The Extended Theory of Trees and Algebraic (Co)datatypes

Fabian Zaiser Luke Ong

Department of Computer Science

UNIVERSITY OF OXFORD

HCVS@ETAPS 2020 (published), 2021 (presented)

Outline

Background

- Trees
- Algebraic (Co)Datatypes

Contributions

- Relationship between trees and (co)datatypes
- Deciding first-order theory of trees

Outline

Background

- Trees
- Algebraic (Co)Datatypes

Contributions

- Relationship between trees and (co)datatypes
- Deciding first-order theory of trees

Outline

Background

- Trees
- Algebraic (Co)Datatypes

Contributions

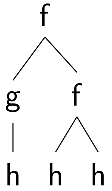
- Relationship between trees and (co)datatypes
- Deciding first-order theory of trees

Trees

- nodes have labels
- children are ordered

Trees

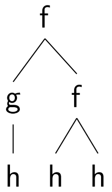
- nodes have labels
- children are ordered



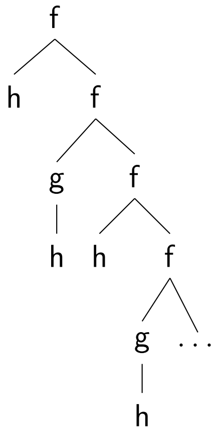
$$\mathsf{f}(\mathsf{g}(\mathsf{h}), \mathsf{f}(\mathsf{h}, \mathsf{h}))$$

Trees

- nodes have labels
- children are ordered



$$\mathsf{f}(\mathsf{g}(\mathsf{h}), \mathsf{f}(\mathsf{h}, \mathsf{h}))$$



$$x = \mathsf{f}(\mathsf{h}, \mathsf{f}(\mathsf{g}(\mathsf{h}), x))$$

First-order theory of trees

Classic Equational Theory of Trees:

- Function symbols/labels: $F = \{\mathsf{f} : 2, \mathsf{g} : 1, \mathsf{h} : 0, \dots\}$ with arities
- Predicate symbols: $P = \emptyset$
- Theory of Finite Trees & Theory of Infinite Trees
- Example formula: $\exists x.\, x = \mathsf{f}(\mathsf{h}, \mathsf{f}(\mathsf{g}(\mathsf{h}), x))$?
- Decision procedures: MAHER 1988 and COMON & LESCANNE 1989 (independently)

First-order theory of trees

Classic Equational Theory of Trees:

- Function symbols/labels: $F = \{\mathsf{f} : 2, \mathsf{g} : 1, \mathsf{h} : 0, \dots\}$ with arities
- Predicate symbols: $P = \emptyset$
- Theory of Finite Trees & Theory of Infinite Trees
- Example formula: $\exists x.\, x = \mathsf{f}(\mathsf{h}, \mathsf{f}(\mathsf{g}(\mathsf{h}), x))$?
- Decision procedures: MAHER 1988 and COMON & LESCANNE 1989 (independently)

Extended Theory of Trees (DJELLOUL, DAO, FRÜHWIRTH 2008):

- Predicate symbols $P = \{\mathrm{fin}\}$: $\mathrm{fin}(t)$ means t is a finite tree
- subsumes Theory of *Finite* and *Infinite* Trees
- Decision procedure with a restriction: F must be infinite

Applications

- matching and unification
- semantics of logic, functional programs
- recursion schemes
- verification of programs
- term rewriting systems
- **in this talk:** algebraic (co)datatypes

Outline

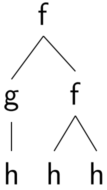
Background

- Trees
- Algebraic (Co)Datatypes

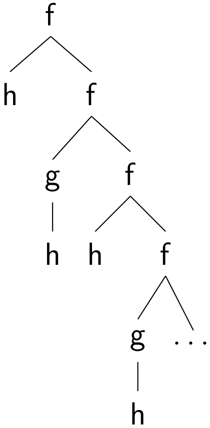
Contributions

- Relationship between trees and (co)datatypes
- Deciding first-order theory of trees

