
**NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS**

**SCHOOL OF SCIENCES**
**DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS**


**BSc THESIS**

# Formalized Proofs of the Extension of Consistent Approximation Fixpoint Theory


**Georgios Panagiotopoulos**

**Supervisors:** **Panagiotis Rontogiannis,** Professor
**Angelos Charalambidis,** Assistant Professor



**ATHENS**

**JULY 2024**

**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Τυπικές Αποδείξεις της Επέκτασης της Θεωρίας Προσέγγισης Σταθερών Σημείων

**Γεώργιος Παναγιωτόπουλος**

**Επιβλέποντες:** **Παναγιώτης Ροντογιάννης,** Καθηγητής
**Άγγελος Χαραλαμπίδης,** Επίκουρος Καθηγητής

**ΑΘΗΝΑ**

**ΙΟΥΛΙΟΣ 2024**

**BSc THESIS**

# Formalized Proofs of the Extension of Consistent Approximation Fixpoint Theory

**Georgios Panagiotopoulos**
**S.N.:** 1115201700113

**SUPERVISORS:** **Panagiotis Rontogiannis,** Professor
**Angelos Charalambidis,** Assistant Professor

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

Τυπικές Αποδείξεις της Επέκτασης της Θεωρίας Προσέγγισης Σταθερών Σημείων

**Γεώργιος Παναγιωτόπουλος**
**Α.Μ.:** 1115201700113

**ΕΠΙΒΛΕΠΟΝΤΕΣ:** **Παναγιώτης Ροντογιάννης,** Καθηγητής
**Άγγελος Χαραλαμπίδης,** Επίκουρος Καθηγητής

# ABSTRACT

The purpose of this thesis is to formalize key proofs within the framework of consistent approximation fixpoint theory using the Lean 4 theorem prover, with the aim to refine the proofs themselves as well as verify their results. This is accomplished by constraining the formalization process to already verified tools, specifically the Lean base library and Mathlib.

**SUBJECT AREA:** Programming Language Semantics

**KEYWORDS:** lattice, fixpoint, logic, proof, lean

# ΠΕΡΙΛΗΨΗ

Ο σκοπός αυτής της εργασίας είναι η τυποποίηση βασικών αποδείξεων της θεωρίας προσέγγισης σταθερών σημείων χρησιμοποιώντας το εργαλείο αποδείξεων Lean 4, με ως στόχο την βελτίωση των ίδιων των αποδείξεων καθώς και στην επαλήθευση των αποτελεσμάτων τους. Αυτό επιτυγχάνεται με την εξ ολόκληρου χρήση ήδη επαληθευμένων εργαλείων για την διαδικασία τυποποιήσης, συγκεκριμένα την βασική βιβλιοθήκη της Lean και την Mathlib.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ:** Σημασιολόγια Γλώσσων Προγραμματισμού

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:** πλέγμα, σταθερο σήμειο, λογικός, αποδείξη, lean

# CONTENTS

# PREFACE

In the context of the Well-Founded semantics of higher-order logic programs, Angelos Charalambidis, Panos Rondogiannis, and Ioanna Symeonidou [1] developed an extension of the theory of consistent approximating operators from [3]. This extension ended up being of some general mathematical interest, necessitating a stricter formalization of the theory presented.

In general, approximation fixpoint theory has a wide range of applications in the context of logic programming and its approximate fields, such as databases and artificial intelligence. It gives us a framework for understanding the semantics of non-monotonic logic, a type logic which closely mirrors human reasoning; deducing from incomplete data and making assumptions based on the now available information, whilst also being able to retract those assumptions when new, contradictory, information is presented.

At the same time, formalizing proofs has gained popularity for several reasons. Obviously, the most important one is the verification of the results themselves, which allows us to have a higher level of confidence in the correctness of our theory. Secondly, many of those theorems are being utilized in systems where correctness is paramount. One such example are compilers, which rely on such theorems to guarantee the reliability of their operations. And of course, formalizing proofs also provides an overview of the dependencies among the theorems, which along with providing a better understanding of the theory, it also allows us to know if a proof remains valid once changes or extensions are made.

In the text, we shall first go over the primary mathematical structures that are required for this theory, then we shall go over the Propositions 8-13, their formalization in Lean, and outline their non-formal proofs. These proofs, originally presented in various papers [1] [2] [3], will follow the more rigorous Lean proof structure, providing a more formal approach. As such, the proofs presented can also function as a supplement for understanding the Lean code.

While some code examples will be provided, they will not be exhaustive due to practical constraints. The snippets will serve as rough guidelines. For the complete project, refer to this public repository.

# 1. INTRODUCTION

Non-monotonicity is a concept that appears in various systems. While this extension of the Approximation Fixpoint Theory spawned from reasoning about the semantics of logic programs, we can certainly map it into other fields. It is thus vital to build an intuitive understanding of the motivation behind the theory, as well as the math itself.

We can start with our understanding of knowledge. A very basic and intuitive way to think about knowledge is through sets. A set of propositions could be thought of as representing our knowledge about a certain system. In two valued logic, if a proposition of the general domain is missing from this set, it means that it is false, and vise versa.

Let's say we know two people, John and Bill. We know nothing about Bill but we know that John uses Windows 11. Trying to represent this in traditional logic already proves challenging, since we are basically forced to assume that Bill doesn't use Windows, which is not necessarily true.

Let's also assume that we know that if someone doesn't use Windows, they are a good person.

$$good(Person) \leftarrow \neg Person(windows)$$

At this point it is important to note, that this is an intensional example for clarity. In reality, this theory discusses extensional structures, structures which we should understand as general formulas of deduction. Those formulas can just as easily be used for numbers or any other relation. We generally don't need to attribute any specific intensional property to the above formulation for it to be useful.

Continuing with this specific example, if we naively try to extract truth from those rules, we will end up with more truth about Bill than about John, namely we will end up saying that we know Bill (represented by the empty set) is a good person while John (represented by the set with just one element) is not.

$$\{\} \subseteq \{windows\} \Rightarrow good(\{\}) = True \geq_{truth} False = good(\{windows\})$$

This approach, of course, doesn't correspond to our intuition about knowledge and truth and yet it is this very approach that is used for defining the semantics of positive logic programs. It is obvious that we need a different concept for monotonicity, as well as a different representation of knowledge if we are to sufficiently reason about more complex systems.

The first problem is with the initial modeling; in simple databases and programs it might be reasonable to assume that Bill doesn't use Windows, we are basically operating under the presumption that our knowledge is complete, so if he did, we'd know about it. That's not a reasonable assumption neither in our everyday lives, nor in more complex systems.

