Verification of programs manipulating ADTs using Shallow Horn Clauses

Théo Losekoot, Thomas Genet, Thomas Jensen

IRISA, France

GT DAAL
2024, April 25

What is the problem?

Objective

Proving relational properties on functional programs manipulating algebraic data types in a completely automatic way.

What is the problem?

Objective

Proving relational properties on functional programs manipulating algebraic data types in a completely automatic way.

Example: Program

```
type nat = Z | S of nat
type nlist = Nil | Cons of nat * nlist
```

```
let rec length l =
  match l with
  | Nil -> Z
  | Cons(_,t) ->
     S(length t)
```

```
let rec less n1 n2 =
  match (n1, n2) with
  | Z, S(_) -> true
  | _, Z -> false
  | S(n1'), S(n2') ->
     less n1' n2'
```

What is the problem?

Objective

Proving relational properties on functional programs manipulating algebraic data types in a completely automatic way.

Example: Program and properties

```
type nat = Z | S of nat
type nlist = Nil | Cons of nat * nlist
```

```
let rec length l =
  match l with
  | Nil -> Z
  | Cons(_,t) ->
     S(length t)
```

```
let rec less n1 n2 =
  match (n1, n2) with
  | Z, S(_) -> true
  | _, Z -> false
  | S(n1'), S(n2') ->
     less n1' n2'
```

$P_1 \doteq \forall l, x,\ less\ Z\ (length\ (Cons(x,\ l)))$
$P_2 \doteq \forall l, x,\ less\ (length\ l)\ (length\ (Cons(x,\ l)))$

Algebraic non-relational properties: [1]

Property P_1

$\forall l, x,\ less\ Z\ (length\ (Cons(x,\ l)))$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_1

$\forall l, x,\ less\ Z\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_1

$\forall l, x,\ less\ Z\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_1

$\forall l, x,\ less\ Z\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$
$less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ \mathcal{L}_{Cons^+})$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_1

$\forall l, x,\ less\ Z\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$
$less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ \mathcal{L}_{Cons^+})$
$less^\sharp\ \mathcal{L}_Z\ \mathcal{L}_{S^+}$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_1

$\forall l, x,\ less\ Z\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n, \mathcal{L}_l)))$
$less^\sharp\ \mathcal{L}_Z\ (length^\sharp\ \mathcal{L}_{Cons^+})$
$less^\sharp\ \mathcal{L}_Z\ \mathcal{L}_{S^+}$
true

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_2

$\forall l, x,\ less\ (length\ l)\ (length\ (Cons(x,\ l)))$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_2

$\forall l, x,\ less\ (length\ l)\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_2

$\forall l, x,\ less\ (length\ l)\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_2

$\forall l, x,\ less\ (length\ l)\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$
$less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ \mathcal{L}_{Cons^+})$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_2

$\forall l, x, \; less \; (length \; l) \; (length \; (Cons(x, \; l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp \; (length^\sharp \; \mathcal{L}_l) \; (length^\sharp \; (Cons^\sharp(\mathcal{L}_n, \; \mathcal{L}_l)))$

Abstract computation

$less^\sharp \; (length^\sharp \; \mathcal{L}_l) \; (length^\sharp \; (Cons^\sharp(\mathcal{L}_n, \; \mathcal{L}_l)))$
$less^\sharp \; (length^\sharp \; \mathcal{L}_l) \; (length^\sharp \; \mathcal{L}_{Cons^+})$
$less^\sharp \; \mathcal{L}_n \; \mathcal{L}_{S^+}$

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_2

$\forall l, x,\ less\ (length\ l)\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$
$less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ \mathcal{L}_{Cons^+})$
$less^\sharp\ \mathcal{L}_n\ \mathcal{L}_{S^+}$
?

[1] Haudebourg, Genet, Jensen, "Regular Language Type Inference with Term Rewriting"

Algebraic non-relational properties: [1]

Property P_2

$\forall l, x,\ less\ (length\ l)\ (length\ (Cons(x,\ l)))$

Type abstraction by regular languages (using tree automaton)

- Natural numbers: $\mathcal{L}_n \doteq \mathcal{L}_Z \cup \mathcal{L}_{S^+}$
- Lists: $\mathcal{L}_l \doteq \mathcal{L}_{Nil} \cup \mathcal{L}_{Cons^+}$

Abstract property: $less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$

Abstract computation

$less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ (Cons^\sharp(\mathcal{L}_n,\ \mathcal{L}_l)))$
$less^\sharp\ (length^\sharp\ \mathcal{L}_l)\ (length^\sharp\ \mathcal{L}_{Cons^+})$
$less^\sharp\ \mathcal{L}_n\ \mathcal{L}_{S^+}$
? $\leftarrow$ Cannot be proved using non-relational abstraction

[1] Haudebourg, Genet, Jensen, “Regular Language Type Inference with Term Rewriting”

Making relations explicit: from functions to clauses

From $len(l) = n$ to $\mathrm{Len}(l, n)$.

Program as clauses: Len and Less

$\mathrm{Len}(Nil,\ Z).$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n)).$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2.$

$\mathrm{Less}(Z,\ S(m)).$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}.$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m)).$

Making relations explicit: from functions to clauses

From $len(l) = n$ to $\mathrm{Len}(l, n)$.