Algebraic (Co)Datatypes

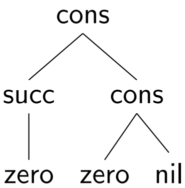


$$x = \mathsf{f}(\mathsf{g}(\mathsf{h}), \mathsf{f}(\mathsf{h}, \mathsf{h}))$$

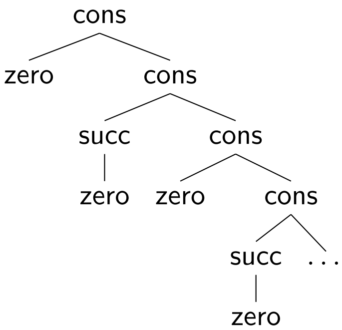


$$x = \mathsf{f}(\mathsf{h}, \mathsf{f}(\mathsf{g}(\mathsf{h}), x))$$

Algebraic (Co)Datatypes



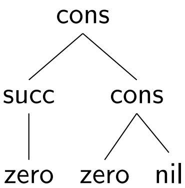

```
x = cons(succ(zero),
   cons(zero, nil))
```



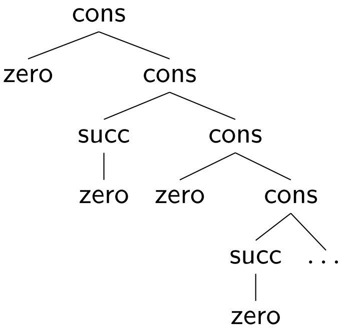
Algebraic Datatypes (inductive, least fixed point):

```
data   nat    = zero  | succ(pred: nat)
data   list   = nil   | cons(head: nat,  tail: list)
```

Algebraic (Co)Datatypes




```
x = cons(succ(zero),
   cons(zero, nil))
```



Algebraic Datatypes (inductive, least fixed point):

```
data    nat     = zero   | succ(pred: nat)
data    list    = nil    | cons(head: nat,  tail: list)
```

Algebraic Codatatype (coinductive, greatest fixed point):

```
codata colist = nil    | cons(head:nat, tail: colist)
```

Theory of Algebraic (Co)Datatypes

- Sorts: $S = \underbrace{\{nat, list\}}_{S_{data}} \cup \underbrace{\{colist\}}_{S_{codata}}$
- Constructor symbols:
  $F_{ctr} = \{\mathsf{zero} : list, \mathsf{succ} : nat \rightarrow nat, \mathsf{nil} : list, \dots\}$
  $\rightarrow$ interpreted as tree constructors
- Selector symbols:
  $F_{sel} = \{pred : nat \rightarrow nat, head : list \rightarrow list, \dots\}$
  $\rightarrow$ interpreted as selector functions: $pred(\mathsf{succ}(x)) = x$
- Function symbols $F = F_{ctr} \cup F_{sel}$; predicate symbols: $P = \emptyset$
- interpretations of datatype terms must be finite
- datatypes can only contain datatypes, codatatypes only codatatypes

Theory of Algebraic (Co)Datatypes

- Sorts: $S = \underbrace{\{nat, list\}}_{S_{data}} \cup \underbrace{\{colist\}}_{S_{codata}}$
- Constructor symbols:
  $F_{ctr} = \{\mathsf{zero} : list, \mathsf{succ} : nat \to nat, \mathsf{nil} : list, \ldots\}$
  $\to$ interpreted as tree constructors
- Selector symbols:
  $F_{sel} = \{pred : nat \to nat, head : list \to list, \ldots\}$
  $\to$ interpreted as selector functions: $pred(\mathsf{succ}(x)) = x$
- Function symbols $F = F_{ctr} \cup F_{sel}$; predicate symbols: $P = \emptyset$
- interpretations of datatype terms must be finite
- datatypes can only contain datatypes, codatatypes only codatatypes
- first-order theory undecidable; quantifier-free: decidable
- part of SMT-LIB (and SMT solvers Z3, CVC4, ...)

