Adding Sorts to an Isabelle Formalization of Superposition

Balazs Toth Martin Desharnais-Schäfer Jasmin Blanchette

CPP 2026, Rennes, France

LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Superposition

Saturation Calculi

Saturation calculi automatically prove theorems.

Saturation Calculi

Saturation calculi automatically prove theorems.

They start from a set of formulas and repeatedly

- derive new formulas,

Saturation Calculi

Saturation calculi automatically prove theorems.

They start from a set of formulas and repeatedly

- derive new formulas,
- remove redundant ones,

Saturation Calculi

Saturation calculi automatically prove theorems.

They start from a set of formulas and repeatedly

- derive new formulas,
- remove redundant ones,

until a contradiction is found or saturation is reached.

Formulas of Superposition

Terms: x or $\mathsf{f}(t_1, \ldots, t_n)$

Formulas of Superposition

Terms: x or $\mathsf{f}(t_1, \ldots, t_n)$

Atoms: $t \approx t'$

Formulas of Superposition

Terms: x or $\mathsf{f}(t_1, \ldots, t_n)$

Atoms: $t \approx t'$

Literals: $t \approx t'$ or $t \not\approx t'$

Formulas of Superposition

Terms: x or $\mathsf{f}(t_1, \ldots, t_n)$

Atoms: $t \approx t'$

Literals: $t \approx t'$ or $t \not\approx t'$

Clauses: $l_1 \vee \cdots \vee l_n$

Superposition

Superposition is a saturation calculus by Bachmair and Ganzinger (1994).

Superposition

Superposition is a saturation calculus by Bachmair and Ganzinger (1994).

The calculus consists of three inference rules.

Superposition

Superposition is a saturation calculus by Bachmair and Ganzinger (1994).

The calculus consists of three inference rules.

Superposition corresponds to ordered resolution extended with equality.

Superposition

Superposition is a saturation calculus by Bachmair and Ganzinger (1994).

The calculus consists of three inference rules.

Superposition corresponds to ordered resolution extended with equality.

E, SPASS, Vampire, and Zipperposition implement superposition.

Example: Inverse of π

Assume $\pi \neq 0$.
Assume for every x ($\neq 0$) that the inverse of x is 1/x.
Then the absolute value of the inverse of π is the absolute value of $1/\pi$.

Example: Inverse of π

Assume $\pi \neq 0$.
Assume for every x ($\neq$ 0) that the inverse of x is 1/x.
Then the absolute value of the inverse of π is the absolute value of $1/\pi$.

$$\downarrow$$

$$\mathsf{pi} \not\approx \mathsf{zero} \ \wedge\ (\forall x.\ x \not\approx \mathsf{zero} \Rightarrow \mathsf{inv}(x) \approx \mathsf{div}(\mathsf{one}, x))$$

$$\Rightarrow \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \approx \mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi}))$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \;\wedge\; (\forall x.\; x \not\approx \mathsf{zero} \Rightarrow \mathsf{inv}(x) \approx \mathsf{div}(\mathsf{one}, x))$$

$$\Rightarrow \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \approx \mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi}))$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \;\wedge\; (\forall x.\; x \not\approx \mathsf{zero} \Rightarrow \mathsf{inv}(x) \approx \mathsf{div}(\mathsf{one}, x))$$

$$\Rightarrow \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \approx \mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi}))$$

↓

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \vee \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero} \lor \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero} \lor \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero} \lor \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero}$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero} \lor \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero}$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero} \lor \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero}$$

$$\mathsf{zero} \not\approx \mathsf{zero}$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero} \lor \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero}$$

$$\mathsf{zero} \not\approx \mathsf{zero}$$

Example: Inverse of π

$$\mathsf{pi} \not\approx \mathsf{zero} \qquad x \approx \mathsf{zero} \lor \mathsf{div}(\mathsf{one}, x) \approx \mathsf{inv}(x)$$

$$\mathsf{abs}(\mathsf{div}(\mathsf{one}, \mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero} \lor \mathsf{abs}(\mathsf{inv}(\mathsf{pi})) \not\approx \mathsf{abs}(\mathsf{inv}(\mathsf{pi}))$$

$$\mathsf{pi} \approx \mathsf{zero}$$

$$\mathsf{zero} \not\approx \mathsf{zero}$$

$$\bot$$

Formalization

We formalized untyped superposition in Isabelle (Desharnais et al. 2024).