So, firstly, we can do away with this by adding a third truth value which we can call "I don't know" or simply $Undefined$. With this tool our example already appears to align better with our intuition; we don't know if Bill uses Windows or not, so by applying our rule:

$$\{\} \subseteq \{windows\} \Rightarrow good(\{\}) = Undefined \leq False = good(\{windows\})$$

We can see that we used a different ordering now, an ordering on which it doesn't matter if something is simply $True$ or $False$; what actually matters is whether or not we know about it. We will call this information ordering. Formally this ordering can be expressed as:

$$Undefined \leq False \wedge Undefined \leq True$$

A problem that we face now, is that we still need the $\leq_{truth}$ ordering. That's because we need our model to be "truth minimal". Intuitively, we know that we wouldn't want to believe every assertion we can think of so long as it doesn't contradict any evidence. For example, it generally wouldn't be reasonable to assume that Bill has an invisible pet dragon in his house, even though this assertion could probably never contradict any evidence. So, it is obvious that we still need a way to keep our models from building towards this excess of assumptions because they are too "lazy" to account for it.

The way we solve for this is by establishing a bijection between the functions that can account for the information ordering and our original naive functions. We can represent the third truth value through the difference of a pair of naive functions $(f_1, f_2)$. We can of course reason about these functions as sets:

$f : D \to \{False, True\}$, then $F$ is a set of element $D$ where:

- $\forall\ d \in F,\ f(d) = True$
- $\forall\ d \notin F,\ f(d) = False$

Or, more concisely:

$$F = \{(d : D) \mid f(d) = True\}$$

If we then have two sets of truths, those two sets are allowed to contradict each other. What we actually know can be represented as their intersection, and what we don't know as their difference. Formally, for sets $F_1$ and $F_2$ representing the $f_1$ and $f_2$ functions respectively, the set of undefined values of the function $f = (f_1, f_2)$ can be expressed as $F_2 - F_1$ and the set of the actual knowledge as $F_2 \cap F_1$.

Note that we take the difference $F_2 - F_1$ rather than $F_1 - F_2$. That's because this approximation of knowledge relies in the reconciliation between $F_2$ assuming too much and $F_1$ assuming too little.

Marc Denecker, Victor W. Marek, and Mirosław Truszczynskic[3] showed that by iterating a proper operator, starting from the worst possible pair $(f_1, f_2)$, we can reach a fixpoint. The contribution of Angelos Charalambidis, Panos Rondogiannis, and Ioanna Symeonidou[1] was to then generalize this approximation structure into higher order relations, where the functions are not simple predicates, but higher order predicates, predicates that can handle as input functions of any complexity. The formalization of the precise inner workings of this generalization is the subject of this thesis.

We will start with some basic definitions.

# 2. DOMAIN THEORY PRELIMINARIES

**Definition 2.1. Partial Order**
A Partial Order is a $\leq$ relation on a set (or type) $D$, that is *reflexive*, *transitive* and *antisymmetric*.

**Definition 2.2. Bounded Order**
A Bounded Order is a $\leq$ relation on a set (or type) $D$, such that Top ($\top$) and Bottom ($\bot$) elements exist.

**Definition 2.3. Complete Semilattice**
A Complete Semilattice is a Partial Order in which all subsets (or sets of elements of the type) have either a supremum (least upper bound) or an infimum (greatest lower bound).

**Definition 2.4. Complete Lattice**
A Complete Lattice is a Partial Order in which all subsets, have a supremum and an infimum.

**Definition 2.5. Chain**
Given a Partial Order ($D$, $\leq$), every linearly ordered subset S of D will be called a chain. A Partial Order is chain-complete if it has a Bottom element and every chain S $\subseteq D$ has a supremum.

**Theorem 2.6.** (Theorem 4 in [1])
A Partial Order ($D$, $\leq$) is a Complete Lattice if it is a Complete Semilattice of supremum. Alternatively a Partial Order is a Complete Lattice if it is a Complete Semilattice of infimum.

*Note:* The above version of the theorem is closer to the one used by Mathlib. In [1] it is asserted that a Partial Order is a Complete Lattice if every non-empty subset has a supremum in the Partial Order and also the Partial Order has a Bottom element. The second assertion is already accounted for in Mathlib, where the supremum of an empty set is the Bottom element.

Mathematically this holds true since every element of the empty set is vacuously an upper bound, and supremum is the least upper bound.

We will be using this theorem for Proposition 8.

**Theorem 2.7. Knaster-Tarski**
Let $L$ be a $(D, \leq)$ complete lattice and let $f : D \to D$ be a monotonic function. Then the set of fixed points of $f$ in $L$ forms a complete lattice under its order, with the least fixpoint being the infimum of all the prefixpoints, i.e. the set of all $x$ such that $f\ x \leq x$.

# 3. UNDERLYING STRUCTURES

Now that we have briefly gone through the basic Domain Theoretic structures, as well as some theorems that we will need, we are ready to talk about the specific structures of the proposed extension.

**Definition 3.1. Bounded Partial Order**
Let $L$ be a Partial Order $(D, \leq)$ and assume that $L$ is Bounded. That is to say that there exist two elements in $D$, Top and Bottom such that

$$\forall\, (x : D),\ \bot \leq\ x\ \leq\ \top$$

This can be expressed in Lean, simply as:

```
class BoundedPartialOrder (D : Type u)
    [PartialOrder D] extends BoundedOrder D
```

That means, that our Baseline Order is a class, which takes a Partial Order instance as a parameter, and which inherits the characteristics of a Bounded Order (i.e. Top and Bottom). $D$ is the underlying type of the elements of this Order.

*Note*: The PartialOrder and the BoundedOrder classes are part of Mathlib.

**Definition 3.2. Twin Complete Lattices**
Let $L_1$ and $L_2$ be Complete Lattices $(S_{D1}, \leq)$ and $(S_{D2}, \leq)$. That is, they are both defined on the previous order. $L_1$ and $L_2$ have the following properties :

1. $S_{D1} \subseteq D \wedge S_{D2} \subseteq D$
2. $S_{D1} \cup S_{D2} = D$
3. $\top, \bot \in S_{D1} \wedge \top, \bot \in S_{D2}$

Since those Twin Complete Lattices use the same order as the Bounded Partial Order, in order to implement it, we need a way to tell Lean to use this very same base Partial Order of the $D$ type.