Outline

Background

- Trees
- Algebraic (Co)Datatypes

Contributions

- Relationship between trees and (co)datatypes
- Deciding first-order theory of trees

Contributions

- we extend Theory of Trees to many-sorted logic
- we formalize its relationship with (co)datatypes
- we design a simplification procedure/constraint solver for the Extended Theory of Trees
  - quantifiers allowed
  - based on Djelloul, Dao, Frühwirth (2008)
  - but: finitely many function symbols allowed!
- proved correctness
- implementation: evaluated on `QF_DT` suite of the SMT-LIB

Contributions

- we extend Theory of Trees to many-sorted logic
- we formalize its relationship with (co)datatypes
- we design a simplification procedure/constraint solver for the Extended Theory of Trees
  - quantifiers allowed
  - based on Djelloul, Dao, Frühwirth (2008)
  - but: finitely many function symbols allowed!
- proved correctness
- implementation: evaluated on `QF_DT` suite of the SMT-LIB

$\Longrightarrow$ Extended Theory of Trees is useful and decidable

Outline

Background

- Trees
- Algebraic (Co)Datatypes

Contributions

- Relationship between trees and (co)datatypes
- Deciding first-order theory of trees

(Co)Datatypes $\rightsquigarrow$ Trees

```
data   nat    = zero  | succ  (pred: nat)
data   list   = nil   | cons  (head: nat,  tail: list)
codata colist = conil | cocons(cohead:nat,cotail:colist)
```

(Co)datatypes

- Sorts: $\{nat, list, colist\}$
- Constructor symbols: $C = \{\text{zero} : list, \text{succ} : nat \to nat, \text{nil} : list, \dots\}$
- Selector symbols: $S = \{pred : nat \to nat, \dots\}$

(Co)Datatypes $\rightsquigarrow$ Trees

(Co)datatypes

- Sorts: $\{nat, list, colist\}$
- Constructor symbols: $C = \{\mathsf{zero} : list, \mathsf{succ} : nat \to nat, \mathsf{nil} : list, \ldots\}$
- Selector symbols: $S = \{pred : nat \to nat, \ldots\}$

(Co)Datatypes $\rightsquigarrow$ Trees

(Co)datatypes

- Sorts: $\{nat, list, colist\}$
- Constructor symbols: $C = \{\mathsf{zero} : list, \mathsf{succ} : nat \to nat, \mathsf{nil} : list, \ldots\}$
- Selector symbols: $S = \{pred : nat \to nat, \ldots\}$

$\rightsquigarrow$

Trees

- Sorts: $\{nat, list, colist\}$
- Function symbols: $F = \{\mathsf{zero} : list, \mathsf{succ} : nat \to nat, \mathsf{nil} : list, \ldots\}$
- Selector symbols: $x = pred(t)$ $\rightsquigarrow \forall y.\, t = \mathsf{succ}(y) \to x = y$
- add $\mathrm{fin}(t)$ for all datatypes

(Co)Datatypes $\rightsquigarrow$ Trees

(Co)datatypes

- Sorts: $\{\mathit{nat}, \mathit{list}, \mathit{colist}\}$
- Constructor symbols: $C = \{\text{zero} : \mathit{list}, \text{succ} : \mathit{nat} \to \mathit{nat}, \text{nil} : \mathit{list}, \ldots\}$
- Selector symbols: $S = \{\mathit{pred} : \mathit{nat} \to \mathit{nat}, \ldots\}$

$\rightsquigarrow$

Trees

- Sorts: $\{\mathit{nat}, \mathit{list}, \mathit{colist}\}$
- Function symbols: $F = \{\text{zero} : \mathit{list}, \text{succ} : \mathit{nat} \to \mathit{nat}, \text{nil} : \mathit{list}, \ldots\}$
- Selector symbols: $x = \mathit{pred}(t)$ $\rightsquigarrow \forall y.\, t = \text{succ}(y) \to x = y$
- add $\mathrm{fin}(t)$ for all datatypes

Example:

In: $x = \text{cons}(\text{zero}, \mathit{tail}(w))$ $\rightsquigarrow$

Out: $\mathrm{fin}(x) \land \mathrm{fin}(w) \land \forall y : \mathit{nat}, z : \mathit{list}.\, w = \text{cons}(y, z) \to x = \text{cons}(\text{zero}, z)$

(Co)Datatypes $\rightsquigarrow$ Trees

Theorem

A quantifier-free formula in the theory of (co)datatypes can be effectively transformed into an equisatisfiable formula in the extended theory of trees (possibly including quantifiers).