Program as clauses: Len and Less

$\mathrm{Len}(Nil,\ Z).$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n)).$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2.$

$\mathrm{Less}(Z,\ S(m)).$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}.$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m)).$

Unique Herbrand model $\mathcal{M}$ of the above-defined clauses

$\mathcal{M}(\mathrm{Len}) = \{(Nil, Z), (Cons(Z, Nil), S(Z)), \ldots\}$
$\mathcal{M}(\mathrm{Less}) = \{(Z, S(Z)), (Z, S(S(Z))), (S(Z), S(S(Z))), \ldots\}$

Making relations explicit: from functions to clauses

From $len(l) = n$ to $\mathrm{Len}(l, n)$.

Program as clauses: Len and Less

$\mathrm{Len}(Nil,\ Z).$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n)).$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2.$

$\mathrm{Less}(Z,\ S(m)).$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}.$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m)).$

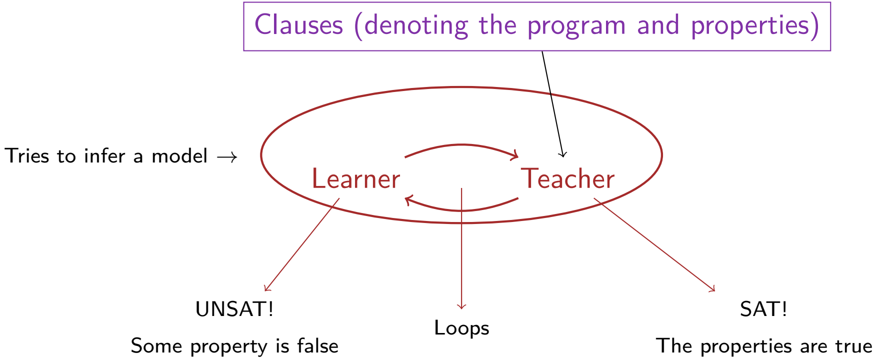
Unique Herbrand model $\mathcal{M}$ of the above-defined clauses

$\mathcal{M}(\mathrm{Len}) = \{(Nil, Z), (Cons(Z, Nil), S(Z)), \ldots\}$
$\mathcal{M}(\mathrm{Less}) = \{(Z, S(Z)), (Z, S(S(Z))), (S(Z), S(S(Z))), \ldots\}$

Property P_2 as clauses

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n').$

Satisfiability procedure: our method at a glance



Model inference: Learner | Teacher

Guesses a model which generalizes ground clauses received from the teacher.

- Proposes more and more refined models
- Stops with UNSAT if the ground clauses are inconsistent

Checks whether a given model meets the specification (clauses).

- Gives a ground counterexample to every clause that the model does not satisfy
- Stops with SAT if the proposed model satisfies every constraint

Learner	Teacher

$\mathrm{Len}(\mathit{Nil},\ Z)$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(\mathit{Cons}(x, l),\ S(n))$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z,\ S(m))$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(\mathit{Cons}(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

$\text{Len} \mapsto \emptyset$
$\text{Less} \mapsto \emptyset$

$\text{Len}(\mathit{Nil},\ Z)$
$\text{Len}(l,\ n) \Rightarrow \text{Len}(\mathit{Cons}(x, l),\ S(n))$
$\text{Len}(l,\ n_1) \wedge \text{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\text{Less}(Z,\ S(m))$
$\text{Less}(n,\ Z) \Rightarrow \text{False}$
$\text{Less}(n,\ m) \Leftrightarrow \text{Less}(S(n),\ S(m))$

$\text{Len}(l,\ n) \wedge \text{Len}(\mathit{Cons}(x, l),\ n') \Rightarrow \text{Less}(n,\ n')$

Learner | Teacher

$\text{Len} \mapsto \emptyset$
$\text{Less} \mapsto \emptyset$

$\times\, \text{Len}(Nil,\ Z)$
$\text{Len}(l,\ n) \Rightarrow \text{Len}(Cons(x, l),\ S(n))$
$\text{Len}(l,\ n_1) \wedge \text{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\times\, \text{Less}(Z,\ S(m))$
$\text{Less}(n,\ Z) \Rightarrow \text{False}$
$\text{Less}(n,\ m) \Leftrightarrow \text{Less}(S(n),\ S(m))$

$\text{Len}(l,\ n) \wedge \text{Len}(Cons(x, l),\ n') \Rightarrow \text{Less}(n,\ n')$

Learner | Teacher

$\mathrm{Len} \mapsto \emptyset$
$\mathrm{Less} \mapsto \emptyset$

$\times\, \mathrm{Len}(Nil,\ Z)$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n))$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\times\, \mathrm{Less}(Z,\ S(m))$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

$\mathrm{Len}(Nil, Z)$
$\mathrm{Less}(Z, S(Z))$

Learner | Teacher

Learner

$\mathrm{Len}(Nil, Z)$

$\mathrm{Less}(Z, S(Z))$

Teacher

$\mathrm{Len}(Nil,\ Z)$

$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n))$

$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z,\ S(m))$

$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$

$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

Learner | Teacher

$\text{Len} \mapsto \{(Nil, Z)\}$
$\text{Less} \mapsto \{(Z, S(x)) \mid x \in Nat\}$