The Mathlib Complete Lattice class doesn't have that capability, so we create our own (which we can then easily map to the Mathlib one).

```
class CompleteLatticeFromOrder
    (D : Type u)
    [PartialOrder D]
    extends SupSet D, InfSet D,
    Sup D, Inf D, BoundedOrder D
```

**The subset relation** can be expressed in Lean with the help of subtypes. Given a predicates $D_1 = D \to Prop$ and $D_2 = D \to Prop$, subtypes are made by restricting the initial type $D$ to the true values of those predicates, i.e.

$$S_{D1} = \{x\ //\ D_1\ x\}$$

$$S_{D2} = \{x \,//\, D_2\ x\}$$

*Note:* The notation for the underlying subtypes $S_{D1}$ and $S_{D2}$ doesn't appear in the Lean code, instead they are called directly either as

```
Subtype D1
```

or through the very formula introduced above

```
{x // D1 x}
```

We can thus create our class instances

```
variable
    (D : Type u)
    (D1 D2 : D → Prop)
    [O : PartialOrder D]
    [L : BoundedPartialOrder D]
    [L1 : CompleteLatticeFromOrder {x // D1 x}]
    [L2 : CompleteLatticeFromOrder {x // D2 x}]
```

**The second relation** can be expressed with the help of some logic, since we are now working with predicates, as:

```
def subtypesCreateType : Prop :=
    ∀ (d : D), D1 d ∨ D2 d
```

**The last expression** merely asserts that the two predicates should yield true to the Bottom and Top values:

```
def subtypesContainTopBot : Prop :=
    D1 L.top ∧ D1 L.bot ∧ D2 L.top ∧ D2 L.bot
```

### Definition 3.3. Interlattice Properties

1. **Interlattice Least Upper Bound Property**: Let $b \in S_{D2}$ and $S \subseteq S_{D1}$ such that for every $x \in S$, $x \leq b$. Then $\bigvee_{L_1} S \leq b$.
2. **Interlattice Greatest Lower Bound Property**: Let $a \in S_{D1}$ and $S \subseteq S_{D2}$ such that for every $x \in S$, $x \geq a$. Then, $\bigwedge_{L_2} S \geq a$.

Expressing those two properties in Lean is pretty straight forward.
For **Interlattice LUB** we have:

```
def interlattice_lub : Prop :=
    ∀ (b : D), D2 b → ∀ (S : Set (Subtype D1)),
    (∀ x, x ∈ S → ↑x ≤ b) →  ↑(L1.sSup S) ≤ b
```

And for **Interlattice GLB** we have:

```
def interlattice_glb : Prop :=
    ∀ (a : D), D1 a → ∀ (S : Set (Subtype D2)),
    (∀ x, x ∈ S → ↑x ≥ a) → ↑(L2.sInf S) ≥ a
```

*Note:* The uparrows (↑) denote the coercion from a type to its parent type, which is the only way we can compare elements of type $S_{D_1}$ with elements of type $S_{D_2}$, namely through the order of type $D$.

**Definition 3.4. Bounded Subtypes**
Given $a \in S_{D1}$ and $b \in S_{D2}$, we write $[a,b]_{L_1} = \{x \in S_{D1} \mid a \leq x \leq b\}$. Symmetrically, $[a,b]_{L_2} = \{x \in S_{D2} \mid a \leq x \leq b\}$

This we can express in Lean as a subtype of either one of our subtypes $S_{D_1}$, $S_{D_2}$. It should of course be a predicate, a function $S_{D_1} \to Prop$. And this predicate should check whether or not our $x$ is within bounds:

```
def boundedSubtype
    (D' : D → Prop)
    (a : Subtype D1) (b : Subtype D2) :
    (Subtype D') → Prop :=
        (λ x =>
            (a : D) ≤ (x : D) ∧ (x : D) ≤ (b : D))
```

*Note:* We also pass a predicate $D'$ which only functions to differentiate between the two cases, namely a subtype of elements of $S_{D_1}$ or of $S_{D_2}$.
The expressions of type $(x : D)$ in this context, are also used to denote the coercion of the subtype to the parent type.

The structures defined up to here suffice in order to formulate and prove the first Proposition, namely **Proposition 8**. For the rest of the Propositions though, some more definitions are required.

**Definition 3.5. Information Ordering**
Given $(x,y), (x',y') \in S_{D_1} \times S_{D_2}$, we will write $(x,y) \precsim (x',y')$ if $x \leq x'$ and $y' \leq y$

Creating a custom ordering for a specific type can be done through the LE class:

```
instance : LE (Subtype D1 × Subtype D2) :=
    {le := λ d d' => d.1 ≤ d'.1 ∧ d'.2 ≤ d.2}
```

*Note:* Since $d$ and $d'$ are of type product in the above code, the notation $d.1$, $d.2$ is used to denote the first and the second element of the pair respectively.

**Definition 3.6. Ordered Product**
We will write: $S_{D_1} \otimes S_{D_2} = \{(x,y) \mid x \in S_{D_1},\ y \in S_{D_2},\ x \leq y\}$

So, Ordered Product basically restricts the product type to include only the pairs that are ordered. It is in other words a subtype of the product type and thus we can create it using a predicate.

```
def ordered_product :
    (Subtype D1 × Subtype D2) → Prop := λ d => (d.1 : D) ≤ (d.2 : D)
```

We will be using the custom notation $\otimes$ to denote this predicate. The defined subtype then, can be expressed as:

```
{x : Subtype D1 × Subtype D2 // ⊗x}
```

*Note:* The code is skipped from the two following definitions due to its triviality.

**Definition 3.7. Consistent Approximating Operator**
A function $A : S_{D_1} \otimes S_{D_2} \to S_{D_1} \otimes S_{D_2}$ is called a consistent approximating operator if it is $\precsim$-monotonic
*Note:* In the Lean code the above assertion will often appear with the name $conA$.

**Definition 3.8. Reliable Pair**
A pair $(a, b) \in S_{D_1} \otimes S_{D_2}$ will be called $A$-reliable if $(a, b) \precsim A(a, b)$

# 4. PROPOSITION 8

**Proposition 8.**

For all $a \in S_{D_1}$ and $b \in S_{D_2}$, the sets $[\bot, b]_{L_1}$ and $[a, \top]_{L_2}$ are Complete Lattices.

It suffices to prove that the bounded subtypes are Complete Semilattices which in turn means they are Complete Lattices (**Theorem 1.6.**)

### Proposition 8. in Lean

```
def Proposition_8_A :
    CompleteLattice {x // (boundedSubtype _ _ _ D1 L1.bot b) x}

def Proposition_8_B :
    CompleteLattice {x // (boundedSubtype _ _ _ D2 a L2.top) x}
```

The expression of the proposition is quite self-explanatory. We just want our functions to return a complete lattice class type of our bounded types.

### Proposition 8. Proof

First we will prove that $[\bot, b]_{L_1}$ is a Complete Semilattice of supremum.

1. Let $S$ be a set of elements of type $[\bot, b]_{L_1}$
2. $\forall\ (x \in S),\ x \leq b,$ by definition of the type $[\bot, b]_{L_1}$
3. $\bigvee_{L_1} S \leq b,$ by Interlattice LUB
4. $\bot \leq \bigvee_{L_1} S,$ by the Bottom element property
5. $\bigvee_{L_1} S \in [\bot, b]_{L_1}$

Since the supremum operation is closed within this subtype and we know that every property of the operation itself holds, given that it holds for the initial type $S_{D_1}$, $[\bot, b]_{L_1}$ is a Complete Semilattice and a Complete Lattice as thus.