(Co)Datatypes $\rightsquigarrow$ Trees

Theorem

A quantifier-free formula in the theory of (co)datatypes can be effectively transformed into an equisatisfiable formula in the extended theory of trees (possibly including quantifiers).

Why quantifier-free? $\rightarrow$ problem with unspecified selectors: $pred(\text{zero})$ could be anything.

(Co)Datatypes $\rightsquigarrow$ Trees

Theorem

A quantifier-free formula in the theory of (co)datatypes can be effectively transformed into an equisatisfiable formula in the extended theory of trees (possibly including quantifiers).

Why quantifier-free? $\to$ problem with unspecified selectors: $pred(\text{zero})$ could be anything.

Theorem

If selectors return a specific default value when called on the wrong constructor then any formula in the theory of (co)datatypes can be effectively transformed into an equisatisfiable one in the extended theory of trees.

Trees $\rightsquigarrow$ (Co)Datatypes?

- non-finite: $\neg\,\mathrm{fin}(x) \rightsquigarrow ???$
- if finite then ...else ...: $(\mathrm{fin}(t) \to \phi) \vee (\neg\,\mathrm{fin}(t) \to \psi) \rightsquigarrow ???$
- impossible!

Trees $\rightsquigarrow$ (Co)Datatypes?

- non-finite: $\neg\,\mathrm{fin}(x) \rightsquigarrow\ ???$
- if finite then …else … : $(\mathrm{fin}(t) \rightarrow \phi) \vee (\neg\,\mathrm{fin}(t) \rightarrow \psi) \rightsquigarrow\ ???$
- impossible!

$\Longrightarrow$ Trees are more expressive than (Co)Datatypes!

Outline

Deciding the Extended Theory of Trees

DJELLOUL, DAO, AND FRÜHWIRTH (2008) designed a simplification precedure:

- more than just a decision procedure
- outputs simplified formula (not just true or false)
- easy to read off all satisfying assignments to free variables

Deciding the Extended Theory of Trees

DJELLOUL, DAO, AND FRÜHWIRTH (2008) designed a simplification precedure:

- more than just a decision procedure
- outputs simplified formula (not just true or false)
- easy to read off all satisfying assignments to free variables

Their assumptions:

- just one sort
- infinitely many constructors/function symbols F

Deciding the Extended Theory of Trees

DJELLOUL, DAO, AND FRÜHWIRTH (2008) designed a simplification precedure:

- more than just a decision procedure
- outputs simplified formula (not just true or false)
- easy to read off all satisfying assignments to free variables

Their assumptions:

- just one sort
- infinitely many constructors/function symbols F

We lift those restrictions:

- many-sorted logic
- finitely many constructors allowed

$\rightarrow$ can be used for (co)datatypes

Complications

With finitely many function symbols ...

- case analysis on constructors:
  - $\forall x : nat.\, x = \mathsf{zero} \lor \exists y : nat.\, x = \mathsf{succ}(y) \rightsquigarrow$ ✓
  - $\forall y : nat.\, x \neq \mathsf{succ}(y) \rightsquigarrow x = \mathsf{zero}$

Complications

With finitely many function symbols ...

- case analysis on constructors:
  - $\forall x : nat.\, x = \textsf{zero} \lor \exists y : nat.\, x = \textsf{succ}(y) \rightsquigarrow$ ✓
  - $\forall y : nat.\, x \neq \textsf{succ}(y) \rightsquigarrow x = \textsf{zero}$
- sorts with only (in)finite inhabitants
  - $bool = \textsf{false} \mid \textsf{true}$
    $inftree = \textsf{c1}(inftree) \mid \textsf{c2}(inftree)$
  - $\forall x : bool.\, \mathrm{fin}(x) \rightsquigarrow$ ✓
  - $\forall x : inftree.\, \mathrm{fin}(x) \rightsquigarrow$ ✗

Complications

With finitely many function symbols . . .

