# Mechanizing Many-Sorted Polyadic Hybrid Logic in the Lean 4 Prover

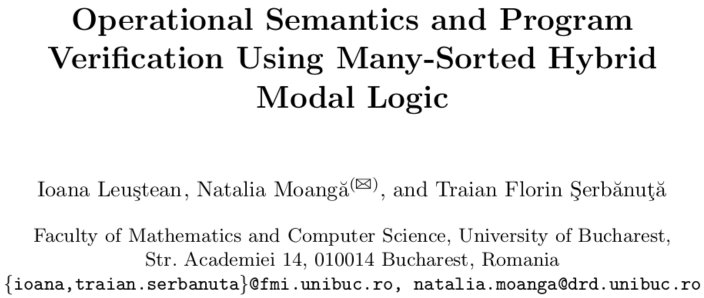
Andrei-Alexandru Oltean

University of Bucharest

11 September 2025

## Mechanizing what?!

- This 2019 paper [1]:



**Operational Semantics and Program Verification Using Many-Sorted Hybrid Modal Logic**

Ioana Leuştean, Natalia Moangă(✉), and Traian Florin Şerbănuţă

Faculty of Mathematics and Computer Science, University of Bucharest,
Str. Academiei 14, 010014 Bucharest, Romania
{ioana,traian.serbanuta}@fmi.unibuc.ro, natalia.moanga@drd.unibuc.ro

## Mechanizing what?!

### Many-Sorted Polyadic Hybrid Logic

```
s ::= 0;
i ::= 0;
while (++i <= n) do
    s ::= s + i
```

## Mechanizing what?!

**Many-Sorted** Polyadic Hybrid Logic

```
s ::= 0;
i ::= 0;
while (++i <= n) do
      s ::= s + i
```

$$\boldsymbol{S} \quad := \quad \{AExp,\ BExp,\ Var,\ Stmt\}$$

## Mechanizing what?!

**Many-Sorted Polyadic Hybrid Logic**

```
s ::= 0;
i ::= 0;
while (++i <= n) do
      s ::= s + i
```

$$
\begin{aligned}
\boldsymbol{S} &:= \{AExp,\ BExp,\ Var,\ Stmt\} \\
\Sigma_{Var,AExp} &:= \{++\} \\
\Sigma_{AExpAExp,AExp} &:= \{+\} \\
\Sigma_{AExpAExp,BExp} &:= \{<=\} \\
\Sigma_{VarAExp,Stmt} &:= \{::=\} \\
\Sigma_{StmtStmt,Stmt} &:= \{;\} \\
\Sigma_{BExpStmt,Stmt} &:= \{while\ \_\ do\ \_\}
\end{aligned}
$$

## Mechanizing what?!

### Many-Sorted Polyadic Hybrid Logic

```
s ::= 0;
i ::= 0;
while (++i <= n) do
      s ::= s + i
```

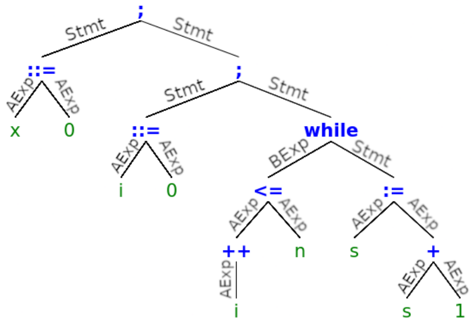
$$
\begin{aligned}
\boldsymbol{S} &:= \{AExp,\ BExp,\ Var,\ Stmt\}\\
\Sigma_{Var,AExp} &:= \{++\}\\
\Sigma_{AExpAExp,AExp} &:= \{+\}\\
\Sigma_{AExpAExp,BExp} &:= \{<=\}\\
\Sigma_{VarAExp,Stmt} &:= \{::=\}\\
\Sigma_{StmtStmt,Stmt} &:= \{;\}\\
\Sigma_{BExpStmt,Stmt} &:= \{while \_ \ do \ \_\}
\end{aligned}
$$

$\boldsymbol{N}_{AExp} := \{0\}$ $\quad$ $\boldsymbol{N}_{Var} := \{s,\ i,\ n\}$

## Mechanizing what?!

### Many-Sorted Polyadic Hybrid Logic

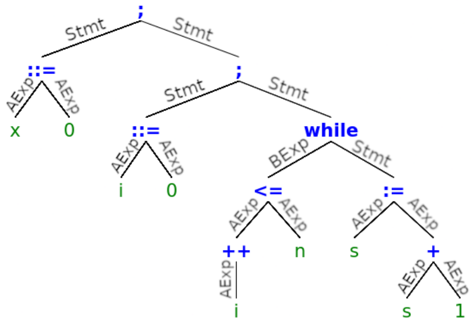
- Programming language AST? Or modal logic AST?



