Iterative Monomorphisation

Jasmin Blanchette and Tanguy Bozec

LMU, München, Germany and ENS Paris-Saclay, Gif-sur-Yvette, France

Rank-1 Polymorphism

$$\forall x : \texttt{list\_int}, f\langle\texttt{list\_int}\rangle(x)$$

Rank-1 Polymorphism

$$\forall x : \texttt{list\_int}, f\langle\texttt{list\_int}\rangle(x)$$
$$\forall x : \texttt{list\_nat}, f\langle\texttt{list\_nat}\rangle(x)$$
$$\forall x : \texttt{list\_bool}, f\langle\texttt{list\_bool}\rangle(x)$$
$$\forall x : \texttt{list\_string}, f\langle\texttt{list\_string}\rangle(x)$$

Rank-1 Polymorphism

$$\forall x : \texttt{list\_int}, f\langle\texttt{list\_int}\rangle(x)$$
$$\forall x : \texttt{list\_nat}, f\langle\texttt{list\_nat}\rangle(x)$$
$$\forall x : \texttt{list\_bool}, f\langle\texttt{list\_bool}\rangle(x)$$
$$\forall x : \texttt{list\_string}, f\langle\texttt{list\_string}\rangle(x)$$

$$\forall {\color{red}\alpha}, \forall x : \texttt{list}({\color{red}\alpha}), f\langle\texttt{list}({\color{red}\alpha})\rangle(x)$$

Rank-1 Polymorphism

$$\forall x : \texttt{list\_int}, f\langle\texttt{list\_int}\rangle(x)$$
$$\forall x : \texttt{list\_nat}, f\langle\texttt{list\_nat}\rangle(x)$$
$$\forall x : \texttt{list\_bool}, f\langle\texttt{list\_bool}\rangle(x)$$
$$\forall x : \texttt{list\_string}, f\langle\texttt{list\_string}\rangle(x)$$

$$\forall \alpha, \forall x : \texttt{list}(\alpha), f\langle\texttt{list}(\alpha)\rangle(x)$$

Type variables are quantified **universally** at the **top level** of a formula.

Motivation

$$\begin{tikzcd}
 & & \text{Zipperposition} \\
 & & \text{Vampire} \\
\text{Isabelle/HOL} \arrow[r] & \text{Sledgehammer} \arrow[uur] \arrow[ur] \arrow[dr] \arrow[ddr] & \\
 & & \text{E} \\
 & & \text{Leo-III}
\end{tikzcd}$$

Motivation

Polymorphic
with type variables

Isabelle/HOL $\longrightarrow$ Sledgehammer

Sledgehammer $\rightarrow$ Zipperposition

Sledgehammer $\rightarrow$ Vampire

Sledgehammer $\rightarrow$ E

Sledgehammer $\rightarrow$ Leo-III

Motivation

Polymorphic
with type variables

Isabelle/HOL ⟶ Sledgehammer

Sledgehammer → Zipperposition

Sledgehammer → Vampire

Sledgehammer → E

Sledgehammer → Leo-III

Motivation

Polymorphic
with type variables

Monomorphic
no type variables

Isabelle/HOL → Sledgehammer → Zipperposition
Sledgehammer → Vampire
Sledgehammer → E
Sledgehammer → Leo-III

Motivation

Polymorphic
with type variables

Monomorphic
no type variables

Isabelle/HOL ⟶ Sledgehammer

Sledgehammer ⟶ Zipperposition ⟶ E

Sledgehammer ⟶ Vampire

Sledgehammer ⟶ E

Sledgehammer ⟶ Leo-III

Motivation

Polymorphic
with type variables

Monomorphic
no type variables

$$
\begin{tikzcd}
 & & \text{Zipperposition} \arrow[r] & \text{E} \\
 & & \text{Vampire} & \\
\text{Isabelle/HOL} \arrow[r] & \text{Sledgehammer} \arrow[ur] \arrow[uur] \arrow[r] \arrow[dr] & \text{E} & \\
 & & \text{Leo-III} &
\end{tikzcd}
$$

Motivation

Polymorphic
with type variables

Monomorphic
no type variables

Isabelle/HOL → Sledgehammer

Sledgehammer → Zipperposition → E ❌

Sledgehammer → Vampire

Sledgehammer → E ❌

Sledgehammer → Leo-III

Solution

Polymorphic problem $\Longrightarrow$ Monomorphic problem

Solution

$$\text{Polymorphic problem} \Longrightarrow \text{Monomorphic problem}$$

Two possibilities:

1. **Encode** type variables in a monomorphic logic.
2. **Instantiate** type variables.

