec prf3 : @(Positive, mNull, w) in
        let prf3 = @(prf1, h, prf2) in
        let prf4 = @(mNullNegPosLem, w, prf3) in
        let prf5 = @(prf4, @(And, Q, True)) in
        let prf6 = @(PJ1, Q, True, prf5) in
        lec prf : Q in
        let prf = prf6 in
          prf)
```

C.2 The Proof of Theorem2

```
dec Theorem2 :
  !(w : PWorld)
   @(Positive, GodLike, w);

def Theorem2 =
  \(w : PWorld)
   let prf1 = @(PosGodLikeLem, w) in
   let Z = @(FST, prf1),
       X = @(FST, @(SND, prf1)) in
   let prf2 = @(SND, @(SND, prf1)) in
   let Q1 = @(Positive, GodLike, w),
       Q2 = @(And, @(And, @(pos, Z), @(mJoint, X, Z)), @(Neg, @(Positive, X, w))) in
     @(WHEN,
       Q1,
       Q2,
       Q1,
       prf2,
       \(q1 : Q1)
        q1,
       \(q2 : Q2)
        let P1 = @(And, @(pos, Z), @(mJoint, X, Z)),
            P2 = @(Neg, @(Positive, X, w)) in
        let p1 = @(PJ1, P1, P2, q2),
            p2 = @(PJ2, P1, P2, q2) in
        let prf3 = @(Axiom3, Z, X, p1, w) in
        let prf4 = @(p2, prf3, @(Positive, GodLike, w)) in
        lec prf : @(Positive, GodLike, w) in
        let prf = prf4 in
          prf)
```

C.3 The Proof of Theorem3

```
dec Theorem3 :
  !(w : PWorld)
   @(mSometime, @(mExistAct, GodLike), w);

def Theorem3 =
  \(w : PWorld)
   let prf1 = @(Theorem2, w) in
   let prf2 = @(Theorem1, GodLike, w, prf1) in
     prf2;
```

C.4 The Proof of `Theorem4`

```
dec Theorem4 :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) -> @(essInd, GodLike, x, w)];
```

To prove `Theorem4`, we will need the lemma `essIndLem`:

```
dec essIndLem :
  !(x : Indiv, w : PWorld)
   [@(GodLike, x, w) ->
    !(Y : [Indiv -> PWorld -> Prop])
     [@(Y, x, w) -> @(mArrow, GodLike, Y, w)]];
```

To prove `essIndLem`, we will need another auxiliary lemma `ActRelAtLem`:

```
dec ActRelAtLem :
  !(w : PWorld, v : PWorld)
  !(Y : [Indiv -> PWorld -> Prop])
   [@(Positive, Y, v) ->
    @(ActRel, w, v) ->
    @(Positive, Y, w)];

def ActRelAtLem =
  \(w : PWorld, v : PWorld)
  \(Y : [Indiv -> PWorld -> Prop])
  \(h1 : @(Positive, Y, v),
    h2 : @(ActRel, w, v))
   lec prf1 : @(mAlways, @(Positive, Y), v) in
   let prf1 = @(Axiom4, v, Y, h1) in
   let prf2 = @(ARelSymmLem, w, v, h2) in
   let prf3 = @(prf1, w, prf2) in
   lec prf  : @(Positive, Y, w) in
   let prf = prf3 in
     prf;
```

The proof of `essIndLem` is obtained by applying the exclusive middle rule, `Axiom1b` and `ActRelAtLem`.

```
def essIndLem =
  \(x : Indiv, w : PWorld)
  \(h : @(GodLike, x, w))
  \(Y : [Indiv -> PWorld -> Prop])
  \(hy : @(Y, x, w))
  \(v : PWorld)
  \(hyp : @(ActRel, w, v))
  \(y : Indiv)
  \(H1 : @(existsAt, y, v),
    H2 : @(GodLike, y, v))
   @(WHEN,
     @(Positive, Y, v),
     @(Neg, @(Positive, Y, v)),
     @(Y, y, v),
     @(ExclMiddleRule0, @(Positive, Y, v)),
     \(p : @(Positive, Y, v))
      lec prf : @(Y, y, v) in
      let prf = @(H2, Y, p) in
```