## Mechanizing what?!

**Many-Sorted Polyadic Hybrid Logic**

- Programming language AST? Or modal logic AST?



- **Both.**

## The challenge

1. Can we represent such a many-sorted, polyadic grammar in Lean?

## The challenge

1. Can we represent such a many-sorted, polyadic grammar in Lean?
2. Can we represent its modal semantics?

## The challenge

1. Can we represent such a many-sorted, polyadic grammar in Lean?
2. Can we represent its modal semantics?
3. Can we prove soundness and completeness of the system?

## The challenge

1. Can we represent such a many-sorted, polyadic grammar in Lean?
2. Can we represent its modal semantics?
3. Can we prove soundness and completeness of the system?

## The challenge

1. Can we represent such a many-sorted, polyadic grammar in Lean?
2. Can we represent its modal semantics?
3. Can we prove soundness and completeness of the system?

But all this is already done it the paper. What is the point of the Lean implementation?

## The challenge

1. Can we represent such a many-sorted, polyadic grammar in Lean?
2. Can we represent its modal semantics?
3. Can we prove soundness and completeness of the system?

But all this is already done it the paper. What is the point of the Lean implementation?

- Humans may err; maybe there are errors in the paper;

## The challenge

1. Can we represent such a many-sorted, polyadic grammar in Lean?
2. Can we represent its modal semantics?
3. Can we prove soundness and completeness of the system?

But all this is already done it the paper. What is the point of the Lean implementation?

- Humans may err; maybe there are errors in the paper;
- The resulting formal artifact could be used as a foundation for verifying software and programming languages in the future.

## What we did

1. Implemented the language, its proof system, and its semantics;

## What we did

1. Implemented the language, its proof system, and its semantics;
2. Found and corrected errors in the original axiomatic system;

## What we did

1. Implemented the language, its proof system, and its semantics;
2. Found and corrected errors in the original axiomatic system;
3. Proved soundness of the new system;

## What we did

1. Implemented the language, its proof system, and its semantics;
2. Found and corrected errors in the original axiomatic system;
3. Proved soundness of the new system;
4. Work is in progress for the completeness proof.

## The syntax

**Definition (Signatures with constant nominals)**

A **signature with constant nominals** is a triple $(S, \Sigma, N)$, where:

- $S$ is a non-empty, countable set;
- $\Sigma$ is an $S^* \times S$-indexed family of countable sets;
- $N$ is an $S$-indexed family of non-empty, countable sets.

## The syntax

**Definition (Signatures with constant nominals)**

A **signature with constant nominals** is a triple $(S, \Sigma, N)$, where:

- $S$ is a non-empty, countable set;
- $\Sigma$ is an $S^* \times S$-indexed family of countable sets;
- $N$ is an $S$-indexed family of non-empty, countable sets.

In code:

```
structure Signature (α : Type u) where
    S     : Set α
    «Σ»   : List S → S → Set α
    N     : S → Set α

    sortsCtbl : Encodable S
    opsCtbl (dom range) : Encodable («Σ» dom range)
    nomCtbl (s)         : Encodable (N s)
    sNonEmpty : Inhabited S
```

## The syntax

We further fix sorted sets of propositional variables (*PROP*), non-constant nominals (*NOM*) and state variables (*SVAR*):

```
structure Symbols (α : Type u) where
    signature   : Signature α

    prop : (s : signature.S) → Set α
    nom  : (s : signature.S) → Set α
    svar : (s : signature.S) → Set α

    propCtbl (s) : Encodable (prop s)
    nomCtbl  (s) : Encodable (nom s)
    svarCtbl (s) : Denumerable (svar s)
```

## The syntax

We further fix sorted sets of propositional variables (*PROP*), non-constant nominals (*NOM*) and state variables (*SVAR*):

```
structure Symbols (α : Type u) where
    signature  : Signature α

    prop : (s : signature.S) → Set α
    nom  : (s : signature.S) → Set α
    svar : (s : signature.S) → Set α

    propCtbl (s) : Encodable (prop s)
    nomCtbl  (s) : Encodable (nom s)
    svarCtbl (s) : Denumerable (svar s)
```

### Definition (Formulas)

The set of formulas is given by the grammar:

$$\varphi_s := p \mid j \mid y \mid \neg\varphi_s \mid \varphi_s \vee \varphi_s \mid \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \mid @^s_k \varphi_t \mid \forall x \varphi_s,$$

$p \in PROP_s$, $j \in NOM_s \cup N_s$, $k \in NOM_t \cup N_t$, $y \in SVAR_s$, $x \in SVAR_t$, $\sigma \in \Sigma_{s_1 \ldots s_n, s}$

