# Algebraic Datatypes are Horn-Clause Theories

František Farka

University of Dundee, and
University of St Andrews
ff32@st-andrews.ac.uk

8 December 2015

# Introduction

University of St Andrews

## Logic programming

- Programming in Horn-clause logic
- Goals resolved by a search - SLD resolution
- Automated theorem proving (ATP)

## Functional programming

- Program specified by a term
- Type of a term is a proposition
- Interactive theorem proving (ITP)

# Introduction

University of St Andrews

## Logic programming

- Programming in Horn-clause logic
- Goals resolved by a search - SLD resolution
- Automated theorem proving (ATP)

## Functional programming

- Program specified by a term
- Type of a term is a proposition
- Interactive theorem proving (ITP)

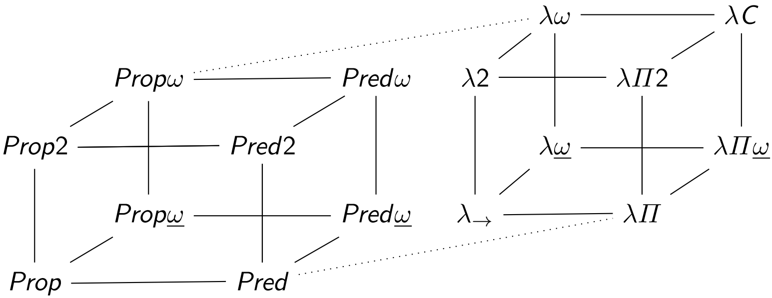
How are the two related?

# Introduction (cont)

University of St Andrews

## Propositions as Types

- Due to Barendregt, 1991
- Relating lambda calculi and different logics



# Propositional Logic Programming

University of St Andrews

## Propositional Logic Programming

- Infinite set of elementary propositions $\mathcal{P}$, propositions denoted `nat`, `bool`, ...
- A program is a set of Horn-clauses, i. e. clauses in the form
  $H \leftarrow B_1, \ldots B_n$
- Resolution step:

$$\frac{P \vdash \quad B_1 \quad , \ldots , \quad P \vdash \quad B_n}{P \vdash \quad A} \quad A \leftarrow B_0 \ldots B_n \in P$$

# Propositional Logic Programming

University of St Andrews

## Propositional Logic Programming

- Infinite set of elementary propositions $\mathcal{P}$, propositions denoted `nat`, `bool`, ...
- A set of clause names $\alpha, \beta_1, \ldots$ equipped with arity ($ar(\alpha) = 1, \ldots$)
- A program is a set of Horn-clauses, i. e. clauses in the form $\alpha : H \leftarrow B_1, \ldots B_n$ where $ar(\alpha) = n$
- Resolution step:

$$\frac{P \vdash \beta_1 : B_1 \qquad , \ldots , \qquad P \vdash \beta_n : B_n}{P \vdash \alpha(\beta_1, \ldots, \beta_n) : A} \; \alpha : A \leftarrow B_0 \ldots B_n \in P$$

# Propositional Logic Programming (cont.)

University of St Andrews

## A proof in PLP

- Success tree - all leafs are empty goals
- Applicative term as a proof

## Example

Resolution in $P_{\texttt{nat}} = \{\zeta : nat \; , \; \sigma : nat \leftarrow nat\}$

$$\dfrac{\dfrac{}{P \vdash \zeta : nat}\;\zeta : nat}{P \vdash \sigma(\zeta) : nat}\;\sigma : nat \leftarrow nat$$

# Propositional Logic Programming (cont.)

University of St Andrews

## A proof in PLP

- Success tree - all leafs are empty goals
- Note that success tree can have infinite branches (coinductive int.)
- Applicative term as a proof

## Example

Resolution in $P_{\texttt{nat}} = \{\zeta : nat \ , \ \sigma : nat \leftarrow nat\}$

$$\cfrac{\cfrac{}{P \vdash \zeta : nat}\ \zeta : nat}{P \vdash \sigma(\zeta) : nat}\ \sigma : nat \leftarrow nat$$

$$\cfrac{\cfrac{\ldots}{P \vdash \sigma(\ldots) : nat}\ \sigma : nat \leftarrow nat}{P \vdash \sigma(\sigma(\ldots)) : nat}\ \sigma : nat \leftarrow nat$$

# Propositional Logic Programming (cont.)

University of St Andrews

**Theorem: Inductive soundness and completeness**

A proposition $A$ is in the least model $M_P$ of a program P iff there is a finite term $\pi$ s. t. $P \vdash \pi : A$

**Theorem: Coinductive soundness**

A proposition $A$ is in the greatest model $M_P^\omega$ of a program P if there is an infinite term $\pi$ s. t. $P \vdash \pi : A$

# Propositional Logic Programming (cont.)

University of St Andrews

**Theorem: Inductive soundness and completeness**

A proposition $A$ is in the least model $M_P$ of a program P iff there is a finite term $\pi$ s. t. $P \vdash \pi : A$

**Theorem: Coinductive soundness**

A proposition $A$ is in the greatest model $M_P^\omega$ of a program P if there is an infinite term $\pi$ s. t. $P \vdash \pi : A$