Learner:

$\text{Len}(Nil, Z)$

$\text{Less}(Z, S(Z))$

Teacher:

$\text{Len}(Nil,\ Z)$
$\text{Len}(l,\ n) \Rightarrow \text{Len}(Cons(x, l),\ S(n))$
$\text{Len}(l,\ n_1) \wedge \text{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\text{Less}(Z,\ S(m))$
$\text{Less}(n,\ Z) \Rightarrow \text{False}$
$\text{Less}(n,\ m) \Leftrightarrow \text{Less}(S(n),\ S(m))$

$\text{Len}(l,\ n) \wedge \text{Len}(Cons(x, l),\ n') \Rightarrow \text{Less}(n,\ n')$

$\text{Len} \mapsto \{(Nil, Z)\}$
$\text{Less} \mapsto \{(Z, S(\cdot))\}$

$\text{Len}(Nil, Z)$
$\text{Less}(Z, S(Z))$

$\text{Len}(Nil,\ Z)$
$\times\ \text{Len}(l,\ n) \Rightarrow \text{Len}(Cons(x, l),\ S(n))$
$\text{Len}(l,\ n_1) \wedge \text{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\text{Less}(Z,\ S(m))$
$\text{Less}(n,\ Z) \Rightarrow \text{False}$
$\times\ \text{Less}(n,\ m) \Leftrightarrow \text{Less}(S(n),\ S(m))$

$\text{Len}(l,\ n) \wedge \text{Len}(Cons(x, l),\ n') \Rightarrow \text{Less}(n,\ n')$

$\text{Len} \mapsto \{(Nil, Z)\}$
$\text{Less} \mapsto \{(Z, S(\cdot))\}$

$\text{Len}(Nil, Z)$
$\text{Less}(Z, S(Z))$

$\text{Len}(Nil,\ Z)$
x $\text{Len}(l,\ n) \Rightarrow \text{Len}(Cons(x, l),\ S(n))$
$\text{Len}(l,\ n_1) \wedge \text{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\text{Less}(Z,\ S(m))$
$\text{Less}(n,\ Z) \Rightarrow \text{False}$
x $\text{Less}(n,\ m) \Leftrightarrow \text{Less}(S(n),\ S(m))$

$\text{Len}(l,\ n) \wedge \text{Len}(Cons(x, l),\ n') \Rightarrow \text{Less}(n,\ n')$

$\text{Len}(Nil, Z) \Rightarrow \text{Len}(Cons(Z, Nil), S(Z))$
$\text{Less}(Z, S(Z)) \Rightarrow \text{Less}(S(Z), S(S(Z)))$

Learner	Teacher
$\mathrm{Len}(Nil, Z)$ $\mathrm{Less}(Z, S(Z))$ $\mathrm{Len}(Nil, Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil), S(Z))$ $\mathrm{Less}(Z, S(Z)) \Rightarrow \mathrm{Less}(S(Z), S(S(Z)))$	$\mathrm{Len}(Nil,\ Z)$ $\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n))$ $\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$ $\mathrm{Less}(Z,\ S(m))$ $\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$ $\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$ $\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

$\mathrm{Len} \mapsto \{(Nil, Z)\} \cup \{(Cons(\cdot,\cdot), S(\cdot))\}$
$\mathrm{Less} \mapsto \{(\cdot, S(\cdot))\}$

Learner

$\mathrm{Len}(Nil, Z)$

$\mathrm{Less}(Z, S(Z))$

$\mathrm{Len}(Nil, Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil), S(Z))$

$\mathrm{Less}(Z, S(Z)) \Rightarrow \mathrm{Less}(S(Z), S(S(Z)))$

Teacher

$\mathrm{Len}(Nil,\ Z)$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n))$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z,\ S(m))$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

$$\text{Len} \mapsto \{(Nil, Z)\} \cup \{(Cons(\cdot,\cdot), S(\cdot))\}$$
$$\text{Less} \mapsto \{(\cdot, S(\cdot))\}$$

Learner

$\text{Len}(Nil, Z)$

$\text{Less}(Z, S(Z))$

$\text{Len}(Nil, Z) \Rightarrow \text{Len}(Cons(Z, Nil), S(Z))$

$\text{Less}(Z, S(Z)) \Rightarrow \text{Less}(S(Z), S(S(Z)))$

Teacher

$\text{Len}(Nil, Z)$

$\text{Len}(l, n) \Rightarrow \text{Len}(Cons(x, l), S(n))$

x $\text{Len}(l, n_1) \wedge \text{Len}(l, n_2) \Rightarrow n_1 = n_2$

$\text{Less}(Z, S(m))$

$\text{Less}(n, Z) \Rightarrow \text{False}$

x $\text{Less}(n, m) \Leftrightarrow \text{Less}(S(n), S(m))$

$\text{Len}(l, n) \wedge \text{Len}(Cons(x, l), n') \Rightarrow \text{Less}(n, n')$

Learner | Teacher

$\text{Len} \mapsto \{(\mathit{Nil}, Z)\} \cup \{(\mathit{Cons}(\cdot,\cdot), S(\cdot))\}$
$\text{Less} \mapsto \{(\cdot, S(\cdot))\}$

