Karlsruher Institut für Technologie

Translating Lean to Higher Order Logic

Praxis der Forschung

Max Nowak | May 15, 2023

```
theorem toArrayLit_eq' (a : Array α) (n : Nat) (hsz : a.size = n) : a = toArrayLit a n hsz := by
  have := aux n
  rw [List.drop_eq_nil_of_le (Nat.le_of_eq hsz)] at this
  exact (data_toArray a).symm.trans $ congrArg List.toArray (this _).symm
where
  aux : ∀ i hi, toListLitAux a n hsz i hi (a.data.drop i) = a.data
  | 0, _ ⇒ rfl
  | i+1, hi ⇒ by
    simp [toListLitAux]
    suffices _::_ = _ by rw [this]; apply aux
    apply List.get_cons_drop
```

Introduction

- Interactive Theorem Provers
  - Based on higher order logic: Isabelle/HOL
  - Based on Martin-Löf type theory and the calculus of inductive constructions: Coq, Lean 4

Introduction

- Interactive Theorem Provers
  - Based on higher order logic: Isabelle/HOL
  - Based on Martin-Löf type theory and the calculus of inductive constructions: Coq, Lean 4

```
theorem sumBounded (xs : Vec ℕ len) (f: ℕ → ℕ) (m : Monotone f)
  : sum (map f xs) ≤ len * (f (max xs))
  := by
    induction xs with
    | @nil ⇒ simp [Nat.zero_eq, Nat.zero_mul, Nat.le_zero_eq, map]
    | @cons len x xs ih ⇒
      rw [Vec.max]
      by_cases x ≤ max xs
      . simp_all [ite_true, map, sum]
        have : f x ≤ f (max xs) := by apply m; simp_all
        linarith
      . simp [ite_false, map, sum, h]
        suffices len * f (max xs) ≤ len * f x by linarith
        have : f (max xs) ≤ f x := by apply m; simp at h; exact le_of_lt h
        exact mul_le_mul_left₂ (f (Vec.max xs)) (f x) this len
```

Introduction

- Interactive Theorem Provers
  - Based on higher order logic: Isabelle/HOL
  - Based on Martin-Löf type theory and the calculus of inductive constructions: Coq, Lean 4

```
theorem sumBounded (xs : Vec ℕ len) (f: ℕ → ℕ) (m : Monotone f)
  : sum (map f xs) ≤ len * (f (max xs))
  := by
    induction xs with
    | @nil ⇒ simp [Nat.zero_eq, Nat.zero_mul, Nat.le_zero_eq, map]
    | @cons len x xs ih ⇒
      rw [Vec.max]
      by_cases x ≤ max xs
      . simp_all [ite_true, map, sum]
        have : f x ≤ f (max xs) := by apply m; simp_all
        linarith
      . simp [ite_false, map, sum, h]
        suffices len * f (max xs) ≤ len * f x by linarith
        have : f (max xs) ≤ f x := by apply m; simp at h; exact le_of_lt h
        exact mul_le_mul_left₂ (f (Vec.max xs)) (f x) this len
```

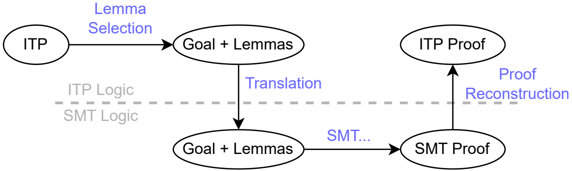
- SMT solvers now good enough at solving HOL problems
- Hammers
  - Isabelle/HOL: Sledgehammer (HOL)
  - Coq: CoqHammer and SMTCoq (FOL)
  - Lean: Lean-Smt (FOL)

Introduction

- Interactive Theorem Provers
  - Based on higher order logic: Isabelle/HOL
  - Based on Martin-Löf type theory and the calculus of inductive constructions: Coq, Lean 4

```
theorem sumBounded (xs : Vec ℕ len) (f: ℕ → ℕ) (m : Monotone f)
  : sum (map f xs) ≤ len * (f (max xs))
  := by
    induction xs with
    | @nil ⇒ simp [Nat.zero_eq, Nat.zero_mul, Nat.le_zero_eq, map]
    | @cons len x xs ih ⇒
      rw [Vec.max]
      by_cases x ≤ max xs
      . simp_all [ite_true, map, sum]
        have : f x ≤ f (max xs) := by apply m; simp_all
        linarith
      . simp [ite_false, map, sum, h]
        suffices len * f (max xs) ≤ len * f x by linarith
        have : f (max xs) ≤ f x := by apply m; simp at h; exact le_of_lt h
        exact mul_le_mul_left₂ (f (Vec.max xs)) (f x) this len
```