Algorithm

P is the set of input formulae
while new formulae are added to P **do**
  for all $\varphi \in P$ **do**
    for all occurrences $f\langle\pi\rangle(...)$ in φ with π polymorphic **do**
      for all occurrences $f\langle\tau\rangle(...)$ in P with τ monomorphic **do**
        if π matches against τ **then**
          add σ, the unifier of π and τ to S
    for all $\sigma \in S$ **do**
      add $\varphi\sigma$ to P
return $\{\varphi \in P \mid \varphi \text{ is monomorphic}\}$

Algorithm

Soundness: instantiation of universally quantified type variables.

Completeness: this algorithm is incomplete.

Algorithm

Soundness: instantiation of universally quantified type variables.

Completeness: this algorithm is incomplete.

- Finding a **finite equisatisfiable** set of monomorphic instances of a first-order polymorphic formula is **undecidable**.

Algorithm

Soundness: instantiation of universally quantified type variables.

Completeness: this algorithm is incomplete.

- Finding a **finite equisatisfiable** set of monomorphic instances of a first-order polymorphic formula is **undecidable**.
- **Bounds** limit the instantiations we perform.

Example

Initial problem:

1. $\forall x : \texttt{int}, f\langle\texttt{int}\rangle(x)$
2. $\forall x : \alpha, y : \texttt{list}(\alpha), f\langle\alpha\rangle(x) \wedge f\langle\texttt{list}(\alpha)\rangle(y)$

Example

Initial problem:

1. $\forall x : \mathtt{int}, f\langle \mathtt{int} \rangle(x)$
2. $\forall x : \alpha, y : \mathtt{list}(\alpha), f\langle \alpha \rangle(x) \wedge f\langle \mathtt{list}(\alpha) \rangle(y)$

Successful match of α against $\mathtt{int}$.

Example

Initial problem:

1. $\forall x : \texttt{int}, f\langle\texttt{int}\rangle(x)$
2. $\forall x : \alpha, y : \texttt{list}(\alpha), f\langle\alpha\rangle(x) \land f\langle\texttt{list}(\alpha)\rangle(y)$

Successful match of α against $\texttt{int}$.

Failure to match $\texttt{list}(\alpha)$ against $\texttt{int}$.

Example

Initial problem:

1. $\forall x : \mathtt{int}, f\langle \mathtt{int} \rangle(x)$
2. $\forall x : \alpha, y : \mathtt{list}(\alpha), f\langle \alpha \rangle(x) \land f\langle \mathtt{list}(\alpha) \rangle(y)$

Successful match of α against $\mathtt{int}$.

We apply the substitution $\alpha \mapsto \mathtt{int}$ to clause 2.

3. $\forall x : \mathtt{int}, y : \mathtt{list}(\mathtt{int}), f\langle \mathtt{int} \rangle(x) \land f\langle \mathtt{list}(\mathtt{int}) \rangle(y)$

Example

1. $\forall x : \texttt{int}, f\langle\texttt{int}\rangle(x)$
2. $\forall x : \alpha, y : \texttt{list}(\alpha), f\langle\alpha\rangle(x) \wedge f\langle\texttt{list}(\alpha)\rangle(y)$
3. $\forall x : \texttt{int}, y : \texttt{list}(\texttt{int}), f\langle\texttt{int}\rangle(x) \wedge f\langle\texttt{list}(\texttt{int})\rangle(y)$

Example

1. $\forall x : \texttt{int}, f\langle\texttt{int}\rangle(x)$
2. $\forall x : \alpha, y : \texttt{list}(\alpha), f\langle\alpha\rangle(x) \wedge f\langle\texttt{list}(\alpha)\rangle(y)$
3. $\forall x : \texttt{int}, y : \texttt{list}(\texttt{int}), f\langle\texttt{int}\rangle(x) \wedge f\langle\texttt{list}(\texttt{int})\rangle(y)$

Successful match of α against $\texttt{list}(\texttt{int})$.

Example

1. $\forall x : \texttt{int}, f\langle\texttt{int}\rangle(x)$
2. $\forall x : \alpha, y : \texttt{list}(\alpha), f\langle\alpha\rangle(x) \wedge f\langle\texttt{list}(\alpha)\rangle(y)$
3. $\forall x : \texttt{int}, y : \texttt{list}(\texttt{int}), f\langle\texttt{int}\rangle(x) \wedge f\langle\texttt{list}(\texttt{int})\rangle(y)$

Successful match of α against $\texttt{list}(\texttt{int})$.

We apply the substitution $\alpha \mapsto \texttt{list}(\texttt{int})$ to clause 2.