Models are as usual in LP:

$$\mathcal{T}_P(X) = \{A \mid \alpha : A \leftarrow B_0, \ldots, B_{n-1} \in P \ \&\& \ \forall i \in 0, \ldots, n-1 \ B_i \in X\}$$

And $M_P = \mu(\mathcal{T}_P)$ and $M_P^\omega = \nu(\mathcal{T}_P)$

# Algebraic Datatypes

University of St Andrews

## Simply Typed Lambda Calculus ($\Lambda_{\rightarrow}$)

- Infinite set $V$ of variables $(x, y, \dots)$, and infinite set $B$ of type variables/identifiers: $\alpha, \beta, \dots$
- Function types: $\sigma \rightarrow \tau$
- Lambda abstraction, for $y : \sigma$ the expression $(\lambda x : \tau . y)$ is of type $\tau \rightarrow \sigma$
- Application, for $x : \sigma \rightarrow \tau$ and $y : \sigma$ is $xy : \tau$

# Algebraic Datatypes

University of St Andrews

## Simply Typed Lambda Calculus ($\Lambda_{\rightarrow}$)

- Infinite set $V$ of variables $(x, y, \ldots)$, and infinite set $B$ of type variables/identifiers: $\alpha, \beta, \ldots$ , nat, bool
- Function types: $\sigma \rightarrow \tau$
- Lambda abstraction, for $y : \sigma$ the expression $(\lambda x : \tau.y)$ is of type $\tau \rightarrow \sigma$
- Application, for $x : \sigma \rightarrow \tau$ and $y : \sigma$ is $xy : \tau$

## Algebraic Datatypes

- Constructors and eliminators/destructors for algebraic data types

# Algebraic Datatypes (cont.)

University of St Andrews

## Algebraic Datatypes

- Algebraic type is a type variable $\alpha$ and a set $C$ of $i$ constructors $c_j$
- Each constructor is equipped with arity $n$ and with a n-tuple of ADTs
- Inference rules:

$$\frac{\Gamma \vdash t_1 : \beta_{j,1} \qquad , \ldots , \qquad \Gamma \vdash t_{ar(c_j)} : \beta_{j,ar(c_j)}}{\Gamma \vdash c_j t_1 \ldots t_{ar(c_j)} : \alpha} \ \mathrm{CON}c_j$$

for $j = 0, \ldots, i-1$, and

$$\frac{\begin{array}{c}\Gamma \vdash t : \alpha \\ \Gamma, x_0 : \beta_{0,1}, \ldots, x_{ar(c_0)} : \beta_{0,ar(c_0)} \vdash s_0 : \gamma \\ , \ldots , \\ \Gamma, x_{i-1} : \beta_{i-1,1}, \ldots, x_{ar(c_{i-1})} : \beta_{i-1,ar(c_{i-1})} \vdash s_{i-1} : \gamma\end{array}}{\Gamma \vdash \mathrm{case}\ t\ \mathrm{of}\ (s_0, \ldots, s_{i-1}) : \gamma} \ \mathrm{CASE}\alpha$$

# Algebraic Datatypes (cont.)

University of St Andrews

## Example (Algebraic Datatypes)

```
data Bool where
    true  : () → Bool
    false : () → Bool

data Nat where
    zero : Nat
    succ : Nat → Nat

two : Nat
two = succ (succ zero)

prec : Nat → Nat
prec x = case x of
    zero       → zero
    succ x₀    → x₀
```

# ADTs are Horn-Clause Theories

University of St Andrews

## Translating ADTs to PLP, map $|\cdot|$

- Let there be an isomorphism of type variables $B$ and propositions $\mathcal{P}$ and of constructors and clause names
- Then for each constructor $c_j$ s. t. $c_j : (\beta_1, \ldots, \beta_n) \to \alpha$ $|c_j| = \gamma_j : A \leftarrow B_1, \ldots, B_n$

## Example (Nat and Bool)

- $|Nat| = \{\zeta : nat \ ; \ \sigma : nat \leftarrow nat\}$, $|Bool| = \{\tau : bool \ ; \ \phi : bool\}$

# ADTs are Horn-Clause Theories

University of St Andrews

## Translating ADTs to PLP, map $|\cdot|$

- Let there be an isomorphism of type variables $B$ and propositions $\mathcal{P}$ and of constructors and clause names
- Then for each constructor $c_j$ s. t. $c_j : (\beta_1, \ldots, \beta_n) \to \alpha$
  $|c_j| = \gamma_j : A \leftarrow B_1, \ldots, B_n$

## Example (Nat and Bool)

- $|Nat| = \{\zeta : nat \ ; \ \sigma : nat \leftarrow nat\}$, $|Bool| = \{\tau : bool \ ; \ \phi : bool\}$

## Lemma: PLP resolution for ADTs

- For an ADT $\alpha$ there exist a term $t$ such that

  $\Gamma \vdash_{\lambda_\to} t : \alpha$ iff $|\Gamma| \vdash_{PLP} \tau : A$ for some proof $\tau$