- SMT solvers now good enough at solving HOL problems
- Hammers
  - Isabelle/HOL: Sledgehammer (HOL)
  - Coq: CoqHammer and SMTCoq (FOL)
  - Lean: Lean-Smt (FOL)



Lean Metaprogramming

- Translate Lean goals $\Gamma \vdash ?\mathrm{m} : \phi$ to new Lean goals
  - Can infer types, distinguish between x : Type and x : Prop
  - Add translated inductive types and definitions

Lean Metaprogramming

- Translate Lean goals $\Gamma \vdash ?\mathrm{m} : \phi$ to new Lean goals
  - Can infer types, distinguish between x : Type and x : Prop
  - Add translated inductive types and definitions
- `Expr` represents values, types, propositions, and proofs
  - $a = b$ represented as `Expr.app (Expr.app (Expr.const "Eq") a) b`

Lean Metaprogramming

- Translate Lean goals $\Gamma \vdash ?\mathrm{m} : \phi$ to new Lean goals
  - Can infer types, distinguish between x : Type and x : Prop
  - Add translated inductive types and definitions
- `Expr` represents values, types, propositions, and proofs
  - $a = b$ represented as `Expr.app (Expr.app (Expr.const "Eq") a) b`

$$\vdash \qquad \forall_{x:\mathrm{List}\ A}\ x = x$$

Lean Metaprogramming

- Translate Lean goals $\Gamma \vdash ?m : \phi$ to new Lean goals
  - Can infer types, distinguish between x : Type and x : Prop
  - Add translated inductive types and definitions
- `Expr` represents values, types, propositions, and proofs
  - $a = b$ represented as `Expr.app (Expr.app (Expr.const "Eq") a) b`

$$\vdash \qquad \forall_{x:\text{List } A} \; x = x$$

$$\vdash \qquad \forall_{x:\text{List'}} \; x = x$$

Lean Metaprogramming

- Translate Lean goals $\Gamma \vdash ?\mathrm{m} : \phi$ to new Lean goals
  - Can infer types, distinguish between x : Type and x : Prop
  - Add translated inductive types and definitions
- `Expr` represents values, types, propositions, and proofs
  - $a = b$ represented as `Expr.app (Expr.app (Expr.const "Eq") a) b`

$$\vdash \qquad \forall_{x:\mathrm{List}\ A}\ x = x$$

$$\vdash \mathrm{prf}_1 : \forall_{x:\mathrm{List'}}\ x = x$$

Lean Metaprogramming

- Translate Lean goals $\Gamma \vdash ?\mathrm{m} : \phi$ to new Lean goals
  - Can infer types, distinguish between x : Type and x : Prop
  - Add translated inductive types and definitions
- `Expr` represents values, types, propositions, and proofs
  - $a = b$ represented as `Expr.app (Expr.app (Expr.const "Eq") a) b`

$$\vdash \ \mathrm{prf}_0 : \forall_{x:\mathrm{List}\ A} \ \ x = x$$

$$\vdash \ \mathrm{prf}_1 : \forall_{x:\mathrm{List'}} \ \ x = x$$

Lean Metaprogramming

- Translate Lean goals $\Gamma \vdash ?\mathrm{m} : \phi$ to new Lean goals
  - Can infer types, distinguish between x : Type and x : Prop
  - Add translated inductive types and definitions
- `Expr` represents values, types, propositions, and proofs
  - $a = b$ represented as `Expr.app (Expr.app (Expr.const "Eq") a) b`