Now, we shall prove that $[a, \top]_{L_2}$ is a Complete Semilattice of infimum.

1. Let $S$ be a set of elements of type $[a, \top]_{L_2}$
2. $\forall\ (x \in S),\ a \leq x,$ by definition of the type $[a, \top]_{L_2}$
3. $a \leq \bigwedge_{L_2} S,$ by Interlattice GLB
4. $\bigwedge_{L_2} S \leq \top,$ by the Top element property
5. $\bigwedge_{L_2} S \in [a, \top]_{L_2}$

Similarly, since the infimum operation is closed within this subtype, $[a, \top]_{L_2}$ is a Complete Lattice.

The Lean code follows this exact convention for both proofs:

- Proving that the supremum/infimum operation is closed for the subtypes
- Creating the Complete Semilattices using the methods of the original Complete Lattice structures $L_1$ and $L_2$
- Using the Mathlib equivalent of the **Theorem 1.6.**

# 5. PROPOSITION 9

**Proposition 9.**

Let $(a, b) \in S_{D_1} \otimes S_{D_2}$ and $A : S_{D_1} \otimes S_{D_2} \to S_{D_1} \otimes S_{D_2}$ be a consistent approximating operator and assume that $(a, b)$ is $A$-reliable. Then:

1. $\forall\, x \in [\bot, b]_{L_1}$ , it holds $\bot \leq A(x, b)_1 \leq b$.
2. $\forall\, x \in [a, \top]_{L_2}$ , it holds $a \leq A(a, x)_2 \leq \top$.

**Proposition 9. in Lean**

```
def Proposition_9_A : ∀ x :
    {x // (boundedSubtype _ _ _ D1 L1.bot b) x},
    L1.bot ≤ (A ⟨(x, b), x.prop.2⟩).val.1 ∧
    (A ⟨(x, b), x.prop.2⟩).val.1 ≤ (b : D)

def Proposition_9_B : ∀ x :
    {x // (boundedSubtype _ _ _ D2 a L2.top) x},
    a ≤ (A ⟨(a, x), x.prop.1⟩).val.2.val ∧
    (A ⟨(a, x), x.prop.1⟩).val.2.val ≤ (L2.top : D)
```

Where:

- ⟨(x, b), x.prop.2⟩) is the assertion that $(x, b)$ is an ordered product (needed to be able to interact with $A$)
- (A ⟨(x, b), x.prop.2⟩).val returns the value of the function (which is $S_{D_1} \times S_{D_2}$, in contrast to returning its property which is that A.1 ≤ A.2)
- (A ⟨(x, b), x.prop.2⟩).val.1 returns the first element of the product
- For subtypes in general, .val is another way of expressing coercion

**Proposition 9. Proof**

The Lean proofs follow the exact convention showcased below.
For the first part, we have:

1. Let $a^* = \bigvee_{L_1} [\bot, b]_{L_1}$
2. $a^* \leq b,$ by Interlattice LUB (need to make sure that the pair $(a^*, b)$ is in the ordered product domain)
3. $\forall\, x \in [\bot, b]_{L_1},\ (x, b) \precsim (a^*, b),$ by
   - $x \leq a^*,$ by $a^* = \bigvee_{L_1} [\bot, b]_{L_1}$
   - $b \leq b,$ by reflexivity
4. $\forall\, x \in [\bot, b]_{L_1},\ A(x, b) \precsim A(a^*, b),$ by $A$ $\precsim$-monotonic
5. $A(a, b) \precsim A(a^*, b),$ by $a \in [\bot, b]_{L_1}$ on (4)
6. $A(a^*, b)_1 \leq A(a, b)_2,$ by transitivity on

- $A(a^*, b)_1 \leq A(a^*, b)_2,$ by $A$ definition
- $A(a^*, b)_2 \leq A(a, b)_2,$ by $A(a, b) \precsim A(a^*, b)$ (5)

7. $A(a, b)_2 \leq b,$ by $(a, b)$ $A$-reliable
8. $\forall\, x \in [\bot, b]_{L_1},\ A(x, b)_1 \leq A(a^*, b)_1,$ by $A(x, b) \precsim A(a^*, b)$ (4)
9. $\forall\, x \in [\bot, b]_{L_1},\ A(x, b)_1 \leq A(a, b)_2,$ by transitivity on
   - $A(x, b)_1 \leq A(a^*, b)_1,$ by (8)
   - $A(a^*, b)_1 \leq A(a, b)_2,$ by (6)
10. $\forall\, x \in [\bot, b]_{L_1},\ A(x, b)_1 \leq b,$ by transitivity on
    - $A(a, b)_2 \leq b,$ by (7)
    - $A(x, b)_1 \leq A(a, b)_2,$ by (9)
11. Thus $\forall\, x \in [\bot, b]_{L_1},\ \bot \leq A(x, b)_1 \leq b,$ by
    - $\forall\, x \in [\bot, b]_{L_1},\ \bot \leq A(x, b)_1,$ by Bottom property
    - $A(x, b)_1 \leq b,$ by (10)

Similarly, for the second part we have:

1. Let $b^* = \bigwedge_{L_2} [a, \top]_{L_2}$
2. $a \leq b^*,$ by Interlattice GLB
3. $\forall\, x \in [a, \top]_{L_2},\ (a, x) \precsim (a, b^*),$ by
   - $a \leq a,$ by reflexivity
   - $b^* \leq x,$ by $b^* = \bigwedge_{L_2} [a, \top]_{L_2}$
4. $\forall\, x \in [a, \top]_{L_2},\ A(a, x) \precsim A(a, b^*),$ by $A$ $\precsim$-monotonic
5. $A(a, b) \precsim A(a, b^*),$ by $b \in [a, \top]_{L_2}$ on (4)
6. $a \leq A(a, b)_1,$ by $(a, b)$ $A$-reliable
7. $A(a, b)_1 \leq A(a, b^*)_2,$ , by transitivity on
   - $A(a, b)_1 \leq A(a, b^*)_1,$ by $A(a, b) \precsim A(a, b^*),$ (5)
   - $A(a, b^*)_1 \leq A(a, b^*)_2,$ by $A$ definition
8. $\forall\, x \in [a, \top]_{L_2},\ A(a, b)_1 \leq A(a, x)_2,$ by transitivity on
   - $A(a, b)_1 \leq A(a, b^*)_2,$ by (7)
   - $A(a, b^*)_2 \leq A(a, x)_2,$ by $A(a, x) \precsim A(a, b^*)$ (4)
9. $\forall\, x \in [a, \top]_{L_2},\ a \leq A(a, x)_2,$ by transitivity on
   - $a \leq A(a, b)_1,$ by $(a, b),$ by (6)
   - $A(a, b)_1 \leq A(a, x)_2,$ by (8)
10. Thus $\forall\, x \in [a, \top]_{L_2},\ a \leq A(x, b)_2 \leq \top,$ by
    - $a \leq A(a, x)_2,$ by (9)
    - $A(x, b)_2 \leq \top$ by Top property

# 6. PROPOSITION 10

Proposition 8. asserts that $([\bot, b]_{L_1}, \leq)$ and $([a, \top]_{L_2}, \leq)$ are complete lattices. If we can prove that the operators $A(\cdot, b)_1$ and $A(a, \cdot)_2$ are monotonic in their respective domains, then they have least fixpoints in the corresponding lattices, by Knaster-Tarski.