```
        prf,
      \(p : @(Neg, @(Positive, Y, v)))
       let nY = @(mNeg, Y) in
       lec prfX1 : @(Positive, nY, v) in
       let prfX1 = @(Axiom1b, v, Y, p) in
       lec prfX2 : @(Positive, nY, w) in
       let prfX2 = @(ActRelAtLem, w, v, nY, prfX1, hyp) in
       lec prfX3 : @(nY, x, w) in
       let prfX3 = @(h, nY, prfX2) in
       let prfX4 = @(prfX3, hy, @(Y, y, v)) in
       lec prf : @(Y, y, v) in
       let prf = prfX4 in
         prf);
```

The proof of Theorem4 is obtained as the following.

```
def Theorem4 =
  \(x : Indiv, w : PWorld)
  \(h : @(GodLike, x, w))
   let Q1 = @(GodLike, x, w),
       Q2 = !(Y : [Indiv -> PWorld -> Prop])
             [@(Y, x, w) -> @(mArrow, GodLike, Y, w)] in
   let q1 = h in
   let q2 = @(essIndLem, x, w, h) in
   lec prf : @(essInd, GodLike, x, w) in
   let prf = @(ANDS, Q1, Q2, q1, q2) in
     prf;
```

C.5 The Proof of Theorem5

```
dec Theorem5 :
  !(w : PWorld)
   [@(mSometime, @(mExistAct, GodLike), w) -> @(mAlways, @(mExistAct, GodLike), w)];

def Theorem5 =
  \(w : PWorld)
  \(h : @(mSometime, @(mExistAct, GodLike), w))
   let u = @(FST, h) in
   lec prf1 : @(And, @(ActRel, w, u), @(mExistAct, GodLike, u)) in
   let prf1 = @(SND, h) in
   let Q1 = @(ActRel, w, u),
       Q2 = @(mExistAct, GodLike, u) in
   let q1 = @(PJ1, Q1, Q2, prf1),
       q2 = @(PJ2, Q1, Q2, prf1) in
   let x = @(FST, q2) in
   let prf2 = @(SND, q2) in
   let R1 = @(existsAt, x, u),
       R2 = @(GodLike, x, u) in
   let r1 = @(PJ1, R1, R2, prf2),
       r2 = @(PJ2, R1, R2, prf2) in
   lec prf3 : @(Positive, NE, u) in
   let prf3 = @(Axiom5, u) in
   lec prf4 : @(essInd, GodLike, x, u) in
   let prf4 = @(Theorem4, x, u, r2) in
   lec prf5 : @(NE, x, u) in
```

```
    let prf5 = @(r2, NE, prf3) in
    let U1 = !(X : [Indiv -> PWorld -> Prop])
               [@(essInd, X, x, u) -> @(mAlways, @(mExistAct, X), u)],
        U2 = True in
    let u1 = @(PJ1, U1, U2, prf5),
        u2 = @(PJ2, U1, U2, prf5) in
    lec prf6 : @(mAlways, @(mExistAct, GodLike), u) in
    let prf6 = @(u1, GodLike, prf4) in
    let prf7 = \(v : PWorld)
               \(hy : @(ActRel, w, v))
                let prfX0 = @(ARelSymmTranLem, w, u, v, q1, hy) in
                lec prfX1 : @(existsAt, x, v) in
                let prfX1 = @(existsAtARelLem, x, w, u, v, q1, hy, r1) in
                lec prfX2 : @(mExistAct, GodLike, v) in
                let prfX2 = @(prf6, v, prfX0) in
                lec prf : @(mExistAct, GodLike, v) in
                let prf = prfX2 in
                  prf in
    lec prf : @(mAlways, @(mExistAct, GodLike), w) in
    let prf = prf7 in
      prf;
```

C.6 The Proof of `Theorem6`

```
dec Theorem6 :
  !(w : PWorld)
   @(mAlways, @(mExistAct, GodLike), w);

def Theorem6 =
  \(w : PWorld)
   let prf1 = @(Theorem3, w) in
   let prf2 = @(Theorem5, w, prf1) in
     prf2;
```

C.7 The Proof of `Corollary1`