Formalization

We formalized untyped superposition in Isabelle (Desharnais et al. 2024).

We formalized soundness and completeness using the saturation framework (Waldmann et al. 2022; Tourret and Blanchette 2021).

Untyped Superposition Locale

locale superposition_calculus =

Untyped Superposition Locale

locale superposition_calculus =

$$\left.\begin{array}{l}\text{nonground\_order}\ \prec_t\ + \\ \text{nonground\_selection\_function select}\ +\ \ldots\end{array}\right\}\ \text{assumptions}$$

Untyped Superposition Locale

locale superposition_calculus =

$$\left.\begin{array}{l}\text{nonground\_order } \prec_t \; + \\ \text{nonground\_selection\_function select } + \; \dots\end{array}\right\} \text{assumptions}$$

for

$\prec_t :: {}'t \Rightarrow {}'t \Rightarrow \mathit{bool}$ **and**

select :: $'t\ \mathit{clause} \Rightarrow {}'t\ \mathit{clause}$ **and** ...

Untyped Superposition Locale

```
locale superposition_calculus =
  nonground_order ≺ₜ +                          ⎫
                                                ⎬ assumptions
  nonground_selection_function select + ...     ⎭
  for
    ≺ₜ :: 't ⇒ 't ⇒ bool and
    select :: 't clause ⇒ 't clause and ...
begin

inductive superposition :: 't clause ⇒ 't clause ⇒ 't clause ⇒ bool where ...

inductive eq_resolution :: 't clause ⇒ 't clause ⇒ bool where ...

inductive eq_factoring :: 't clause ⇒ 't clause ⇒ bool where ...

end
```

Adding Types

Why Do We Want Types?

$$x \approx y \qquad \neg \mathsf{p}$$

The clause set is satisfiable in any interpretation with a single-element domain.

Why Do We Want Types?

$$x \approx y \qquad \mathsf{p} \not\approx \mathsf{t}$$

The constant t encodes truth.

Why Do We Want Types?

$$x \approx y \qquad \mathsf{p} \not\approx \mathsf{t}$$

The constant t encodes truth.

The clause set is unsatisfiable.

Why Do We Want Types?

$$x \approx y \qquad \mathsf{p} \not\approx \mathsf{t}$$

The symbols p and t have the Boolean type.

Why Do We Want Types?

$$x \approx y \qquad \mathsf{p} \not\approx \mathsf{t}$$

The symbols p and t have the Boolean type.

The clause set is satisfiable if the types of x and y are not Boolean.

Why Do We Want Types?

$$x \approx y \qquad \mathsf{p} \not\approx \mathsf{t}$$

The symbols p and t have the Boolean type.

The clause set is satisfiable if the types of x and y are not Boolean.

Modern provers support types natively.

Simple Monomorphic Types – Example

$$\frac{}{\mathcal{F}, \mathcal{V} \vdash \mathsf{replicate}\ x\ \mathsf{five} : \mathit{String}}$$

Simple Monomorphic Types – Example

$$\frac{\mathcal{F}\ \textsf{replicate}\ 2 = ([\mathit{Char}, \mathit{Nat}],\ \mathit{String})}{\mathcal{F}, \mathcal{V} \vdash \textsf{replicate}\ x\ \textsf{five} : \mathit{String}}$$

Simple Monomorphic Types – Example

$$\frac{\mathcal{F}\ \text{replicate}\ 2 = ([\mathit{Char}, \mathit{Nat}], \mathit{String}) \qquad \mathcal{V}\ \mathit{x} = \mathit{Char}}{\mathcal{F}, \mathcal{V} \vdash \text{replicate}\ \mathit{x}\ \text{five} : \mathit{String}}$$