4. $\forall x : \texttt{list}(\texttt{int}), y : \texttt{list}(\texttt{list}(\texttt{int})),$
   $f\langle\texttt{list}(\texttt{int})\rangle(x) \wedge f\langle\texttt{list}(\texttt{list}(\texttt{int}))\rangle(y)$

Example

1. $\forall x : \mathtt{int}, f\langle\mathtt{int}\rangle(x)$
2. $\forall x : \alpha, y : \mathtt{list}(\alpha), f\langle\alpha\rangle(x) \wedge f\langle\mathtt{list}(\alpha)\rangle(y)$
3. $\forall x : \mathtt{int}, y : \mathtt{list}(\mathtt{int}), f\langle\mathtt{int}\rangle(x) \wedge f\langle\mathtt{list}(\mathtt{int})\rangle(y)$

Successful match of α against $\mathtt{list}(\mathtt{int})$.

We apply the substitution $\alpha \mapsto \mathtt{list}(\mathtt{int})$ to clause 2.

4. $\forall x : \mathtt{list}(\mathtt{int}), y : \mathtt{list}(\mathtt{list}(\mathtt{int})),$
   $f\langle\mathtt{list}(\mathtt{int})\rangle(x) \wedge f\langle\mathtt{list}(\mathtt{list}(\mathtt{int}))\rangle(y)$

This can generate an **infinite** number of new formulae.

Bounds

Since we cannot **exhaustively enumerate** all type variables instantiations, we use **heuristics** to determine which instantiations we perform:

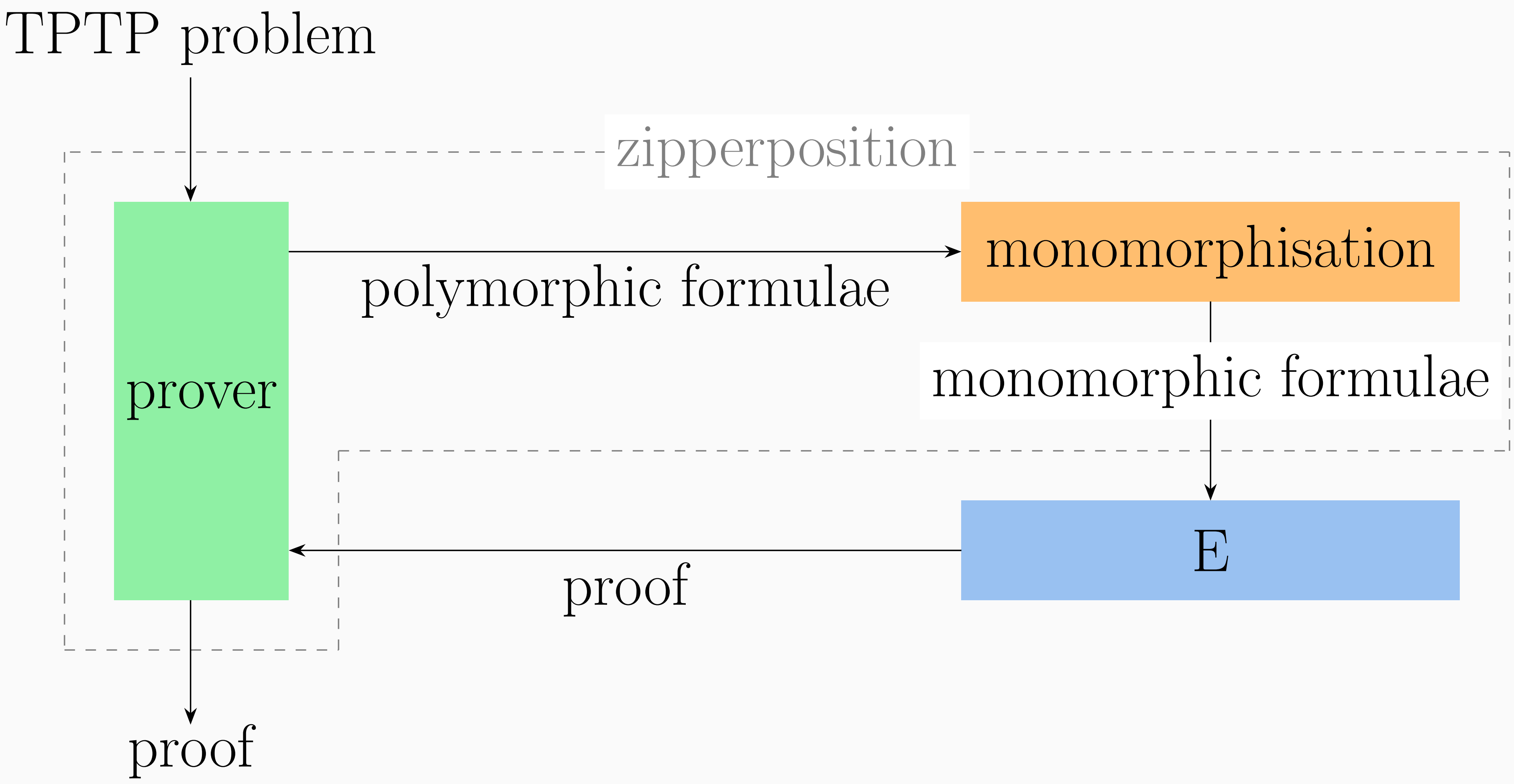
- We limit the number of **iterations**.
- We filter type arguments by **function symbol**.