**Proof** $A(\cdot, b)_1$ **monotone in** $[\bot, b]_{L_1}$
For $x, y \in [\bot, b]_{L_1}, x \leq y \Rightarrow (x, b) \precsim (y, b) \Rightarrow A(x, b) \precsim A(y, b) \Rightarrow A(x, b)_1 \leq A(y, b)_1$

**Proof** $A(a, \cdot)_2$ **monotone in** $[a, \top]_{L_2}$
For $x, y \in [a, \top]_{L_2}, x \leq y \Rightarrow (a, y) \precsim (a, x) \Rightarrow A(a, y) \precsim A(a, x) \Rightarrow A(a, x)_2 \leq A(a, y)_2$

Thus the operators have least fixpoints. We can now define the stable revision operator.

**Definition 6.1. Stable Revision Operator**
The stable revision operator is defined as:
$C_A(a, b) = (b\downarrow, a\uparrow) = (lfp(A(\cdot, b)_1),\ lfp(A(a, \cdot)_2))$

The operation $b\downarrow$ can be expressed in Lean as:

```
@OrderHom.lfp
    {x // (boundedSubtype _ _ _ D1 L1.bot b) x}
    (Proposition_8_A b interlub)
    (A₁OrderHom ...)
```

Where:

- OrderHom.lfp is the Mathlib function that returns the least fixpoint of a monotone function
- {x // (boundedSubtype _ _ _ D1 L1.bot b) x} is the domain of the monotone function
- Proposition 8 is of course used to prove that the domain is a complete lattice
- $A_1$OrderHom is a struct that holds $A(\cdot, b)_1$ along with the proof that it is monotone
- The dots (. . . ) are not standard Lean notation and are just used for clarity

Similarly, for $a\uparrow$ we have:

```
@OrderHom.lfp
    {x // (boundedSubtype _ _ _ D2 a L2.top) x}
    (Proposition_8_B a interglb)
    (A₂OrderHom ...)
```

*Note:* The elements $a\uparrow$ and $b\downarrow$ appear in the code with the notation $rOa$ and $rOb$, which stand for revision operator $a$ and revision operator $b$.

### Proposition 10.

Let $A : S_{D_1} \otimes S_{D_2} \to S_{D_1} \otimes S_{D_2}$ be a consistent approximating operator. For every $A$-reliable pair $(a, b)$:

1. $b\downarrow \leq b$
2. $a \leq a\uparrow$
3. $a\uparrow \leq b$
4. $b\downarrow \leq a\uparrow$ (i.e. $(b\downarrow, a\uparrow) \in S_{D_1} \otimes S_{D_2}$)

### Proposition 10. in Lean

```
def Proposition_10  :
    (rOb ...) ≤ b.val ∧
    a.val ≤ (rOa ...) ∧
    (rOa ...) ≤ b.val ∧
    (rOb ...) ≤ (rOa ...).val.val
```

Again, the following proof is structured in the image of the Lean code.

### Proposition 10. Proof

1. $a\uparrow \in$ lowerBounds $\{x \in [a, \top]_{L_2} \mid A(a, x)_2 \leq x\}$, by Knaster-Tarski ($a\uparrow$ being the least fixpoint of $(A(a, \cdot)_2)$)
2. $b \in \{x \in [a, \top]_{L_2} \mid A(a, x)_2 \leq x\}$, by $A(a, b)_2 \leq b$ (since $(a, b)$ is $A$-reliable)
3. Thus $a\uparrow \leq b$, by $a\uparrow \in$ lowerBounds
4. Let $a^* = \bigvee_{L1}\{(x : S_{D_1}) \mid x \leq a\uparrow\}$
5. $a \in \{(x : S_{D_1}) \mid x \leq a\uparrow\}$, by $(a\uparrow : [a, \top]_{L_2})$
6. $a \leq a^*$, by $a \leq \bigvee_{L1}\{(x : S_{D_1}) \mid x \leq a\uparrow\} = a^*$
7. $a^* \leq a\uparrow$, by Interlattice LUB
8. $a^* \leq b$, by transitivity on
   - $a^* \leq a\uparrow$, by (7)
   - $a\uparrow \leq b$, by (3)
9. $(a^*, b) \precsim (a^*, a\uparrow)$, by
   - $a^* \leq a^*$
   - $a\uparrow \leq b$
10. $A(a^*, b) \precsim A(a^*, a\uparrow)$, by $A \precsim$-monotonicity
11. $(a, a\uparrow) \precsim (a^*, a\uparrow)$, by
    - $a \leq a^*$, by (6)

- $a\uparrow \leq a\uparrow,$ by reflexivity

12. $A(a, a\uparrow) \precsim A(a^*, a\uparrow),$ by $A \precsim$-monotonicity
13. $A(a^*, b)_1 \leq A(a^*, a\uparrow)_2,$ by transitivity on
    - $A(a^*, b)_1 \leq A(a^*, a\uparrow)_1,$ by $A(a^*, b) \precsim A(a^*, a\uparrow)$ (10)
    - $A(a^*, a\uparrow)_1 \leq A(a^*, a\uparrow)_2,$ by $A$ definition
14. $A(a^*, b)_1 \leq A(a, a\uparrow)_2,$ by transitivity on
    - $A(a^*, b)_1 \leq A(a^*, a\uparrow)_2,$ by (13)
    - $A(a^*, a\uparrow)_2 \leq A(a, a\uparrow)_2,$ by $A(a, a\uparrow) \precsim A(a^*, a\uparrow)$ (12)
15. $A(a, a\uparrow)_2 = a\uparrow,$ by $a\uparrow$ least fixpoint $A(a, \cdot)_2$
16. Thus $A(a^*, b)_1 \leq a\uparrow$
17. And thus $A(a^*, b)_1 \in \{(x : S_{D_1}) \mid x \leq a\uparrow\}$
18. $A(a^*, b)_1 \leq a^*,$ by $a^* = \bigvee_{L1}\{(x : S_{D_1}) \mid x \leq a\uparrow\}$
19. Thus $a^* \in \{(x : [\bot, b]_{L_1}) \mid A(x, b)_1 \leq x\}$
20. $b\downarrow \in$ lowerBounds $\{(x : [\bot, b]_{L_1}) \mid A(x, b)_1 \leq x\},$ by Knaster-Tarski
21. $b\downarrow \leq a^*,$ by $a^* \in \{(x : [\bot, b]_{L_1}) \mid A(x, b)_1 \leq x\}$ and $b\downarrow$ lower bound
22. $b\downarrow \leq b,$ by transitivity on
    - $b\downarrow \leq a^*,$ , by (21)
    - $a^* \leq b,$ , by (8)
23. $a \leq a\uparrow$, trivially from $a\uparrow$ domain
24. $a\uparrow \leq b,$ by line (3)
25. $b\downarrow \leq a\uparrow,$ by transitivity on
    - $b\downarrow \leq a^*,$ by line (20)
    - $a^* \leq a\uparrow,$ by line (7)

# 7. PROPOSITION 11

Before we can talk about Proposition 11, we need one more definition.