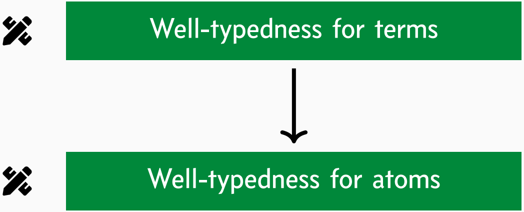
Simple Monomorphic Types – Example

$$\frac{\mathcal{F}\ \mathsf{replicate}\ 2 = ([\mathit{Char}, \mathit{Nat}],\ \mathit{String}) \qquad \mathcal{V}\ x = \mathit{Char} \qquad \mathcal{F}\ \mathsf{five}\ 0 = ([], \mathit{Nat})}{\mathcal{F}, \mathcal{V} \vdash \mathsf{replicate}\ x\ \mathsf{five} : \mathit{String}}$$

Liftings

Well-typedness for terms

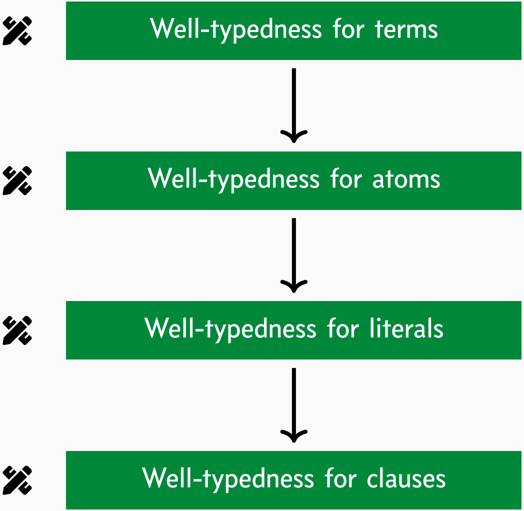
Liftings



Well-typedness for terms

Well-typedness for atoms

Liftings



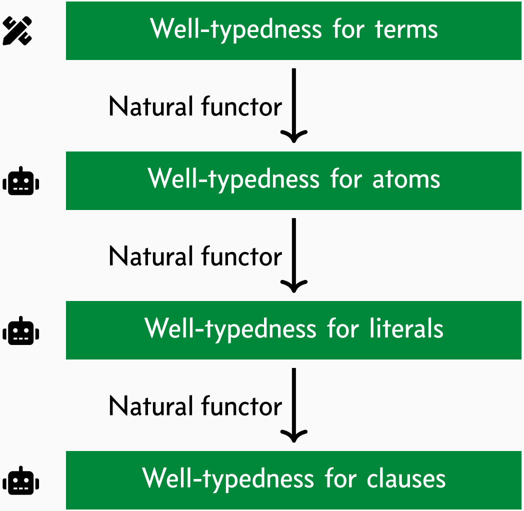
Well-typedness for terms

Well-typedness for atoms

Well-typedness for literals

Well-typedness for clauses

Liftings



✎ Well-typedness for terms

↓ Natural functor

🤖 Well-typedness for atoms

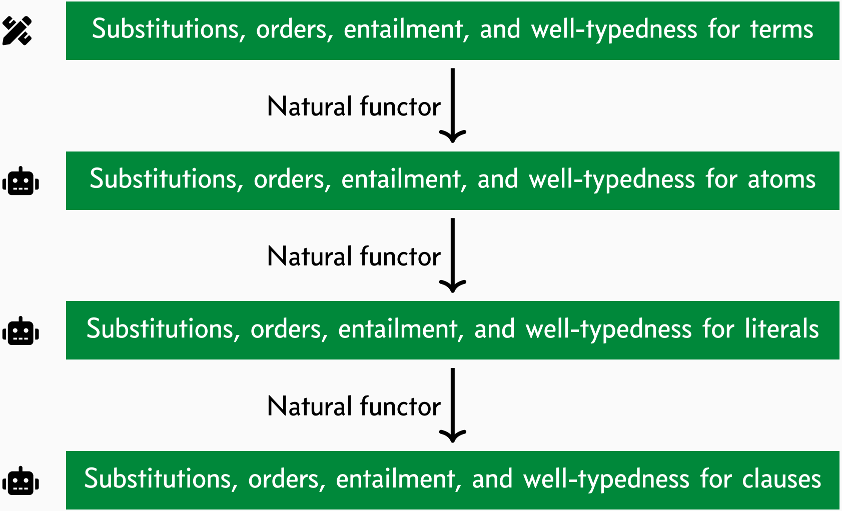
↓ Natural functor

🤖 Well-typedness for literals

↓ Natural functor

🤖 Well-typedness for clauses

Liftings



✎ Substitutions, orders, entailment, and well-typedness for terms

↓ Natural functor

🤖 Substitutions, orders, entailment, and well-typedness for atoms

↓ Natural functor

🤖 Substitutions, orders, entailment, and well-typedness for literals

↓ Natural functor

🤖 Substitutions, orders, entailment, and well-typedness for clauses

Typed Superposition Locale

locale superposition_calculus $=$
nonground_order $\prec_t$ +
nonground_selection_function select +
type_system welltyped + ...

Typed Superposition Locale

```
locale superposition_calculus =
  nonground_order ≺_t +
  nonground_selection_function select +
  type_system welltyped + ...
  for