Bounds

Since we cannot **exhaustively enumerate** all type variables instantiations, we use **heuristics** to determine which instantiations we perform:

- We limit the number of **iterations**.
- We filter type arguments by **function symbol**.
- We limit the number of **substitutions** we generate.
- We limit the number of **applications** of the substitutions.

Zipperposition and E



Benchmarks

We wanted to test the **usefulness** of our implementation of iterative monomorphisation.

We had two questions:

1. Does Zipperposition benefit from the ability to call E on monomorphised problems?
2. Does E perform well on monomorphised problems?

Benchmarks

Polymorphic
with type variables

Monomorphic
no type variables

Isabelle/HOL → Sledgehammer

Sledgehammer → Zipperposition → E ❓

Sledgehammer → Vampire

Sledgehammer → E ❓

Sledgehammer → Leo-III

Methodology

We used the **TPTP** (Thousand Problems for Theorem Provers) problem set for our evaluations.

Methodology

We used the **TPTP** (Thousand Problems for Theorem Provers) problem set for our evaluations.

We split the problem set into two:

- 500 problems for adjusting bounds and parameters
- 1034 problems for the benchmarks

Zipperposition benefits from monomorphisation

	Zipperposition without E	Zipperposition with E	Union
500 problems	168	198	207
1034 problems	337	410	434

Zipperposition benefits from monomorphisation

	Zipperposition without E	Zipperposition with E	Union
500 problems	168	198	207
1034 problems	337	410	434

This is **expected** Zipperposition benefits greatly from E in a monomorphic setting.

Zipperposition benefits from monomorphisation

Polymorphic — with type variables

Monomorphic — no type variables

Isabelle/HOL ⟶ Sledgehammer

Sledgehammer ⟶ Zipperposition ⟶ E ✅

Sledgehammer ⟶ Vampire

Sledgehammer ⟶ E ❓

Sledgehammer ⟶ Leo-III

E performs well on monomorphised problems

	Polymorphic	Monomorphised
E	-	340
Zipperposition	339	

Calling E on monomorphised problems is a **viable** option.

E performs well on monomorphised problems

	Polymorphic	Monomorphised
E	-	340
Zipperposition	339	351

Calling E on monomorphised problems is a **viable** option.

E performs well on monomorphised problems

	Polymorphic	Monomorphised
E	-	340
Zipperposition	339	351
Leo-III	157	231

Calling E on monomorphised problems is a **viable** option.

Polymorphic provers perform **better** on monomorphised problems.

E performs well on monomorphised problems

	Polymorphic	Monomorphised	Union
E	-	340	340
Zipperposition	339	351	404
Leo-III	157	231	274

Calling E on monomorphised problems is a **viable** option.

Polymorphic provers perform **better** on monomorphised problems.

E performs well on monomorphised problems

Polymorphic
with type variables

Monomorphic
no type variables

Isabelle/HOL → Sledgehammer

Sledgehammer → Zipperposition → E ✅

Sledgehammer → Vampire

Sledgehammer → E ✅

Sledgehammer → Leo-III

Conclusion

- Goal: Polymorphic problem $\Longrightarrow$ Monomorphic problem.
- Solution: **instantiating** type variables with ground types.

Conclusion

- Goal: Polymorphic problem $\Longrightarrow$ Monomorphic problem.
- Solution: **instantiating** type variables with ground types.
- **Bounds** are necessary in practice.

Conclusion

- Goal: Polymorphic problem $\Longrightarrow$ Monomorphic problem.
- Solution: **instantiating** type variables with ground types.
- **Bounds** are necessary in practice.
- It is a **viable** means for extending monomorphic provers.
- Iterative monomorphisation can **outperform** native implementations of polymorphism.

Conclusion

Polymorphic
with type variables

Monomorphic
no type variables

Isabelle/HOL ⟶ Sledgehammer

Sledgehammer ⟶ Zipperposition ⟶ E ✅

Sledgehammer ⟶ Vampire

Sledgehammer ⟶ E ✅

Sledgehammer ⟶ Leo-III

Iterative Monomorphisation

Jasmin Blanchette and Tanguy Bozec

LMU, München, Germany and ENS Paris-Saclay, Gif-sur-Yvette, France