- case analysis on constructors:
  - $\forall x : nat.\, x = \mathsf{zero} \lor \exists y : nat.\, x = \mathsf{succ}(y) \rightsquigarrow$ ✓
  - $\forall y : nat.\, x \neq \mathsf{succ}(y) \rightsquigarrow x = \mathsf{zero}$
- sorts with only (in)finite inhabitants
  - $bool = \mathsf{false} \mid \mathsf{true}$
    $inftree = \mathsf{c1}(inftree) \mid \mathsf{c2}(inftree)$
  - $\forall x : bool.\, \mathrm{fin}(x) \rightsquigarrow$ ✓
  - $\forall x : inftree.\, \mathrm{fin}(x) \rightsquigarrow$ ✗
- unique infinite inhabitants
  - $\neg\, \mathrm{fin}(x : nat) \rightsquigarrow x = \mathsf{succ}(x)$

The basic idea

Extension of DJELLOUL, DAO, FRÜHWIRTH (2008).

Perform case splitting:

- for sorts with finitely many constructors:
  - if $x : nat$ then $x = \mathsf{zero} \vee \exists y.x = \mathsf{succ}(y)$
  - Example: input $\exists x : nat.\, \alpha$ is transformed into $(\exists x.\, x = \mathsf{zero} \wedge \alpha) \vee (\exists x, y.\, x = \mathsf{succ}(y) \wedge \alpha)$

The basic idea

Extension of DJELLOUL, DAO, FRÜHWIRTH (2008).

Perform case splitting:

- for sorts with finitely many constructors:
  - if $x : nat$ then $x = \mathsf{zero} \vee \exists y. x = \mathsf{succ}(y)$
  - Example: input $\exists x : nat.\, \alpha$ is transformed into $(\exists x.\, x = \mathsf{zero} \wedge \alpha) \vee (\exists x, y.\, x = \mathsf{succ}(y) \wedge \alpha)$
- for sorts with finitely many (in)finite inhabitants:
  - if $x : nat$ then $\mathrm{fin}(x) \vee x = \mathsf{succ}(x)$

The basic idea

Extension of Djelloul, Dao, Frühwirth (2008).

Perform case splitting:

- for sorts with finitely many constructors:
  - if $x : nat$ then $x = \mathsf{zero} \vee \exists y.x = \mathsf{succ}(y)$
  - Example: input $\exists x : nat.\, \alpha$ is transformed into $(\exists x.\, x = \mathsf{zero} \wedge \alpha) \vee (\exists x, y.\, x = \mathsf{succ}(y) \wedge \alpha)$
- for sorts with finitely many (in)finite inhabitants:
  - if $x : nat$ then $\mathrm{fin}(x) \vee x = \mathsf{succ}(x)$
- but be clever about when to case split
  - avoid unnecessary work
  - avoid infinite loops

Results

Theorem

- *Our simplification procedure returns a simplified formula that is equivalent in the Extended Theory of Trees.*
- *Simplified formula allows reading off all satisfying assignments of free variables.*
- *If input formula is closed, the result is* true *or* false*.*

$\rightarrow$ proof in the paper

Results

Theorem

- *Our simplification procedure returns a simplified formula that is equivalent in the Extended Theory of Trees.*
- *Simplified formula allows reading off all satisfying assignments of free variables.*
- *If input formula is closed, the result is* true *or* false*.*

$\rightarrow$ proof in the paper

Demo!

Try it! $\rightarrow$ tinyurl.com/trees-codata

- $x = \mathsf{succ}(x) \vee \mathrm{fin}(x) \quad \rightsquigarrow \quad \mathrm{true}$
- $x \neq \mathsf{nil} \wedge \mathrm{fin}(x) \quad \rightsquigarrow \quad \exists y, z.\, x = \mathsf{cons}(y, z) \wedge \mathrm{fin}(y) \wedge \mathrm{fin}(z)$

Evaluation

Theory:

- worst-case: non-elementary time complexity
- but can't do better (Vorobyov 1996)

Evaluation

Theory:

- worst-case: non-elementary time complexity
- but can't do better (VOROBYOV 1996)

Practice:

- prototype implementation in Scala
- evaluated on 4000 tests of `QF_DT` (Quantifier-Free DataTypes) suite of the SMT-LIB
  - translate from Datatypes to Trees; then solve translated formula
  - 90% took < 1 second; 5% timed out after 10 seconds
  - standard SMT solvers have an advantage by solving the original quantifier-free formula directly
  - lots of "low-hanging fruit" for improvements

Evaluation

Theory:

- worst-case: non-elementary time complexity
- but can't do better (VOROBYOV 1996)

Practice:

- prototype implementation in Scala
- evaluated on 4000 tests of `QF_DT` (Quantifier-Free DataTypes) suite of the SMT-LIB
  - translate from Datatypes to Trees; then solve translated formula
  - 90% took < 1 second; 5% timed out after 10 seconds
  - standard SMT solvers have an advantage by solving the original quantifier-free formula directly
  - lots of "low-hanging fruit" for improvements

Try it yourself! $\rightarrow$ tinyurl.com/trees-codata

Summary

The Extended Theory of Trees is ...