Learner:

$\text{Len}(\mathit{Nil}, Z)$

$\text{Less}(Z, S(Z))$

$\text{Len}(\mathit{Nil}, Z) \Rightarrow \text{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$

$\text{Less}(Z, S(Z)) \Rightarrow \text{Less}(S(Z), S(S(Z)))$

Teacher:

$\text{Len}(\mathit{Nil},\ Z)$

$\text{Len}(l,\ n) \Rightarrow \text{Len}(\mathit{Cons}(x, l),\ S(n))$

$x\ \text{Len}(l,\ n_1) \wedge \text{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\text{Less}(Z,\ S(m))$

$\text{Less}(n,\ Z) \Rightarrow \text{False}$

$x\ \text{Less}(n,\ m) \Leftrightarrow \text{Less}(S(n),\ S(m))$

$\text{Len}(l,\ n) \wedge \text{Len}(\mathit{Cons}(x, l),\ n') \Rightarrow \text{Less}(n,\ n')$

$$\begin{pmatrix} \text{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \ \wedge \\ \text{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z))) \end{pmatrix} \Rightarrow S(Z)) = S(S(Z))$$

$$\text{Less}(S(Z), S(Z)) \quad \Rightarrow \quad \text{Less}(Z, Z)$$

Learner

$\mathrm{Len}(\mathit{Nil}, Z)$
$\mathrm{Less}(Z, S(Z))$
$\mathrm{Len}(\mathit{Nil}, Z) \Rightarrow \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$
$\mathrm{Less}(Z, S(Z)) \Rightarrow \mathrm{Less}(S(Z), S(S(Z)))$

$$\begin{pmatrix} \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \ \wedge \\ \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z))) \end{pmatrix} \implies \bot$$

$\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)$

Teacher

$\mathrm{Len}(\mathit{Nil},\ Z)$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(\mathit{Cons}(x, l),\ S(n))$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z,\ S(m))$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(\mathit{Cons}(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

Learner | Teacher

Details coming soon

$\mathrm{Len} \mapsto \{([x_1, \ldots, x_n], S^n(Z)) \mid n \in \mathbb{N}\}$
$\mathrm{Less} \mapsto \{(n_1, n_2) \mid n_1 < n_2\}$

Learner

$\mathrm{Len}(Nil, Z)$

$\mathrm{Less}(Z, S(Z))$

$\mathrm{Len}(Nil, Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil), S(Z))$

$\mathrm{Less}(Z, S(Z)) \Rightarrow \mathrm{Less}(S(Z), S(S(Z)))$

$$\begin{pmatrix} \mathrm{Len}(Cons(Z, Nil), S(Z)) \ \wedge \\ \mathrm{Len}(Cons(Z, Nil), S(S(Z))) \end{pmatrix} \implies \bot$$

$\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)$

Teacher

$\mathrm{Len}(Nil,\ Z)$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n))$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z,\ S(m))$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

Details coming soon

$\mathrm{Len} \mapsto \{([x_1, \ldots, x_n], S^n(Z)) \mid n \in \mathbb{N}\}$
$\mathrm{Less} \mapsto \{(n_1, n_2) \mid n_1 < n_2\}$

Learner:

$\mathrm{Len}(Nil, Z)$

$\mathrm{Less}(Z, S(Z))$

$\mathrm{Len}(Nil, Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil), S(Z))$

$\mathrm{Less}(Z, S(Z)) \Rightarrow \mathrm{Less}(S(Z), S(S(Z)))$

$$\begin{pmatrix} \mathrm{Len}(Cons(Z, Nil), S(Z)) \ \wedge \\ \mathrm{Len}(Cons(Z, Nil), S(S(Z))) \end{pmatrix} \implies \bot$$

$\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)$

Teacher:

$\mathrm{Len}(Nil,\ Z)$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n))$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z,\ S(m))$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

Details coming soon

$\mathrm{Len} \mapsto \{([x_1, \ldots, x_n], S^n(Z)) \mid n \in \mathbb{N}\}$
$\mathrm{Less} \mapsto \{(n_1, n_2) \mid n_1 < n_2\}$

$\mathrm{Len}(Nil, Z)$
$\mathrm{Less}(Z, S(Z))$
$\mathrm{Len}(Nil, Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil), S(Z))$
$\mathrm{Less}(Z, S(Z)) \Rightarrow \mathrm{Less}(S(Z), S(S(Z)))$
$$\begin{pmatrix} \mathrm{Len}(Cons(Z, Nil), S(Z)) \ \wedge \\ \mathrm{Len}(Cons(Z, Nil), S(S(Z))) \end{pmatrix} \implies \bot$$
$\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)$

$\mathrm{Len}(Nil, \ Z)$
$\mathrm{Len}(l, \ n) \Rightarrow \mathrm{Len}(Cons(x, l), \ S(n))$
$\mathrm{Len}(l, \ n_1) \wedge \mathrm{Len}(l, \ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z, \ S(m))$
$\mathrm{Less}(n, \ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n, \ m) \Leftrightarrow \mathrm{Less}(S(n), \ S(m))$

$\mathrm{Len}(l, \ n) \wedge \mathrm{Len}(Cons(x, l), \ n') \Rightarrow \mathrm{Less}(n, \ n')$

SAT!