**Definition 7.1. Prudent Pair**
An $A$-reliable approximation $(a, b)$ is $A$-prudent if $a \leq b\downarrow$

We can express this in Lean as:

```
def prudent (ab : {x // reliable A x}) : Prop :=
    let a := ab.1.1.1
    a.val ≤ rOb ...
```

Where:

- ab holds a pair that is $A$-reliable
- ab.1 is a more concise notation for ab.val, which returns the value of object
- ab.2 is notation for ab.property, which returns the property of the object
- In our case the value is an element of type of ordered pair
- And the property is of course the reliability
- Another step futher and ab.1.1 gives us a product element
- In an analogous way ab.1.2 gives us the proof that $a \leq b$
- Last step, we choose the first element of the product, by calling ab.1.1.1
- a.val $\leq$ rOb... asserts that $a \leq b\downarrow$ (as mentioned before, a.val is another way of expressing the coercion of a subtype, which happens to interfere less with the notation of the paper[1])

**Proposition 11.**

Let $A : S_{D_1} \otimes S_{D_2} \to S_{D_1} \otimes S_{D_2}$ be a consistent approximating operator and let $(a, b) \in S_{D_1} \otimes S_{D_2}$ be $A$-prudent. Then, $(a, b) \precsim (b\downarrow, a\uparrow)$ and $(b\downarrow, a\uparrow)$ is $A$-prudent.

**Proposition 11. in Lean**

We split Proposition 11 in two parts, one that asserts the inequality and the $A$-reliability of $(b\downarrow, a\uparrow)$ (a pressuposition of prudence) and the second one which asserts that $(b\downarrow, a\uparrow)$ is, in fact, $A$-prudent.

```
def Proposition_11_A :
    (a, b) ≾ ((rOb ...).val,(rOa ...).val) ∧
    reliable A ⟨((rOb ...).val, (rOa ...).val),
    -- Proposition_10.2.2.2 holds its last assertion, i.e. the proof that b↓ ≤
    a↑, which is a precondition of the domain of A
    (Proposition_10 ...).2.2.2⟩
```

```
def Proposition_11_B : prudent
    (rOb ...).val
    (rOa ...).val
    interlub
    (Proposition_10 ...).2.2.2
    A conA
    -- Proposition_11_A.2 holds the reliability assertion of the revised pair,
    which is a precondition of prudence
    (Proposition_11_A ...).2
```

A somewhat interesting thing to note here is that the assertion $b\downarrow \leq a\uparrow$ is crucial, since we couldn't reason about what the function $A$ does to the pair, if we didn't know that the pair was in fact in the ordered product domain. That's why Proposition 10 is called in the very formulation of Proposition 11. And of course, the second part of Proposition 11 needs the first one in order to be expressed.

### Proposition 11 Proof

1. $(a, b) \precsim (b\downarrow, a\uparrow),$ by
    - $a \leq b\downarrow,$ by $(a, b)$ $A$-prudent assumption
    - $a\uparrow \leq b,$ by Proposition 10 (3)
2. $(b\downarrow, b) \precsim (b\downarrow, a\uparrow),$ by
    - $b\downarrow \leq b\downarrow,$ by reflexivity
    - $a\uparrow \leq b,$ by Proposition 10 (3)
3. $A(b\downarrow, b) \precsim A(b\downarrow, a\uparrow),$ by $A$ $\precsim$-monotonicity
4. $A(b\downarrow, b)_1 = b\downarrow,$ by $b\downarrow$ least fixpoint of $A(\cdot, b)_1$
5. $b\downarrow \leq A(b\downarrow, a\uparrow)_1,$ by
    - $A(b\downarrow, b)_1 \leq A(b\downarrow, a\uparrow)_1,$ by $A(b\downarrow, b) \precsim A(b\downarrow, a\uparrow)$ (3)
    - $A(b\downarrow, b)_1 = b\downarrow$, by (4)
6. $(a, a\uparrow) \precsim (b\downarrow, a\uparrow),$ by
    - $a \leq b\downarrow,$ by $(a, b)$ A-prudent assumption
    - $a\uparrow \leq a\uparrow,$ by reflexivity
7. $A(a, a\uparrow) \precsim A(b\downarrow, a\uparrow),$ by $A$ $\precsim$-monotonicity
8. $A(a, a\uparrow)_2 = a\uparrow,$ by $a\uparrow$ least fixpoint of $A(a, \cdot)_2$
9. $A(b\downarrow, a\uparrow)_2 \leq a\uparrow,$ by
    - $A(b\downarrow, a\uparrow)_2 \leq A(a, a\uparrow)_2,$ by $A(a, a\uparrow) \precsim A(b\downarrow, a\uparrow)$ (7)
    - $A(a, a\uparrow)_2 = a\uparrow,$ , by (8)
10. $(b\downarrow, a\uparrow) \precsim A(b\downarrow, a\uparrow)$ by
    - $b\downarrow \leq A(b\downarrow, a\uparrow)_1,$ by (5)

- $A(b{\downarrow}, a{\uparrow})_2 \leq a{\uparrow},$ by (9)