- useful: for (co)datatypes, logic programming, term rewriting, verification, ...
- powerful: more expressive than (co)datatypes
- decidable: even admits a simplification procedure

Summary

The Extended Theory of Trees is . . .

- useful: for (co)datatypes, logic programming, term rewriting, verification, . . .
- powerful: more expressive than (co)datatypes
- decidable: even admits a simplification procedure

Future research:

- heuristics for simplification procedure
- Craig interpolation

Backup slides

The basic idea

Manipulate normal forms ϕ (DJELLOUL, ET AL 2008):

$$\phi \equiv \exists \bar{x}.\ \alpha \ \wedge \bigwedge_{i=1}^{n} \neg\ \phi_i$$

"simple conjunction"

nested normal form

The basic idea

Manipulate normal forms ϕ (DJELLOUL, ET AL 2008):

$$\phi \equiv \exists \bar{x}.\ \alpha\ \wedge \bigwedge_{i=1}^{n} \neg\ \phi_i$$

"simple conjunction" nested normal form

Perform case analysis:

- for sorts with finitely many constructors:
  - if $x : nat$ then $x = \mathsf{zero} \vee \exists y.x = \mathsf{succ}(y)$
  - $\exists x : nat.\, \alpha \wedge \ldots$
    $\rightsquigarrow (\exists x.\, x = \mathsf{zero} \wedge \alpha \wedge \cdots) \vee (\exists x, y.\, x = \mathsf{succ}(y) \wedge \alpha \wedge \cdots)$

The basic idea

Manipulate normal forms ϕ (DJELLOUL, ET AL 2008):

$$\phi \equiv \exists \bar{x}.\ \underbrace{\alpha}_{\text{“simple conjunction”}} \wedge \bigwedge_{i=1}^{n} \neg \underbrace{\phi_i}_{\text{nested normal form}}$$

Perform case analysis:

- for sorts with finitely many constructors:
  - if $x : nat$ then $x = \mathsf{zero} \vee \exists y . x = \mathsf{succ}(y)$
  - $\exists x : nat .\, \alpha \wedge \ldots$
    $\rightsquigarrow (\exists x .\, x = \mathsf{zero} \wedge \alpha \wedge \cdots) \vee (\exists x, y .\, x = \mathsf{succ}(y) \wedge \alpha \wedge \cdots)$
- for sorts with finitely many (in)finite inhabitants:
  - if $x : nat$ then $\operatorname{fin}(x) \vee x = \mathsf{succ}(x)$

The basic idea

Manipulate normal forms ϕ (DJELLOUL, ET AL 2008):

$$\phi \equiv \exists \bar{x}.\ \alpha \ \wedge \bigwedge_{i=1}^{n} \neg\ \phi_i$$

"simple conjunction" nested normal form

Perform case analysis:

- for sorts with finitely many constructors:
  - if $x : nat$ then $x = \mathsf{zero} \vee \exists y. x = \mathsf{succ}(y)$
  - $\exists x : nat.\, \alpha \wedge \ldots$
    $\rightsquigarrow (\exists x.\, x = \mathsf{zero} \wedge \alpha \wedge \cdots) \vee (\exists x, y.\, x = \mathsf{succ}(y) \wedge \alpha \wedge \cdots)$
- for sorts with finitely many (in)finite inhabitants:
  - if $x : nat$ then $\mathrm{fin}(x) \vee x = \mathsf{succ}(x)$
- but avoid infinite loops:
  $x = \mathsf{succ}(x) \quad \rightsquigarrow \quad \exists y. x = \mathsf{succ}(y) \wedge y = \mathsf{succ}(y) \quad \rightsquigarrow \quad \ldots$