    ≺_t :: 't ⇒ 't ⇒ bool and
    select :: 't clause ⇒ 't clause and
    welltyped :: ('v ⇒ 'ty) ⇒ 't ⇒ 'ty ⇒ bool and ...
```

Typed Superposition Locale

```
locale superposition_calculus =
  nonground_order ≺t +
  nonground_selection_function select +
  type_system welltyped + ...
  for
    ≺t :: 't ⇒ 't ⇒ bool and
    select :: 't clause ⇒ 't clause and
    welltyped :: ('v ⇒ 'ty) ⇒ 't ⇒ 'ty ⇒ bool and ...
begin
  ...

inductive eq_resolution :: ('t, 'v, 'ty) typed_clause ⇒ ('t, 'v, 'ty) typed_clause ⇒ bool where
  ...

end
```

Completeness Proof

Theorem

For every set N that is saturated, if N entails $\bot$, then $\bot \in N$.

Completeness Proof

Theorem

For every set N that is saturated, if N entails $\bot$, then $\bot \in N$.

N

Completeness Proof

Theorem

For every set N that is saturated, if N entails $\bot$, then $\bot \in N$.

$$\begin{tikzcd}
N \arrow[d, "\text{ground clauses}"'] \\
N_G
\end{tikzcd}$$

Completeness Proof

Theorem

For every set N that is saturated, if N entails $\bot$*, then* $\bot \in N$.

$$\begin{tikzcd}
N \arrow[d, "\text{ground clauses}"'] & \\
N_G \arrow[r, "\text{construct model}"'] & M_G
\end{tikzcd}$$

Completeness Proof

Theorem

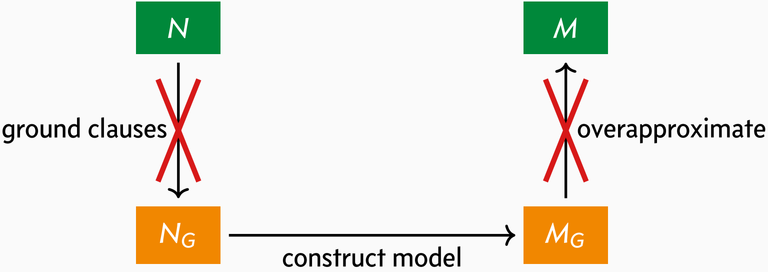
For every set N that is saturated, if N entails $\bot$, then $\bot \in N$.

$$\begin{tikzcd}
N \arrow[d, "\text{ground clauses}"'] & & M \\
N_G \arrow[rr, "\text{construct model}"'] & & M_G \arrow[u, "\text{overapproximate}"']
\end{tikzcd}$$

Completeness Proof

Goal

For every set N that is saturated, if N entails ⊥, then ⊥ ∈ N.