$$\vdash \ \mathrm{prf}_0 : \forall_{x:\mathrm{List}\ A} \ x = x$$

$$\vdash \ \mathrm{prf}_1 : \forall_{x:\mathrm{List'}} \ x = x$$

- Finally, directly encode to TPTP-THF0
  - Simply-typed higher order logic

Lean vs HOL

Lean	HOL
And ϕ ψ	$\phi \wedge \psi$
$\phi \to \psi$	$\phi \Rightarrow \psi$

Lean vs HOL

Lean	HOL
And ϕ ψ	$\phi \wedge \psi$
$\phi \to \psi$	$\phi \Rightarrow \psi$
$\mathbb{N}$	$\mathbb{N}$

inductive $\mathbb{N}$: Type

zero : $\mathbb{N}$

cons : $\mathbb{N} \to \mathbb{N}$

Lean vs HOL

Lean	HOL
And ϕ ψ	$\phi \wedge \psi$
$\phi \to \psi$	$\phi \Rightarrow \psi$
$\mathbb{N}$	$\mathbb{N}$
Even n	even n

inductive $\mathbb{N}$: Type

zero : $\mathbb{N}$

cons : $\mathbb{N} \to \mathbb{N}$

inductive Even : $\mathbb{N} \to$ Prop

base : Even 0

step : $(x : \mathbb{N}) \to$ Even $x \to$ Even $(x + 2)$

Lean vs HOL

Lean	HOL
And ϕ ψ	$\phi \wedge \psi$
$\phi \rightarrow \psi$	$\phi \Rightarrow \psi$
$\mathbb{N}$	$\mathbb{N}$
Even n	even n
$(x : A) \rightarrow \phi\ x$	$\forall_{x:A}\ \phi\ x$

inductive $\mathbb{N}$: Type

zero : $\mathbb{N}$

cons : $\mathbb{N} \rightarrow \mathbb{N}$

inductive Even : $\mathbb{N} \rightarrow$ Prop

base : Even 0

step : $(x : \mathbb{N}) \rightarrow \text{Even } x \rightarrow \text{Even } (x + 2)$

Lean vs HOL

Lean	HOL
$\text{And}\ \phi\ \psi$	$\phi \land \psi$
$\phi \rightarrow \psi$	$\phi \Rightarrow \psi$
$\mathbb{N}$	$\mathbb{N}$
$\text{Even}\ n$	$\text{even}\ n$
$(x : A) \rightarrow \phi\ x$	$\forall_{x:A}\ \phi\ x$
$A \rightarrow B$	$A \rightarrow B$
$\lambda_{x:A}\ f\ x$	$\lambda_{x:A}\ f\ x$

inductive $\mathbb{N}$: Type

zero : $\mathbb{N}$

cons : $\mathbb{N} \rightarrow \mathbb{N}$

inductive Even : $\mathbb{N} \rightarrow$ Prop

base : Even 0

step : $(x : \mathbb{N}) \rightarrow \text{Even}\ x \rightarrow \text{Even}\ (x + 2)$

Lean vs HOL

Lean	HOL
And $\phi\ \psi$	$\phi \wedge \psi$
$\phi \to \psi$	$\phi \Rightarrow \psi$
$\mathbb{N}$	$\mathbb{N}$
Even n	even n
$(x : A) \to \phi\ x$	$\forall_{x:A}\ \phi\ x$
$A \to B$	$A \to B$
$\lambda_{x:A}\ f\ x$	$\lambda_{x:A}\ f\ x$
$(x : A) \to B\ x$	no

inductive $\mathbb{N}$: Type

zero : $\mathbb{N}$

cons : $\mathbb{N} \to \mathbb{N}$

inductive Even : $\mathbb{N} \to$ Prop

base : Even 0

step : $(x : \mathbb{N}) \to$ Even $x \to$ Even $(x + 2)$

Lean vs HOL

Lean	HOL
And ϕ ψ	$\phi \wedge \psi$
$\phi \to \psi$	$\phi \Rightarrow \psi$
$\mathbb{N}$	$\mathbb{N}$
Even n	even n
$(x : A) \to \phi\ x$	$\forall_{x:A}\ \phi\ x$
$A \to B$	$A \to B$
$\lambda_{x:A}\ f\ x$	$\lambda_{x:A}\ f\ x$
$(x : A) \to B\ x$	no
List A	no