## The syntax

In code:

```
inductive FormL (symbs : Symbols α) :
    List symbs.signature.S → Type u
| prop : symbs.prop s → FormL symbs [s]
| nom  : symbs.nominal s → FormL symbs [s]
| svar : symbs.svar s → FormL symbs [s]
| appl : symbs.signature.«Σ» (h :: t) s →
            FormL symbs (h :: t) → FormL symbs [s]
| or   : FormL symbs [s] → FormL symbs [s] → FormL symbs [s]
| neg  : FormL symbs [s] → FormL symbs [s]
| at   : symbs.nominal t → FormL symbs [t] → FormL symbs [s]
| bind : symbs.svar t → FormL symbs [s] → FormL symbs [s]
| cons : FormL symbs [s₁] → FormL symbs (s₂ :: t) →
            FormL symbs (s₁ :: s₂ :: t)

abbrev Form (symbs : Symbols α) (s : symbs.signature.S) :=
    FormL symbs [s]
```

## The semantics

In code:

```
def Sat (M : Model symbs) (g : Assignment M)
    (w : WProd M.Fr.W sorts) : FormL symbs sorts → Prop
| .prop p         => w ∈ M.Vp p
| .nom n          => w = M.VNom n
| .svar x         => w = g x
| .appl σ arg     => ∃ w', Sat M g w' arg ∧ ⟨w, w'⟩ ∈ M.Fr.R σ
| .neg φ          => ¬ Sat M g w φ
| .or φ ψ         => Sat M g w φ ∨ Sat M g w ψ
| .at k φ         => let u := M.VNom k;  Sat M g u φ
| .bind x φ       => ∀ g', g'.variant g x → Sat M g' w φ
| .cons φ ψs      => Sat M g w.1 φ ∧ Sat M g w.2 ψs
```

## The proof system

Q: Can we take the following schema as an axiom?

$$\vdash^s \forall x \sigma^{\Box}(\varphi_1, \ldots, \varphi_n) \to \sigma^{\Box}(\varphi_1, \ldots, \forall x \varphi_i, \ldots, \varphi_n) \qquad \text{(Barcan)}$$

## The proof system

Q: Can we take the following schema as an axiom?

$$\vdash^{s} \forall x \sigma^{\Box}(\varphi_1, \ldots, \varphi_n) \to \sigma^{\Box}(\varphi_1, \ldots, \forall x \varphi_i, \ldots, \varphi_n) \tag{Barcan}$$

No! We found a countermodel:

```
theorem BarcanAntecedentTrue  :
    ⟨countermodel, g, w₀⟩ ⊨ barcan_antecedent
theorem BarcanConsequentFalse :
    ⟨countermodel, g, w₀⟩ ⊭ barcan_consequent
```

## The proof system

Q: Can we take the following schema as an axiom?

$$\vdash^s \forall x \sigma^{\Box}(\varphi_1, \ldots, \varphi_n) \to \sigma^{\Box}(\varphi_1, \ldots, \forall x \varphi_i, \ldots, \varphi_n) \qquad \text{(Barcan)}$$

No! We found a countermodel:

```
theorem BarcanAntecedentTrue  :
    ⟨countermodel, g, w₀⟩ ⊨ barcan_antecedent
theorem BarcanConsequentFalse :
    ⟨countermodel, g, w₀⟩ ⊭ barcan_consequent
```

We need a restricted version:

$$\vdash^s \forall x \sigma^{\Box}(\varphi_1, \ldots, \varphi_n) \to \sigma^{\Box}(\varphi_1, \ldots, \forall x \varphi_i, \ldots, \varphi_n), \qquad \text{(Barcan)}$$

if $x$ does not occur free in $\varphi_j$ for $j \neq i$

## The proof system

Q: Can we use the following proof rule?

$$\text{If } \vdash^{s} @^{s}_{j}\varphi, \text{then } \vdash^{s'} \varphi, \quad \text{if } j \text{ does not occur in } \varphi \qquad \text{(Name@)}$$

## The proof system

Q: Can we use the following proof rule?

$$\text{If } \vdash^{s} @_{j}^{s}\varphi, \text{then } \vdash^{s'} \varphi, \quad \text{if } j \text{ does not occur in } \varphi \qquad (\text{Name@})$$

No! Using it on constant nominals leads to unsound derivations. We require:

$$\text{If } \vdash^{s} @_{j}^{s}\varphi, \text{then } \vdash^{s'} \varphi, \quad \text{if } j \notin N \text{ and } j \text{ does not occur in } \varphi \qquad (\text{Name@})$$

## The proof system

Q: Can we use the following proof rule?