Models: what representation?

Len $\mapsto \emptyset$
Less $\mapsto \emptyset$

Len $\mapsto \{(Nil, Z)\}$
Less $\mapsto \{(Z, S(\cdot))\}$

Len $\mapsto \{(Nil, Z)\} \cup \{(Cons(\cdot,\cdot), S(\cdot))\}$
Less $\mapsto \{(\cdot, S(\cdot))\}$

Len $\mapsto \{([x_1, \ldots, x_n], S^n(Z)) \mid n \in \mathbb{N}\}$
Less $\mapsto \{(n_1, n_2) \mid n_1 < n_2\}$

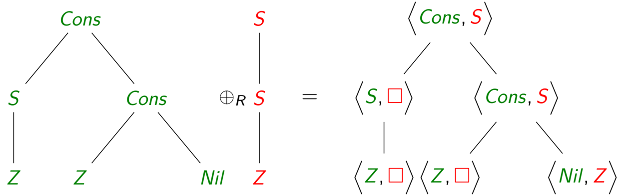
We need a finite representation of these models that

- Allows to *express* various relation between recursive algebraic datatypes;
- Can be *synthesized* from examples (Learner);
- Admits a procedure for *model-checking* constrained clauses (Teacher).

Convolution

Core idea

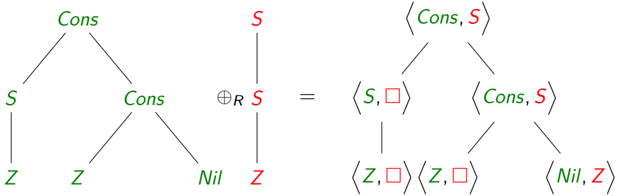
Overlay terms to read them at the same time using a classical tree automaton on a product alphabet



Convolution

Core idea

Overlay terms to read them at the same time using a classical tree automaton on a product alphabet



Convoluted tree automaton denoting the relation Len

Automaton $\mathcal{A}_{\mathrm{Len}}$ has final states $\{q_l\}$ and transitions

$$\left\{ \begin{array}{ll} \langle Nil, Z\rangle() \to q_l & \langle Z, \square\rangle() \to q_n \\ \langle Cons, S\rangle(q_n, q_l) \to q_l & \langle S, \square\rangle(q_n) \to q_n \end{array} \right\}$$

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $\mathit{Vars}(B) \subseteq \mathit{Vars}(H)$

A set of SHoCs is a model that inductively defines relations.

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $\mathit{Vars}(B) \subseteq \mathit{Vars}(H)$

A set of SHoCs is a model that inductively defines relations.

Length

$\mathrm{Len}(\mathit{Nil}, Z)$
$\mathrm{Len}(\mathit{Cons}(x, l),\ S(n)) \Leftarrow \mathrm{Len}(l,\ n)$

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $\mathit{Vars}(B) \subseteq \mathit{Vars}(H)$

A set of SHoCs is a model that inductively defines relations.

Height using Max

$$\mathrm{Height}(\mathit{Leaf}, Z)$$

$$\mathrm{Height}(\mathit{Node}(l, x, r),\ S(n)) \Leftarrow \mathrm{Height}(l, h_l) \land \mathrm{Height}(r, h_r) \land \mathrm{Max}(h_l, h_r, n)$$

$$\mathrm{Max}(\ldots) \Leftarrow \ldots$$

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $\mathit{Vars}(B) \subseteq \mathit{Vars}(H)$

A set of SHoCs is a model that inductively defines relations.

Height using Max: forbidden

$$\mathrm{Height}(\mathit{Leaf}, Z)$$

$$\mathrm{Height}(\mathit{Node}(l, x, r),\ S(n)) \Leftarrow \mathrm{Height}(l, h_l) \wedge \mathrm{Height}(r, h_r) \wedge \mathrm{Max}(h_l, h_r, n)$$

$$\mathrm{Max}(\ldots) \Leftarrow \ldots$$

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $\mathit{Vars}(B) \subseteq \mathit{Vars}(H)$

A set of SHoCs is a model that inductively defines relations.

Height using Shallower

$\mathrm{Height}(\mathit{Leaf}, Z)$
$\mathrm{Height}(\mathit{Node}(l, x, r),\ S(n)) \Leftarrow \mathrm{Height}(l,\ n) \land \mathrm{Shal}(r, n)$
$\mathrm{Height}(\mathit{Node}(l, x, r),\ S(n)) \Leftarrow \mathrm{Height}(r,\ n) \land \mathrm{Shal}(l, n)$
$\mathrm{Shal}(\mathit{Leaf}, Z)$
$\mathrm{Shal}(\mathit{Leaf}, S(n))$
$\mathrm{Shal}(\mathit{Node}(l, x, r),\ S(n)) \Leftarrow \mathrm{Shal}(l, n) \land \mathrm{Shal}(l, n)$

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $\mathit{Vars}(B) \subseteq \mathit{Vars}(H)$

A set of SHoCs is a model that inductively defines relations.

Double

$\mathit{Double}(Z, Z)$
$\mathit{Double}(S(S(x_1)), S(x_2)) \Leftarrow \mathit{Double}(x_1, x_2)$

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $Vars(B) \subseteq Vars(H)$

A set of SHoCs is a model that inductively defines relations.