inductive $\mathbb{N}$: Type

zero : $\mathbb{N}$

cons : $\mathbb{N} \to \mathbb{N}$

inductive Even : $\mathbb{N} \to$ Prop

base : Even 0

step : $(x : \mathbb{N}) \to$ Even $x \to$ Even $(x + 2)$

inductive List : $(A :$ Type$) \to$ Type

nil : List A

cons : $A \to$ List $A \to$ List A

Lean vs HOL

Lean	HOL
And ϕ ψ	$\phi \wedge \psi$
$\phi \to \psi$	$\phi \Rightarrow \psi$
$\mathbb{N}$	$\mathbb{N}$
Even n	even n
$(x : A) \to \phi\ x$	$\forall_{x:A}\ \phi\ x$
$A \to B$	$A \to B$
$\lambda_{x:A}\ f\ x$	$\lambda_{x:A}\ f\ x$
$(x : A) \to B\ x$	no
List A	no

- Need non-simple types to write $(x : A) \to B\ x$

inductive $\mathbb{N}$: Type

zero : $\mathbb{N}$

cons : $\mathbb{N} \to \mathbb{N}$

inductive Even : $\mathbb{N} \to$ Prop

base : Even 0

step : $(x : \mathbb{N}) \to$ Even $x \to$ Even $(x + 2)$

inductive List : $(A : \text{Type}) \to$ Type

nil : List A

cons : $A \to$ List $A \to$ List A

Anatomy of Inductive Types

$$\textbf{inductive}\ \mathrm{Vec} : (A : \mathrm{Type}) \to (\mathit{len} : \mathbb{N}) \to \mathrm{Type}$$

$$\mathrm{nil} : \mathrm{Vec}\ A\ 0$$

$$\mathrm{cons} : (\mathit{len} : \mathbb{N}) \to A \to \mathrm{Vec}\ A\ \mathit{len} \to \mathrm{Vec}\ A\ (\mathit{len} + 1)$$

- Kinds of parameters
  - A is a uniform type parameter
  - *len* is a type index

Anatomy of Inductive Types

Karlsruher Institut für Technologie

$$\textbf{inductive}\ \mathrm{Vec} : (A : \mathrm{Type}) \to (len : \mathbb{N}) \to \mathrm{Type}$$

$$\mathrm{nil} : \mathrm{Vec}\ A\ 0$$

$$\mathrm{cons} : (len : \mathbb{N}) \to A \to \mathrm{Vec}\ A\ len \to \mathrm{Vec}\ A\ (len + 1)$$

	value	type
uniform	*VU*	*TU*
index	*VX*	*TX*

- Kinds of parameters
  - A is a uniform type parameter (TU)
  - *len* is a type index (VX)

Anatomy of Inductive Types

$$\textbf{inductive}\ \mathrm{Vec} : (A : \mathrm{Type}) \to (len : \mathbb{N}) \to \mathrm{Type}$$

$$\quad \mathrm{nil} : \mathrm{Vec}\ A\ 0$$

$$\quad \mathrm{cons} : (len : \mathbb{N}) \to A \to \mathrm{Vec}\ A\ len \to \mathrm{Vec}\ A\ (len + 1)$$

	value	type
uniform	*VU*	*TU*
index	*VX*	*TX*

- Kinds of parameters
  - A is a uniform type parameter (TU)
  - len is a type index (VX)
- Want only simple data types $\mathrm{T} : \mathrm{Type}$
- Two approaches
  - Monomorphization
  - Guard construction

Guard Construction

inductive Vec : $(A : \text{Type}) \to \mathbb{N} \to \text{Type}$

nil : Vec A 0

cons : $(len : \mathbb{N}) \to (x : A) \to$

$(xs : \text{Vec } A\ len) \to$

Vec $A\ (len + 1)$

double : Vec A n $\to$ Vec A $(2 * n)$

Guard Construction

$$\textbf{inductive}\ \mathrm{Vec} : (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{Type}$$
$$\quad \mathrm{nil} : \mathrm{Vec}\ A\ 0$$
$$\quad \mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to$$
$$\qquad (xs : \mathrm{Vec}\ A\ len) \to$$
$$\qquad \mathrm{Vec}\ A\ (len + 1)$$

$$\textbf{inductive}\ \mathrm{VecE} : (A : \mathrm{Type}) \to \mathrm{Type}$$
$$\quad \mathrm{nil} : \mathrm{VecE}\ A$$
$$\quad \mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to$$
$$\qquad (xs : \mathrm{VecE}\ A) \to$$
$$\qquad \mathrm{VecE}\ A$$