```
dec Corollary1 :
  !(E : [Indiv -> PWorld -> Prop], P : [Indiv -> PWorld -> Prop])
  !(x : Indiv, w : PWorld)
   [@(essInd, E, x, w) ->
    @(P, x, w) ->
    @(mArrow, E, P, w)];

def Corollary1 =
  \(E : [Indiv -> PWorld -> Prop], P : [Indiv -> PWorld -> Prop])
  \(x : Indiv, w : PWorld)
  \(h1 : @(essInd, E, x, w),
    h2 : @(P, x, w))
   let Q1 = @(E, x, w),
       Q2 = !(Y : [Indiv -> PWorld -> Prop])
              [@(Y, x, w) -> @(mArrow, E, Y, w)] in
   let q1 = @(PJ1, Q1, Q2, h1),
       q2 = @(PJ2, Q1, Q2, h1) in
   let prf = @(q2, P, h2) in
     prf;
```

C.8 The Proof of Modal Collapse

C.8.1 The proof of ModalCollapsLem

```
dec ModalCollapsLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(Q, w) -> @(mAlways, Q, w)];
```

To prove `ModalCollapsLem`, we will need to prove lemma `MCAuxLem`:

```
dec MCAuxLem :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mExistAct, GodLike, w) ->
    @(Q, w) ->
    @(mAlways,
      \(u : PWorld)
       @(mForallAct, \(z : Indiv, v : PWorld) [@(GodLike, z, v) -> @(Q, v)], u), w)];

def MCAuxLem =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
  \(h1 : @(mExistAct, GodLike, w),
    h2 : @(Q, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
  \(x : Indiv)
  \(hyp1 : @(existsAt, x, u),
    hyp2 : @(GodLike, x, u))
   let z = @(FST, h1) in
   let prf1 = @(SND, h1) in
   let Q1 = @(existsAt, z, w),
       Q2 = @(GodLike, z, w) in
   let q1 = @(PJ1, Q1, Q2, prf1),
       q2 = @(PJ2, Q1, Q2, prf1) in
   let Z = \(z : Indiv, v : PWorld)
            @(Q, v) in
     @(WHEN,
       @(Positive, Z, u),
       @(Neg, @(Positive, Z, u)),
       @(Q, u),
       @(ExclMiddleRule0, @(Positive, Z, u)),
       \(q : @(Positive, Z, u))
        @(hyp2, Z, q),
       \(q : @(Neg, @(Positive, Z, u)))
        lec prfX1 : @(Positive, @(mNeg, Z), u) in
        let prfX1 = @(Axiom1b, u, Z, q) in
        lec prfX2 : @(mNeg, Z, x, u) in
        let prfX2 = @(hyp2, @(mNeg, Z), prfX1) in
        lec prfX3 : @(mAlways, @(Positive, @(mNeg, Z)), u) in
        lec prfX3 : !(vx : PWorld)
```

```
                    [@(ActRel, u, vx) -> @(Positive, @(mNeg, Z), vx)] in
        let prfX3 = @(Axiom4, u, @(mNeg, Z), prfX1) in
        let prfX4 = @(ARelSymmLem, w, u, hy) in
        lec prfX5 : @(Positive, @(mNeg, Z), w) in
        let prfX5 = @(prfX3, w, prfX4) in
        let prfX6 = @(q2, @(mNeg, Z), prfX5, h2) in
        let prfX7 = @(prfX6, @(Q, u)) in
        lec prf : @(Q, u) in
        let prf = prfX7 in
          prf);
```

We then have the proof of `ModalCollapsLem` as the following.