Double: forbidden

$Double(Z, Z)$
$Double(S(S(x_1)), S(x_2)) \Leftarrow Double(x_1, x_2)$

Shallow Horn Clauses (SHoCs)

Syntactic restriction of Horn clauses

A SHoC is a definite Horn clause $H \Leftarrow B$ such that

- The head H is linear and shallow
- The body B is flat
- There is no existential variable, i.e., $\mathit{Vars}(B) \subseteq \mathit{Vars}(H)$

A set of SHoCs is a model that inductively defines relations.

SHoCs are closed by boolean operations

$\text{Len} \mapsto \{([x_1, \ldots, x_n], S^n(Z)) \mid n \in \mathbb{N}\}$
$\text{Less} \mapsto \{(n_1, n_2) \mid n_1 < n_2\}$

Learner

$\text{Len}(Nil, Z)$

$\text{Less}(Z, S(Z))$

$\text{Len}(Nil, Z) \Rightarrow \text{Len}(Cons(Z, Nil), S(Z))$

$\text{Less}(Z, S(S(Z)))$

$$\begin{pmatrix}\text{Len}(Cons(Z, Nil), S(Z)) \;\; \wedge \\ \text{Len}(Cons(Z, Nil), S(S(Z)))\end{pmatrix} \implies \bot$$

$\text{Less}(S(Z), S(Z)) \Rightarrow \text{Less}(Z, Z)$

Teacher

$\text{Len}(Nil, \; Z)$
$\text{Len}(l, \; n) \Rightarrow \text{Len}(Cons(x, l), \; S(n))$
$\text{Len}(l, \; n_1) \wedge \text{Len}(l, \; n_2) \Rightarrow n_1 = n_2$

$\text{Less}(Z, \; S(m))$
$\text{Less}(n, \; Z) \Rightarrow \text{False}$
$\text{Less}(n, \; m) \Leftrightarrow \text{Less}(S(n), \; S(m))$

$\text{Len}(l, \; n) \wedge \text{Len}(Cons(x, l), \; n') \Rightarrow \text{Less}(n, \; n')$

Len $\mapsto$
Less $\mapsto$
SHoCs that defines Len and Less

Learner

$\mathrm{Len}(\mathit{Nil}, Z)$

$\mathrm{Less}(Z, S(Z))$

$\mathrm{Len}(\mathit{Nil}, Z) \Rightarrow \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$

$\mathrm{Less}(Z, S(S(Z)))$

$$\begin{pmatrix} \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \ \wedge \\ \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z))) \end{pmatrix} \Longrightarrow \bot$$

$\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)$

Teacher

$\mathrm{Len}(\mathit{Nil}, Z)$
$\mathrm{Len}(l, n) \Rightarrow \mathrm{Len}(\mathit{Cons}(x, l), S(n))$
$\mathrm{Len}(l, n_1) \wedge \mathrm{Len}(l, n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z, S(m))$
$\mathrm{Less}(n, Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n, m) \Leftrightarrow \mathrm{Less}(S(n), S(m))$

$\mathrm{Len}(l, n) \wedge \mathrm{Len}(\mathit{Cons}(x, l), n') \Rightarrow \mathrm{Less}(n, n')$

Details coming now:
Model generalisation

$\mathrm{Len} \mapsto$
$\mathrm{Less} \mapsto$
SHoCs that defines Len and Less

Learner:

$\mathrm{Len}(Nil, Z)$

$\mathrm{Less}(Z, S(Z))$

$\mathrm{Len}(Nil, Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil), S(Z))$

$\mathrm{Less}(Z, S(S(Z)))$

$$\begin{pmatrix} \mathrm{Len}(Cons(Z, Nil), S(Z)) \ \wedge \\ \mathrm{Len}(Cons(Z, Nil), S(S(Z))) \end{pmatrix} \implies \bot$$

$\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)$

Teacher:

$\mathrm{Len}(Nil,\ Z)$
$\mathrm{Len}(l,\ n) \Rightarrow \mathrm{Len}(Cons(x, l),\ S(n))$
$\mathrm{Len}(l,\ n_1) \wedge \mathrm{Len}(l,\ n_2) \Rightarrow n_1 = n_2$

$\mathrm{Less}(Z,\ S(m))$
$\mathrm{Less}(n,\ Z) \Rightarrow \mathrm{False}$
$\mathrm{Less}(n,\ m) \Leftrightarrow \mathrm{Less}(S(n),\ S(m))$

$\mathrm{Len}(l,\ n) \wedge \mathrm{Len}(Cons(x, l),\ n') \Rightarrow \mathrm{Less}(n,\ n')$

Learner: Model synthesis

$$\begin{array}{l}
\mathrm{Len}(\mathit{Nil}, Z) \\
\mathrm{Less}(Z, S(Z)) \\
\mathrm{Len}(\mathit{Nil}, Z) \Rightarrow \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \\
\mathrm{Less}(Z, S(S(Z))) \\
\begin{pmatrix} \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \ \wedge \\ \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z))) \end{pmatrix} \implies \bot \\
\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)
\end{array}$$