11. $(b{\downarrow}, a{\uparrow})$ $A$-reliable, by (10)
12. Let $(a{\uparrow}){\downarrow} = lfp(A(\cdot, a{\uparrow})_1)$
13. $\forall\, x \in [\bot, a{\uparrow}]_{L_1},\ (x, b) \precsim (x, a{\uparrow}),$ by
    - $x \leq x,$ by reflexivity
    - $a{\uparrow} \leq b,$ by Proposition 10 (3)
14. $\forall\, x \in [\bot, a{\uparrow}]_{L_1},\ x \leq b,$ by transitivity on
    - $x \leq a{\uparrow},$ by domain of $x$
    - $a{\uparrow} \leq b,$ by Proposition 10 (3)
15. $\forall\, x \in [\bot, a{\uparrow}]_{L_1},\ A(x, b)_1 \leq A(x, a{\uparrow})_1,$ by $A \precsim$-monotonicity
16. $\forall\, x \in \{x \mid A(x, a{\uparrow})_1 \leq x\},\ A(x, b)_1 \leq x,$ by transitivity on
    - $A(x, b)_1 \leq A(x, a{\uparrow})_1,$ by (15)
    - $A(x, a{\uparrow})_1 \leq x,$ by domain of $x$
17. $b{\downarrow} \in$ lowerBounds $\{x \mid A(x, b)_1 \leq x\},$ by Knaster-Tarski
18. $(a{\uparrow}){\downarrow} \in \{x \mid A(x, a{\uparrow})_1 \leq x\},$ , by Knaster-Tarski
19. $A((a{\uparrow}){\downarrow}, b)_1 \leq (a{\uparrow}){\downarrow},$ by
    - $\forall\, x \in \{x \mid A(x, a{\uparrow})_1 \leq x\},\ A(x, b)_1 \leq x,$ by (16)
    - $(a{\uparrow}){\downarrow} \in \{x \mid A(x, a{\uparrow})_1 \leq x\},$ by (18)
20. $(a{\uparrow}){\downarrow} \in \{x \mid A(x, b)_1 \leq x\},$ by (19)
21. Thus $b{\downarrow} \leq (a{\uparrow}){\downarrow},$ by $b{\downarrow} \in$ lowerBounds $\{x \mid A(x, b)_1 \leq x\}$

# 8. PROPOSITION 12

**Proposition 12.**

Let $\{(a_\kappa, b_\kappa)\}_{\kappa<\lambda}$, where $\lambda$ is an ordinal, be a chain in $S_{D_1} \otimes S_{D_2}$ ordered by the $\precsim$ relation. Then:

1. $\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\} \leq \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\}$
2. The least upper bound of the chain with respect to $\precsim$ exists and it is equal to $(\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\}, \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\})$

Before we get into the specifics of Proposition 12, we will need the definitions for chains and chain complete posets. Mathlib provides definitions for omega complete partial orders but those support chains (as the name would suggest) up to the smallest infinite ordinal whilst, as we shall soon see, Proposition 12 needs to reason about chains up so some limit ordinal.

Without straying too much from the Mathlib definitions, we can express the chain function as a monotone function from ordinals to elements of the partial order domain, instead of a function from naturals to elements:

```
def Chain (D : Type u) [Preorder D] :=
    Ordinal →o D
```

*Note:* The →o symbol denotes monotonicity.

In the same spirit we can define our chain complete partial order class similar to the omega complete partial order class, with the supremum defined up to some limit ordinal:

```
class ChainCompletePartialOrder (D : Type*) extends PartialOrder D where
    LimitOrdinal : Ordinal
    Is_Limit : LimitOrdinal.IsLimit
    cSup : Chain D → D
    le_cSup : ∀ c : Chain D,
    ∀ (i : {x | x < LimitOrdinal}), c i ≤ cSup c
    cSup_le : ∀ (c : Chain D) (x),
    (∀ (i : {x | x < LimitOrdinal}), c i ≤ x) → cSup c ≤ x
```

**Proposition 12. in Lean**

This proposition is a again split in two parts, since this supremum/infimum pair being ordered is a presupposition to even be able to reason about it as being in the domain of the chain. We first define the necessary extra variable baggage for this very formulation.

```
variable
    (limitOrdinal : Ordinal)
    (isLimit : limitOrdinal.IsLimit)
    (chain :
    (@ChainCompletePartialOrder.Chain
    {x : Subtype D1 × Subtype D2 | ⊗x} (
    @InfoPoset _ _ _ O).toPreorder))
```

Then Proposition 12. can be formulated as:

```
def Proposition_12_A :
    L1.sSup {(chain i).val.1 | i < limitOrdinal} ≤
    (L2.sInf {(chain i).val.2 | i < limitOrdinal}).val

def Proposition_12_B :
    ChainCompletePartialOrder
    {x : Subtype D1 × Subtype D2 | ⊗x}
```

## Proposition 12. Proof

1. $\forall\ u,\ k$ ordinals, $a_k \leq b_u,$ by cases
    - $u \leq k$
        - $a_k \leq b_k,$ by $(a_k, b_k)$ ordered pair
        - $(a_u, b_u) \precsim (a_k, b_k),$ by chain monotone
        - $b_k \leq b_u,$ by $(a_u, b_u) \precsim (a_k, b_k)$
        - $a_k \leq b_u,$ by transitivity
    - $k \leq u,$ by $k < u \to k \leq u$
        - $a_u \leq b_u,$ by $(a_u, b_u)$ ordered pair
        - $(a_k, b_k) \precsim (a_u, b_u),$ by chain monotone
        - $a_k \leq a_u,$ by $(a_k, b_k) \precsim (a_u, b_u)$
        - $a_k \leq b_u,$ by transitivity
2. Thus $\forall$ ($u$ : ordinal) , $b_u \in$ upperBounds $\{a_\kappa \mid \kappa \leq \lambda\}$
3. $\forall\ u,\ \bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\} \leq b_u,$ by Intelattice LUB property
4. $\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\} \leq \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\},$ by Interlattice GLB property
5. Assuming a limit ordinal $\lambda$
    - $\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\}$ is the supremum of the $\{a_\kappa \mid \kappa < \lambda\}$ chain
    - $\bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\}$ is the infimum of the $\{b_\kappa \mid \kappa < \lambda\}$ chain
6. Thus $(\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\},\ \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\})$ is the least upper bound of the $\{(a_\kappa, b_\kappa)\}_{\kappa<\lambda}$ chain ordered by $\precsim$

*Note:* The pair $(\bigvee_{L_1}\{a_\kappa \mid \kappa < \lambda\},\ \bigwedge_{L_2}\{b_\kappa \mid \kappa < \lambda\})$ we will denote as $(a^\infty, b^\infty)$.

# 9. PROPOSITION 13

**Proposition 13.**

Let $A : S_{D_1} \otimes S_{D_2} \to S_{D_1} \otimes S_{D_2}$ be a consistent approximating operator and let $\{(a_\kappa, b_\kappa)\}_{\kappa<\lambda}$, where $\lambda$ is an ordinal, be a chain of $A$-prudent pairs from $S_{D_1} \otimes S_{D_2}$. Then $\bigvee_{\precsim}\{(a_\kappa, b_\kappa)\}_{\kappa<\lambda}$, is $A$-prudent.

In order to prove prudence, we first need to prove reliability and so again we split this proposition into two parts.

*Note:* The second part of Proposition 12 holds the structure which is responsible for giving us the supremum of a certain chain.