N

M

ground clauses

overapproximate

N_G

M_G

construct model

We add types to the nonground level.

Completeness Proof

Goal

For every set N that is saturated, if N entails $\bot$, then $\bot \in N$.

$$\begin{tikzcd}
N \arrow[d, "\text{ground welltyped clauses } \checkmark"'] & M \\
N_G \arrow[r, "\text{construct model}"'] & M_G \arrow[u, "\times" marking, "\text{overapproximate}"']
\end{tikzcd}$$

We add types to the nonground level.

Adapting the Inference Rules

1. The unifiers in the inference rules are type preserving.

Adapting the Inference Rules

1. The unifiers in the inference rules are type preserving.
2. The renaming substitutions in the superposition rule are type preserving.

Adapting the Inference Rules

1. The unifiers in the inference rules are type preserving.
2. The renaming substitutions in the superposition rule are type preserving.
3. For each type, sufficiently many variables exist.

Adapting the Inference Rules

1. The unifiers in the inference rules are type preserving.
2. The renaming substitutions in the superposition rule are type preserving.
3. For each type, sufficiently many variables exist.
4. The variable-type environments in the superposition rule are compatible.

Adapting the Inference Rules

1. The unifiers in the inference rules are type preserving.
2. The renaming substitutions in the superposition rule are type preserving.
3. For each type, sufficiently many variables exist.
4. The variable-type environments in the superposition rule are compatible.
5. One additional side condition for the superposition rule.

Well-typedness Preservation

Theorem

Let C, D, and E be clauses and $\mathcal{V}_1$*,* $\mathcal{V}_2$*, and* $\mathcal{V}_3$ *be variable-type environments. If there exists an inference* superposition $\langle \mathcal{V}_1, D\rangle\ \langle \mathcal{V}_2, E\rangle\ \langle \mathcal{V}_3, C\rangle$*, then*

Well-typedness Preservation

Theorem

Let C, D, and E be clauses and $\mathcal{V}_1$, $\mathcal{V}_2$, and $\mathcal{V}_3$ be variable-type environments. If there exists an inference superposition $\langle \mathcal{V}_1, D \rangle$ $\langle \mathcal{V}_2, E \rangle$ $\langle \mathcal{V}_3, C \rangle$*, then*

$$(\text{clause.is\_welltyped}\ \mathcal{F}\ \mathcal{V}_1\ D \wedge \text{clause.is\_welltyped}\ \mathcal{F}\ \mathcal{V}_2\ E) \longrightarrow \text{clause.is\_welltyped}\ \mathcal{F}\ \mathcal{V}_3\ C$$

Completeness Proof with Types

Theorem

For every set N that is saturated, if N entails $\bot$, then $\bot \in N$.

$$\begin{tikzcd}
N \arrow[d, swap, "\text{ground welltyped clauses } \checkmark"] & & M \\
N_G \arrow[rr, swap, "\text{construct model}"] & & M_G \arrow[u, swap, "\text{overapproximate } \checkmark"]
\end{tikzcd}$$

Why Are Modular Proofs Useful?

Ahmed and I (2025) derived a formalization of typed ordered resolution.

Why Are Modular Proofs Useful?

Ahmed and I (2025) derived a formalization of typed ordered resolution.

The formalization is compatible with Yamada and Thiemann (2024) by adding only about 150 lines of proof text.

Why Are Modular Proofs Useful?

Ahmed and I (2025) derived a formalization of typed ordered resolution.

The formalization is compatible with Yamada and Thiemann (2024) by adding only about 150 lines of proof text.

We are extending the approach to rank-1 polymorphism.

Conclusion

We parameterized our formalization of superposition with a monomorphic type system.

Conclusion

We parameterized our formalization of superposition with a monomorphic type system.

We work on rank-1 polymorphism.

Conclusion

We parameterized our formalization of superposition with a monomorphic type system.

We work on rank-1 polymorphism.

Our goal is to obtain a verified executable superposition prover.