Learner: Model synthesis

$$\mathrm{Len}(\mathit{Nil}, Z)$$

$$\sout{\mathrm{Less}(Z, S(Z))}$$

$$\mathrm{Len}(\mathit{Nil}, Z) \Rightarrow \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$$

$$\sout{\mathrm{Less}(Z, S(S(Z)))}$$

$$\begin{pmatrix} \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \;\; \wedge \\ \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z))) \end{pmatrix} \implies \bot$$

$$\sout{\mathrm{Less}(S(Z), S(Z)) \Rightarrow \mathrm{Less}(Z, Z)}$$

Learner: Model synthesis

$$\mathrm{Len}(\mathit{Nil}, Z)$$
$$\mathrm{Len}(\mathit{Nil}, Z) \Rightarrow \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$$
$$\begin{pmatrix} \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \ \wedge \\ \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z))) \end{pmatrix} \implies \bot$$

Learner: Model synthesis

$$\mathrm{Len}(\mathit{Nil}, Z)$$
$$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$$
$$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))$$

Learner: Model synthesis

Input

$\mathrm{Len}(\mathit{Nil}, Z)$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))$

Learner: Model synthesis

Input

$\mathrm{Len}(\mathit{Nil}, Z)$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))$

Output

Learner: Model synthesis

Input

$\text{Len}(\mathit{Nil}, Z)$

$\neg\text{Len}(\mathit{Nil}, Z) \quad \vee \quad \text{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))$

$\neg\text{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \neg\text{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))$

- **Choose** one literal per clause that must be satisfied

Output

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$

$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$

$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
-
-
-

Output

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$

$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$

$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
-
-

Output

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
-
-

Output

$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow$
$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(Nil, Z)}$
$\neg\mathrm{Len}(Nil, Z) \quad \vee \quad \underline{\mathrm{Len}(Cons(Z, Nil), S(Z))}$
$\neg\mathrm{Len}(Cons(Z, Nil), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(Cons(Z, Nil), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
-

Output

$\mathrm{Len}(Nil, Z) \Leftarrow$
$\mathrm{Len}(Cons(x, l), S(n)) \Leftarrow$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
-

Output

$$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow \top$$
$$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow \top$$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow \top$
$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow \top$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow \top$
$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow \top$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow \top$$
$$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow \top$$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(Nil, Z)}$
$\neg\mathrm{Len}(Nil, Z) \quad \vee \quad \underline{\mathrm{Len}(Cons(Z, Nil), S(Z))}$
$\neg\mathrm{Len}(Cons(Z, Nil), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(Cons(Z, Nil), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC ←
- Verify that every chosen negative literal is not satisfied

Output

$\mathrm{Len}(Nil, Z) \Leftarrow \top$
$\mathrm{Len}(Cons(x, l), S(n)) \Leftarrow \top$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow \top$
$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow P(n)$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(Nil, Z)}$
$\neg\mathrm{Len}(Nil, Z) \quad \vee \quad \underline{\mathrm{Len}(Cons(Z, Nil), S(Z))}$
$\neg\mathrm{Len}(Cons(Z, Nil), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(Cons(Z, Nil), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\mathrm{Len}(Nil, Z) \Leftarrow \top$$
$$\mathrm{Len}(Cons(x, l), S(n)) \Leftarrow P(n) \wedge Q(l)$$

Learner: Model synthesis

Input

$$\underline{\mathrm{Len}(Nil, Z)}$$
$$\neg\mathrm{Len}(Nil, Z) \quad \lor \quad \underline{\mathrm{Len}(Cons(Z, Nil), S(Z))}$$
$$\neg\mathrm{Len}(Cons(Z, Nil), S(Z)) \quad \lor \quad \underline{\neg\mathrm{Len}(Cons(Z, Nil), S(S(Z)))}$$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\mathrm{Len}(Nil, Z) \Leftarrow \top$$
$$\mathrm{Len}(Cons(x, l), S(n)) \Leftarrow P(n) \land Q(l) \land R(x, l)$$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow \top$$
$$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow P(n) \wedge Q(l) \wedge R(x, l) \wedge \mathrm{Len}(l, n)$$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(Nil, Z)}$
$\neg\mathrm{Len}(Nil, Z) \quad \vee \quad \underline{\mathrm{Len}(Cons(Z, Nil), S(Z))}$
$\neg\mathrm{Len}(Cons(Z, Nil), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(Cons(Z, Nil), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\mathrm{Len}(Nil, Z) \Leftarrow \top$$
$$\mathrm{Len}(Cons(x, l), S(n)) \Leftarrow \mathrm{Len}(l, x)$$

Learner: Model synthesis

Input

$\underline{\text{Len}(Nil, Z)}$
$\neg\text{Len}(Nil, Z) \quad \vee \quad \underline{\text{Len}(Cons(Z, Nil), S(Z))}$
$\neg\text{Len}(Cons(Z, Nil), S(Z)) \quad \vee \quad \underline{\neg\text{Len}(Cons(Z, Nil), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\text{Len}(Nil, Z) \Leftarrow \top$$
$$\text{Len}(Cons(x, l), S(n)) \Leftarrow \text{Len}(l, x)$$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(Nil, Z)}$
$\neg\mathrm{Len}(Nil, Z) \quad \vee \quad \underline{\mathrm{Len}(Cons(Z, Nil), S(Z))}$
$\neg\mathrm{Len}(Cons(Z, Nil), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(Cons(Z, Nil), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\mathrm{Len}(Nil, Z) \Leftarrow \top$$
$$\mathrm{Len}(Cons(x, l), S(n)) \Leftarrow \mathrm{Len}(l, n)$$