**Proposition 13. in Lean**

```
def Proposition_13_A
   -- Assuming that every element of the chain is reliable
   (A_reliable : ∀ i, reliable A (chain i.val)) :
   -- Prove that the supremum of the chain is reliable
   reliable A ((Proposition_12_B ...).cSup chain)

def Proposition_13_B
   -- Assuming that every element of the chain is prudent
   (prudent_chain :
   ∀ i, prudent interlub A conA ⟨chain i, A_reliable limitOrdinal i⟩) :
   -- Prove that the supremum of the chain is prudent
   prudent interlub A conA
       ⟨((Proposition_12_B ···).cSup chain), (Proposition_13_A ···)⟩
```

**Proposition 13. Proof**

$\forall\, i \leq \lambda$ :

1. $(a_i, b_i) \precsim (a^\infty, b^\infty)$, by $(a^\infty, b^\infty)$ supremum of chain
2. $A(a_i, b_i) \precsim A(a^\infty, b^\infty)$, by $A$ $\precsim$-monotone
3. $(a_i, b_i) \precsim A(a^\infty, b^\infty)$, by transitivity on:
    - $(a_i, b_i) \precsim A(a_i, b_i)$, by $(a_i, b_i)$ $A$-reliable
4. Thus $A(a^\infty, b^\infty)$ is upper bound of chain
5. $(a^\infty, b^\infty) \precsim A(a^\infty, b^\infty)$, by $(a^\infty, b^\infty)$ least upper bound of chain
6. Thus $(a^\infty, b^\infty)$ is $A$-reliable
7. Let $b_i{\downarrow} = lfp(A(\cdot, b_i)_1)$
8. $b_i{\downarrow} \in$ lowerBounds $\{x \mid A(x, b_i)_1 \leq x\}$, by Knaster-Tarski
9. Let $(b^\infty){\downarrow} = lfp(A(\cdot, b^\infty)_1)$
10. $(b^\infty){\downarrow} \in \{x \mid A(x, b^\infty)_1 \leq x\}$, by Knaster-Tarski
11. $\forall\, x \in [\bot, b^\infty]$, $(x, b_i) \precsim (x, b^\infty)$, by

- $x \leq x,$ by reflexivity
- $b^\infty \leq b_i,$ by $b^\infty = \bigwedge_{L_2}\{b_\kappa \mid \kappa \leq \lambda\}$

12. $\forall\, x \in [\bot, b^\infty],\ A(x, b_i) \precsim A(x, b^\infty),$ by A $\precsim$-monotone
13. $\forall\, x \in [\bot, b^\infty],\ \ x \leq b_i,$ by transitivity ($b^\infty \leq b_i,$ (11.2))
14. $\forall\, x \in \{x \mid A(x, b^\infty)_1 \leq x\},\ A(x, b_i)_1 \leq x,$ by transitivity on
    - $A(x, b_i)_1 \leq A(x, b^\infty)_1,$ by $A(x, b_i) \precsim A(x, b^\infty)$
    - $A(x, b^\infty)_1 \leq x$
15. $A((b^\infty)\downarrow, b_i)_1 \leq (b^\infty)\downarrow,$ by
    - $\forall\, x \in \{x \mid A(x, b^\infty)_1 \leq x\},\ A(x, b_i)_1 \leq x,$ by (14)
    - $(b^\infty)\downarrow \in\ \{x \mid A(x, b^\infty)_1 \leq x\},$ by (10)
16. $\forall\, x \in [\bot, b_i],\ A(x, b_i)_1 \leq x \rightarrow b_i\downarrow \leq x,$ by $b_i\downarrow$ least upper bound of $\{x \mid A(x, b_1)_1 \leq x\}$
17. $b_i\downarrow \leq (b^\infty)\downarrow,$ by
    - $\forall\, x \in [\bot, b_i],\ A(x, b_i)_1 \leq x \rightarrow b_i\downarrow \leq x,$ by (16)
    - $A((b^\infty)\downarrow, b_i)_1 \leq (b^\infty)\downarrow,$ by (15)
    - $(b^\infty)\downarrow \leq b_i,$ by transitivity on
      - $(b^\infty)\downarrow \leq b^\infty),$ by the domain of $A(\cdot, b^\infty)_1$
      - $b^\infty \leq b_i,$ by (11.2)
18. $a_i \leq b_i\downarrow,$ by prudent chain
19. $a_i \leq (b^\infty)\downarrow,$ by transitivity $(b_i\downarrow \leq (b^\infty)\downarrow)$
20. $\forall (x_a, x_b), (a_i, b_i) \precsim (x_a, x_b) \rightarrow a^\infty \leq x_a,$ by $a^\infty$ least upper bound
21. $(a_i, b_i) \precsim ((b^\infty)\downarrow, b^\infty),$ by
    - $a_i \leq (b^\infty)\downarrow,$ by (19)
    - $b^\infty \leq b_i,$ by (11.2)
22. $a^\infty \leq (b^\infty)\downarrow,$ by
    - $\forall (x_a, x_b), (a_i, b_i) \precsim (x_a, x_b) \rightarrow a^\infty \leq x_a,$ , (20)
    - $(a_i, b_i) \precsim ((b^\infty)\downarrow, b^\infty),$ by (22)
23. Thus $(a^\infty, b^\infty)$ is prudent

# 10. CONCLUSION

While this work has focused on specific propositions, the methodology can be extended to other formalizations, even outside the area of logic programming. Future work could explore the formalization of additional theorems, such as the last two vital theorems of the extension, which are constructed by putting together all those propositions.

Another interesting direction would be to formalize this twin lattice structure and all of its properties as a distinct class in Lean, carefully defining all the necessary structures with optimal code practices, so that it could be integrated into the Mathlib library. That would indeed be a strong contribution to both proof assistants and the field of approximation fixpoint theory.

About a theoretic direction, there has also been some exploration of matching the creativity of LLMs with the rigor of Lean [5] [4] which is a promising direction. However, this endeavor faces several challenges due to a significant data scarcity problem, specifically lack of mappings natural language proofs to formalized ones —a gap that this thesis indirectly addresses. Then there is also the issue of complexity and performance, as such systems are forced to integrate multiple components (data extraction, interaction with proof assistants, model training, proof search and so on).

Nonetheless, such advancements outline another connection between artificial intelligence and this very topic. And who is to say that a reconciliation between LLMs and non-monotonic logics isn't exactly what we need to take the next step in general artificial intelligence and fully autonomous proof writing systems?

# 11. REPOSITORY

https://github.com/UpTheShipCreek/Extension-of-Consistent-Approximation-Fixpoint-Theory-Proofs

# BIBLIOGRAPHY

[1] Angelos Charalambidis, Panos Rondogiannis, and Ioanna Symeonidou. Approximation fixpoint theory and the well-founded semantics of higher-order logic programs. *CoRR*, abs/1804.08335, 2018.

[2] Marc Denecker, Victor Marek, and Mirosław Truszczyński. *Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning*, pages 127–144. Springer US, Boston, MA, 2000.

[3] Marc Denecker, Victor W. Marek, and Mirosław Truszczyński. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation*, 192(1):84–121, 2004.

[4] Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. Theoreml-lama: Transforming general-purpose llms into lean4 experts, 2024.

[5] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models, 2023.