- Erase indices

$$\mathrm{double} : \mathrm{VecE}\ A \to \mathrm{VecE}\ A$$

Guard Construction

$\textbf{inductive } \text{Vec} : (A : \text{Type}) \to \mathbb{N} \to \text{Type}$

$\quad \text{nil} : \text{Vec } A \; 0$

$\quad \text{cons} : (\mathit{len} : \mathbb{N}) \to (x : A) \to$

$\qquad (\mathit{xs} : \text{Vec } A \; \mathit{len}) \to$

$\qquad \text{Vec } A \; (\mathit{len} + 1)$

$\textbf{inductive } \text{VecE} : (A : \text{Type}) \to \text{Type}$

$\quad \text{nil} : \text{VecE } A$

$\quad \text{cons} : (\mathit{len} : \mathbb{N}) \to (x : A) \to$

$\qquad (\mathit{xs} : \text{VecE } A) \to$

$\qquad \text{VecE } A$

- Erase indices
- Derive guard predicate

$\textbf{inductive } \text{VecG} : (A : \text{Type}) \to \mathbb{N} \to \text{VecE } A \to \text{Prop}$

$\text{double} : \text{VecE } A \to \text{VecE } A$

Guard Construction

$$\textbf{inductive}\ \mathrm{Vec} : (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{Vec}\ A\ 0$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \mathrm{Vec}\ A\ len) \to \mathrm{Vec}\ A\ (len + 1)$$

$$\textbf{inductive}\ \mathrm{VecE} : (A : \mathrm{Type}) \to \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{VecE}\ A$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \mathrm{VecE}\ A) \to \mathrm{VecE}\ A$$

- Erase indices
- Derive guard predicate

$$\textbf{inductive}\ \mathrm{VecG} : (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{VecE}\ A \to \mathrm{Prop}$$
$$\mathrm{nil} : \mathrm{VecG}\ A\ 0\ (\mathrm{VecE.nil}\ A)$$

$$\mathrm{double} : \mathrm{VecE}\ A \to \mathrm{VecE}\ A$$

Guard Construction

$$\textbf{inductive}\ \mathrm{Vec} : (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{Vec}\ A\ 0$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \mathrm{Vec}\ A\ len) \to \mathrm{Vec}\ A\ (len + 1)$$

$$\textbf{inductive}\ \mathrm{VecE} : (A : \mathrm{Type}) \to \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{VecE}\ A$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \mathrm{VecE}\ A) \to \mathrm{VecE}\ A$$

- Erase indices
- Derive guard predicate

$$\textbf{inductive}\ \mathrm{VecG} : (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{VecE}\ A \to \mathrm{Prop}$$
$$\mathrm{nil} : \mathrm{VecG}\ A\ 0\ (\mathrm{VecE.nil}\ A)$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \mathrm{Vec}\ A\ len) \to \mathrm{VecG}\ A\ (len + 1)\ (\mathrm{VecE.cons}\ A\ \_\ x\ xs)$$

$$\mathrm{double} : \mathrm{VecE}\ A \to \mathrm{VecE}\ A$$

Guard Construction

$$\textbf{inductive } \mathrm{Vec} : (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{Vec}\ A\ 0$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \mathrm{Vec}\ A\ len) \to \mathrm{Vec}\ A\ (len + 1)$$

$$\textbf{inductive } \mathrm{VecE} : (A : \mathrm{Type}) \to \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{VecE}\ A$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \mathrm{VecE}\ A) \to \mathrm{VecE}\ A$$