Learner: Model synthesis

Input

$\underline{\mathrm{Len}(\mathit{Nil}, Z)}$
$\neg\mathrm{Len}(\mathit{Nil}, Z) \quad \vee \quad \underline{\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z))}$
$\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(Z)) \quad \vee \quad \underline{\neg\mathrm{Len}(\mathit{Cons}(Z, \mathit{Nil}), S(S(Z)))}$

- **Choose** one literal per clause that must be satisfied
- Create a SHoC to recognize each chosen positive atom
- **Choose** atoms for the body of every created SHoC
- Verify that every chosen negative literal is not satisfied

Output

$$\mathrm{Len}(\mathit{Nil}, Z) \Leftarrow \top$$
$$\mathrm{Len}(\mathit{Cons}(x, l), S(n)) \Leftarrow \mathrm{Len}(l, n)$$

Implementation

`https://gitlab.inria.fr/tlosekoo/auto-forestation`.

- Library for terms, clauses, SHoCs, etc...
  $\sim$ 5k lines of Ocaml code.
- **Learner** implemented with
  - $\sim$ 100 tricky lines of Clingo[1] for SHoCs inference
  - $\sim$ 500 lines of Ocaml for instance encoding/decoding
- **Teacher** implemented with heuristics to make it practical
  $\sim$ 2k lines of Ocaml code

[1] Clingo in an Answer Set Programming solver

Approximation

Some relations cannot be expressed by SHoCs

$$\mathrm{Plus}(n, Z, n)$$
$$\mathrm{Plus}(n, S(m), S(r)) \Leftarrow \mathrm{Plus}(n, m, r)$$
$$r_1 = r_2 \Leftarrow \mathrm{Plus}(n, m, r_1) \wedge \mathrm{Plus}(n, m, r_2)$$

Approximation

Some relations cannot be expressed by SHoCs

$$\mathrm{Plus}(n, Z, n)$$
$$\mathrm{Plus}(n, S(m), S(r)) \Leftarrow \mathrm{Plus}(n, m, r)$$
$$r_1 = r_2 \Leftarrow \mathrm{Plus}(n, m, r_1) \wedge \mathrm{Plus}(n, m, r_2)$$

Proving property

$$\mathrm{Leq}(n, r) \Leftarrow \mathrm{Plus}(n, m, r)$$

does not require to exactly represent Plus and Leq, as any over-approximation Plus^+ and under-approximation Leq^- is safe to prove this property.

Approximation

Some relations cannot be expressed by SHoCs

$$\mathrm{Plus}(n, Z, n)$$
$$\mathrm{Plus}(n, S(m), S(r)) \Leftarrow \mathrm{Plus}(n, m, r)$$
$$\cancel{r_1 = r_2 \Leftarrow \mathrm{Plus}(n, m, r_1) \wedge \mathrm{Plus}(n, m, r_2)}$$

Proving property

$$\mathrm{Leq}(n, r) \Leftarrow \mathrm{Plus}(n, m, r)$$

does not require to exactly represent Plus and Leq, as any over-approximation Plus^+ and under-approximation Leq^- is safe to prove this property.

Completeness and termination

Relative completeness of the Learner/Teacher procedure

If there exists a SHoCs satisfying every input clause, then the Learner will end up finding one.

Completeness and termination

Relative completeness of the Learner/Teacher procedure

If there exists a SHoCs satisfying every input clause, then the Learner will end up finding one.

- There may only exist models of clauses that are outside of SHoCs expressivity, in which case the Learner/Teacher loops forever
- If the model proposed by the Learner satisfies every clause, then the Teacher may not terminate

Completeness and termination

Relative completeness of the Learner/Teacher procedure

If there exists a SHoCs satisfying every input clause, then the Learner will end up finding one.

- There may only exist models of clauses that are outside of SHoCs expressivity, in which case the Learner/Teacher loops forever
- If the model proposed by the Learner satisfies every clause, then the Teacher may not terminate

✓	$\forall l, x,$	$less\ (length\ l)\ (length\ (Cons(x,\ l)))$
✓	$\forall p, l,$	$(prefix\ p\ (concat\ p\ l))$
✓	$\forall l,$	$length\ l\ =\ length\ (insert\_sort\ l)$
✓	$\forall t,$	$leq\ (height\ t)\ (size\ t)$
×	$\forall t,$	$height\ t\ =\ height\ (mirror\ t)$
×	$\forall x, l,$	$(count\ x\ (Cons(x, l)))\ =\ S(count\ x\ l)$

Conclusion

What is the problem, again?

Automatically prove relational properties on first-order monomorphic programs transforming algebraic data structures

In this talk

- Learner/Teacher overview
- Shallow Horn Clauses to represent models

Shallow Horn Clauses = Relational Alternating Tree Automata