$$\text{If } \vdash^{s} @_{j}\sigma(\ldots, k, \ldots) \land @_{k}\varphi \to \psi, \text{ then } \vdash^{s} @_{j}\sigma(\ldots, \varphi, \ldots) \to \psi, \tag{Paste}$$

for $k \neq j$, and $k$ does not occur in $\varphi$ or $\psi$

## The proof system

Q: Can we use the following proof rule?

$$\text{If } \vdash^s @_j\sigma(\ldots, k, \ldots) \wedge @_k\varphi \to \psi, \text{ then } \vdash^s @_j\sigma(\ldots, \varphi, \ldots) \to \psi, \qquad \text{(Paste)}$$

$$\text{for } k \neq j, \text{ and } k \text{ does not occur in } \varphi \text{ or } \psi$$

Corrected:

$$\text{If } \vdash^s @_j\sigma(\ldots, k, \ldots) \wedge @_k\varphi \to \psi, \text{ then } \vdash^s @_j\sigma(\ldots, \varphi, \ldots) \to \psi, \qquad \text{(Paste)}$$

$$\text{for } k \neq j, k \notin N, \text{ and } k \text{ does not occur in } \varphi, \psi, \text{ or the } \ldots \text{ formulas}$$

```
inductive Proof {symbs : Symbols α} (Λ : AxiomSet symbs) :
    (s : symbs.signature.S) → Form symbs s → Type u
  -- Λ:
  | ax     : (ψ : Λ s) → Proof Λ s ψ
  -- Propositional:
  | prop1 φ ψ   : Proof Λ s (φ → (ψ → φ))
  | prop2 φ ψ χ : Proof Λ s ((φ → (ψ → χ)) → (φ → ψ) → (φ → χ))
  | prop3 φ ψ   : Proof Λ s ((∼ψ → ∼φ) → (φ → ψ))
  -- K:
  | k φ ψ χ
      (σ : symbs.signature.«Σ» _ s)
      (C : (φ → ψ).Context χ):
          Proof Λ s (ℋ⟨σ⟩⍰ χ → (ℋ⟨σ⟩⍰ C[φ] → ℋ⟨σ⟩⍰ C[ψ]))
  | mp     : Proof Λ s (φ → ψ) → Proof Λ s φ → Proof Λ s ψ
  | ug {φ : Form symbs s₁}
       (C : φ.Context ψ):
          Proof Λ s₁ φ → Proof Λ s₂ (ℋ⟨σ⟩⍰ ψ)
  -- H(@, ∀):
  -- 1. Axioms about @
  | kAt j φ ψ      : Proof Λ s (ℋ@j (φ → ψ) → (ℋ@j φ → ℋ@j ψ))
  | agree j k φ    : Proof Λ s (ℋ@k (ℋ@j φ) ↔ ℋ@j φ)
  | selfDual j φ : Proof Λ s (ℋ@j φ ↔ ∼ ℋ@j (∼φ))
  | intro j φ      : Proof Λ s (ℋNom j → (φ ↔ ℋ@j φ))
  | back j φ ψ
```

## Soundness and completeness

```
theorem Soundness {Λ : AxiomSet symbs} : ⊢(Λ, s) ψ → ⊨Mod(Λ) ψ
```

Can we do the same for completeness?

## Soundness and completeness

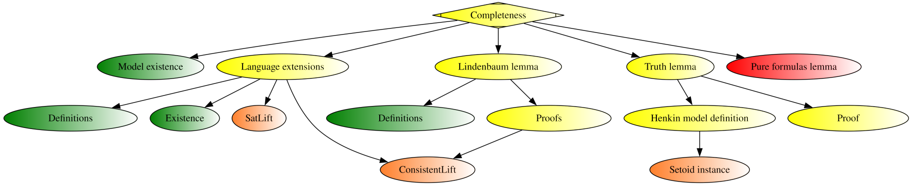
```
theorem Soundness {Λ : AxiomSet symbs} : ⊢(Λ, s) ψ → ⊨Mod(Λ) ψ
```

Can we do the same for completeness?

Yes, but it's more complex. We took a top-down approach.

# Soundness and completeness

State of the completeness formalization:



## Future work

1. Complete the proof of completeness;

## Future work

1. Complete the proof of completeness;
2. Formalize applications to PL semantics;

## Future work

1. Complete the proof of completeness;
2. Formalize applications to PL semantics;
3. DSL for seamless specification of PL's.

Thank you!

## Bibliography I

[1] Ioana Leuștean, Natalia Moangă, and Traian Florin Șerbănuță. "Operational Semantics and Program Verification Using Many-Sorted Hybrid Modal Logic". In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Ed. by Serenella Cerrito and Andrei Popescu. Cham: Springer International Publishing, 2019, pp. 446–476. DOI: 10.1007/978-3-030-29026-9_25.