- Erase indices
- Derive guard predicate
- Replace $T\ u\ i$ with $\{x : E\ u \mid G\ u\ i\ x\}$
  - $\{x : \mathbb{N} \mid x < 5\}$ is a *subtype* of $\mathbb{N}$

$$\textbf{inductive } \mathrm{VecG} : (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{VecE}\ A \to \mathrm{Prop}$$
$$\mathrm{nil} : \mathrm{VecG}\ A\ 0\ (\mathrm{VecE.nil}\ A)$$
$$\mathrm{cons} : (len : \mathbb{N}) \to (x : A) \to (xs : \{v : \mathrm{VecE}\ A \mid \mathrm{VecG}\ A\ len\ v\}) \to \mathrm{VecG}\ A\ (len + 1)\ (\mathrm{VecE.cons}\ A\ \_\ x\ xs)$$

$$\mathrm{double} : \{x : \mathrm{VecE}\ A \mid \mathrm{VecG}\ A\ n\ x\} \to \{x : \mathrm{VecE}\ A \mid \mathrm{VecG}\ A\ (2 * n)\ x\}$$

Monomorphization

- Inductive data types: Monomorphize TU (and VU) parameters
  - Predicates: Only monomorphize TU parameters

$$
\begin{aligned}
T &: (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{Type} \\
T' &: \mathrm{Type} \\
\text{“}T' &:= T\ \mathbb{N}\ 3\text{”}
\end{aligned}
$$

Monomorphization

- Inductive data types: Monomorphize TU (and VU) parameters
  - Predicates: Only monomorphize TU parameters

$$
\begin{aligned}
T &: (A : \mathrm{Type}) \to \mathbb{N} \to \mathrm{Type} \\
T' &: \mathrm{Type} \\
``T' &:= T\ \mathbb{N}\ 3\ "
\end{aligned}
$$

- Functions: Monomorphize parameters occuring in *data* positions

$$
\begin{aligned}
f &: (A : \mathrm{Type}) \to (n : \mathbb{N}) \to (k : \mathbb{N}) \to k < n \to \mathrm{Vec}\ A\ n \\
f' &: (k : \mathbb{N}) \to k < 7 \to \mathrm{Vec}\ \mathbb{N}\ 7 \\
f' &:= f\ \mathbb{N}\ 7
\end{aligned}
$$

Monomorphization

- Inductive data types: Monomorphize TU (and VU) parameters
  - Predicates: Only monomorphize TU parameters

$$T : (A : \text{Type}) \to \mathbb{N} \to \text{Type}$$
$$T' : \text{Type}$$
$$\text{“}T' := T\ \mathbb{N}\ 3\text{”}$$

- Functions: Monomorphize parameters occuring in *data* positions

$$f : (A : \text{Type}) \to (n : \mathbb{N}) \to (k : \mathbb{N}) \to k < n \to \text{Vec}\ A\ n$$
$$f' : (k : \mathbb{N}) \to k < 7 \to \text{Vec}\ \mathbb{N}\ 7$$
$$f' := f\ \mathbb{N}\ 7$$

- What about goals such as $(A : \text{Type}) \to \ldots$?
  - Lift binder by introducing **axiom** A : Type

Encoding Inductive Types to TPTP

- Inductive data types
  - Add injectivity theorems
  - Structures get projection theorems

inductive $\text{List}_{\mathbb{Z}}$: Type

nil : $\text{List}_{\mathbb{Z}}$

cons : $\mathbb{Z} \to \text{List}_{\mathbb{Z}} \to \text{List}_{\mathbb{Z}}$

```
thf(type, ListZ : $tType).
thf(type, nil : ListZ).
thf(type, cons : Z → ListZ → ListZ).
```

Encoding Inductive Types to TPTP

- Inductive data types
  - Add injectivity theorems
  - Structures get projection theorems

$$\textbf{inductive } \mathrm{List}_{\mathbb{Z}} : \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{List}_{\mathbb{Z}}$$
$$\mathrm{cons} : \mathbb{Z} \to \mathrm{List}_{\mathbb{Z}} \to \mathrm{List}_{\mathbb{Z}}$$

```
thf(type, ListZ : $tType).
thf(type, nil : ListZ).
thf(type, cons : Z → ListZ → ListZ).
```

- Inductive predicates: Treat Prop as Bool

$$\textbf{inductive } \mathrm{VecG} : \mathbb{N} \to \mathrm{VecE} \to \mathrm{Prop}$$

```
thf(type, VecG : N → VecE → Bool).
```