```
def ModalCollapsLem =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
  \(h : @(Q, w))
  \(u : PWorld)
  \(hy : @(ActRel, w, u))
   let prfC1 = @(Theorem6, w) in
   let prfC2 = @(mAlwaysElimLem, w, @(mExistAct, GodLike), prfC1) in
   let prfA1 = @(Theorem3, w) in
   let v = @(FST, prfA1) in
   let prfA2 = @(SND, prfA1) in
   let P1 = @(ActRel, w, v),
       P2 = @(mExistAct, GodLike, v) in
   let p1 = @(PJ1, P1, P2, prfA2),
       p2 = @(PJ2, P1, P2, prfA2) in
   let x = @(FST, p2) in
   let prfA3 = @(SND, p2) in
   let U1 = @(existsAt, x, v),
       U2 = @(GodLike, x, v) in
   let u1 = @(PJ1, U1, U2, prfA3),
       u2 = @(PJ2, U1, U2, prfA3) in
   lec prfB0 : @(ActRel, v, u) in
   let prfB0 = @(ARelSymmTranLem, w, v, u, p1, hy) in
   let prf1 = @(Theorem6, v) in
   let prfB1 = @(prf1, u, prfB0) in
   let z = @(FST, prfB1) in
   let prfB2 = @(SND, prfB1) in
   let V1 = @(existsAt, z, u),
       V2 = @(GodLike, z, u) in
   let v1 = @(PJ1, V1, V2, prfB2),
       v2 = @(PJ2, V1, V2, prfB2) in
   lec prf3 : @(Q, u) in
   let prf3 = @(MCAuxLem, w, Q, prfC2, h, u, hy, z, v1, v2) in
   lec prf : @(Q, u) in
   let prf = prf3 in
     prf;
```

C.8.2 The Proof of `ModalCollapsThm1`

```
dec ModalCollapsThm1 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
```

```
   @(And,
     [@(Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(Q, w)]);

def ModalCollapsThm1 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
   let P1 = [@(Q, w) -> @(mAlways, Q, w)],
       P2 = [@(mAlways, Q, w) -> @(Q, w)] in
   let p1 = @(ModalCollapsLem, w, Q),
       p2 = @(mAlwaysElimLem, w, Q) in
     @(ANDS, P1, P2, p1, p2);
```

C.8.3 The Proof of `ModalCollapsThm2`

```
dec ModalCollapsThm2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(Q, w) -> @(mSometime, Q, w)],
     [@(mSometime, Q, w) -> @(Q, w)]);

dec MCAuxLem2 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   [@(mSometime, Q, w) -> @(Q, w)];

def MCAuxLem2 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
  \(h : @(mSometime, Q, w))
   let u = @(FST, h) in
   let prf1 = @(SND, h) in
   let P1 = @(ActRel, w, u),
       P2 = @(Q, u) in
   let p1 = @(PJ1, P1, P2, prf1),
       p2 = @(PJ2, P1, P2, prf1) in
   let prf2 = @(ModalCollapsLem, u, Q, p2) in
   let prf3 = @(prf2, w, @(ARelSymmLem, w, u, p1)) in
   lec prf : @(Q, w) in
   let prf = prf3 in
     prf;

def ModalCollapsThm2 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
   let P1 = [@(Q, w) -> @(mSometime, Q, w)],
       P2 = [@(mSometime, Q, w) -> @(Q, w)] in
   let p1 = @(mSometimeIntrLem, w, Q),
       p2 = @(MCAuxLem2, w, Q) in
     @(ANDS, P1, P2, p1, p2);
```

C.8.4 The Proof of ModalCollapsThm3

```
dec ModalCollapsThm3 :
  !(w : PWorld)
  !(Q : [PWorld -> Prop])
   @(And,
     [@(mSometime, Q, w) -> @(mAlways, Q, w)],
     [@(mAlways, Q, w) -> @(mSometime, Q, w)]);

def ModalCollapsThm3 =
  \(w : PWorld)
  \(Q : [PWorld -> Prop])
   let Q1 = [@(mSometime, Q, w) -> @(mAlways, Q, w)],
       Q2 = [@(mAlways, Q, w) -> @(mSometime, Q, w)] in
   let q1 = \(h : @(mSometime, Q, w))
             let prf1 = @(MCAuxLem2, w, Q, h) in
             let prf2 = @(ModalCollapsLem, w, Q, prf1) in
             lec prf : @(mAlways, Q, w) in
             let prf = prf2 in
               prf in
   let q2 = \(h : @(mAlways, Q, w))
             let prf1 = @(mAlwaysElimLem, w, Q, h) in
             let prf2 = @(mSometimeIntrLem, w, Q, prf1) in
             lec prf : @(mSometime, Q, w) in
             let prf = prf2 in
               prf in
     @(ANDS, Q1, Q2, q1, q2);
```