Encoding Inductive Types to TPTP

Karlsruher Institut für Technologie

- Inductive data types
  - Add injectivity theorems
  - Structures get projection theorems

$$\textbf{inductive } \mathrm{List}_{\mathbb{Z}} : \mathrm{Type}$$
$$\mathrm{nil} : \mathrm{List}_{\mathbb{Z}}$$
$$\mathrm{cons} : \mathbb{Z} \to \mathrm{List}_{\mathbb{Z}} \to \mathrm{List}_{\mathbb{Z}}$$

```
thf(type, ListZ : $tType).
thf(type, nil : ListZ).
thf(type, cons : Z → ListZ → ListZ).
```

- Inductive predicates: Treat Prop as Bool

$$\textbf{inductive } \mathrm{VecG} : \mathbb{N} \to \mathrm{VecE} \to \mathrm{Prop}$$
$$\mathrm{nil} : \mathrm{VecG}\ 0\ \mathrm{VecE.nil}$$
$$\mathrm{cons} : (\mathit{len} : \mathbb{N}) \to (x : \mathbb{Z}) \to$$
$$(\mathit{xs} : \{v : \mathrm{VecE} \mid \mathrm{VecG}\ \mathit{len}\ v\}) \to$$
$$\mathrm{VecG}\ (\mathit{len} + 1)\ (\mathrm{VecE.cons}\ \_\ x\ \mathit{xs})$$

```
thf(type, VecG : N → VecE → Bool).
thf(axiom, VecG 0 VecE.nil).
thf(axiom, ∀ len : N, ∀ x : Z,
  ∀ xs : VecE, VecG len xs ⇒
  VecG (len + 1) (VecE.cons _ x xs)).
```

Encoding Functions to TPTP

- Best case: Directly via lambda

def increment : $\mathbb{N} \to \mathbb{N}$

$:= \lambda_n \; n + 1$

`thf(type, increment : N` $\to$ `N).`

`thf(axiom, increment` $=$ `(`λ`x : N, add x 1)).`

Encoding Functions to TPTP

- Best case: Directly via lambda

def increment : $\mathbb{N} \to \mathbb{N}$

$:= \lambda_n \; n + 1$

```
thf(type, increment : N → N).
thf(axiom, increment = (λx : N, add x 1)).
```

- Functions with pattern matching: Multiple equations

def map : $(A \to B) \to \text{List}_A \to \text{List}_B$

$\mid f, \; []_A \Rightarrow []_B$

$\mid f, \; x ::_A xs \; \Rightarrow (f \; x) ::_B (\text{map } xs)$

```
thf(type, map : (A → B) → ListA → ListB).
thf(axiom, ∀f, map f nilA = nilB).
thf(axiom, ∀f ∀x ∀xs,
  map f (consA x xs) = consB (f x) (map f xs)).
```

Encoding Functions to TPTP

- Best case: Directly via lambda

def increment : $\mathbb{N} \to \mathbb{N}$

$:= \lambda_n \;\; n + 1$

```
thf(type, increment : N → N).
thf(axiom, increment = (λx : N, add x 1)).
```

- Functions with pattern matching: Multiple equations

def map : $(A \to B) \to \text{List}_A \to \text{List}_B$

$\mid f, \; [\,]_A \Rightarrow [\,]_B$

$\mid f, \; x ::_A xs \;\Rightarrow (f\; x) ::_B (\text{map}\; xs)$

```
thf(type, map : (A → B) → ListA → ListB).
thf(axiom, ∀f, map f nilA = nilB).
thf(axiom, ∀f ∀x ∀xs,
  map f (consA x xs) = consB (f x) (map f xs)).
```

- Treat $\{x : T \mid P\; x\}$ as T

Summary and Observations

Karlsruher Institut für Technologie

```
theorem sumBounded (xs : Vec ℕ len) (f: ℕ → ℕ) (m : Monotone f)
  : sum (map f xs) ≤ len * (f (max xs))
  := by
    induction xs with
    | @nil ⇒ smt
    | @cons len x xs ih ⇒ smt
```

- Map inductive types to simple types
  - Guard construction
  - Monomorphization
- Translation is extensible
- Ability to reason about higher order functions
- Type indices depending on other type indices seem to work
- Typeclasses and their structure projections are spammy
  - Proof for $1 + 2 = 3$ is 250kB
  - CVC5 on $1 + 0 + 2 = 3$ times out
- Further work
  - Lemma selection and proof reconstruction
  - Recursors
  - Pre- and postconditions