# Resolution with And-Or Trees

University of St Andrews

## And-Or Trees

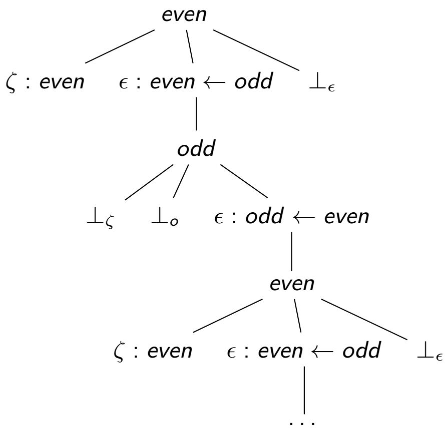
- Due to Komendantskaya and Johann, 2015

### Example

For a program P:

$$\zeta : \textit{even}$$
$$\epsilon : \textit{even} \leftarrow \textit{odd}$$
$$o : \textit{odd} \leftarrow \textit{even}$$

resolve goal *even*



# Resolution with And-Or Trees (cont.)

University of St Andrews

## Inductive success

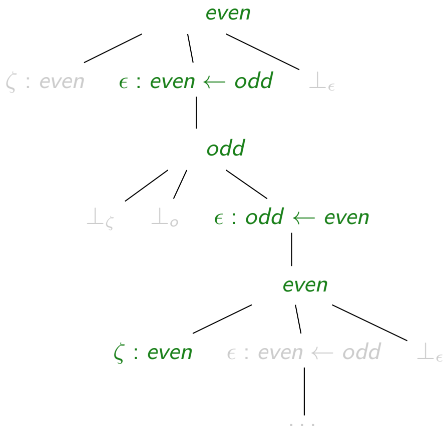
- A finite subtree; all children in and-nodes any one child in or-nodes.

## Example

For a program P:

$$\zeta : even$$
$$\epsilon : even \leftarrow odd$$
$$o : odd \leftarrow even$$

resolve goal *even* with $\epsilon(o(\zeta))$



# Resolution with And-Or Trees (cont.)

University of St Andrews

## Inductive success

- A finite subtree; all children in and-nodes any one child in or-nodes.

## Example

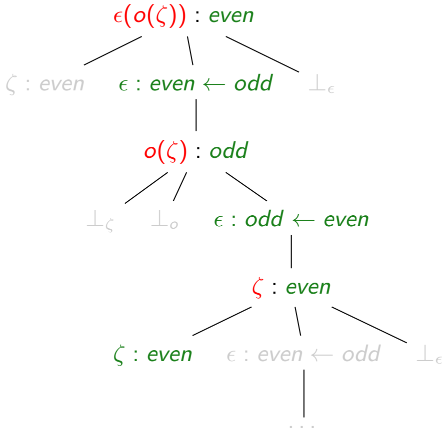
For a program P:

$$\zeta : \mathit{even}$$

$$\epsilon : \mathit{even} \leftarrow \mathit{odd}$$

$$o : \mathit{odd} \leftarrow \mathit{even}$$

resolve goal *even* with $\epsilon(o(\zeta))$



# Resolution with And-Or Trees (cont.)

University of St Andrews

## Coinductive success

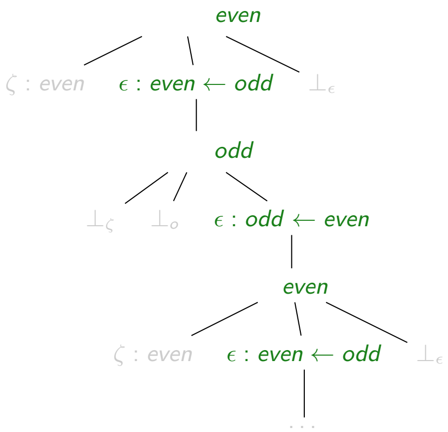
- An infinite subtree; all children in and-nodes any one child in or-nodes.

### Example

For a program P:

$$\zeta : even$$
$$\epsilon : even \leftarrow odd$$
$$o : odd \leftarrow even$$

resolve goal *even*



# Resolution with And-Or Trees (cont.)

University of St Andrews

## Coinductive success

- An infinite subtree; all children in and-nodes any one child in or-nodes.

### Example

For a program P:

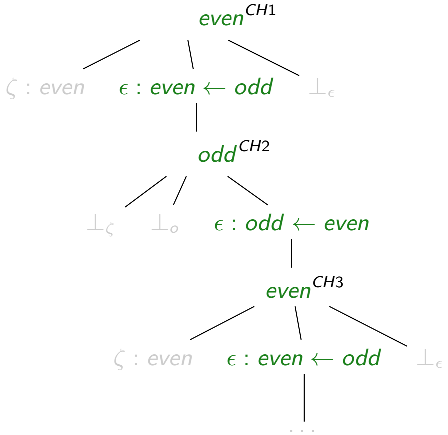
$\zeta : even$

$\epsilon : even \leftarrow odd$

$o : odd \leftarrow even$

resolve goal *even* with

$\nu h.\sigma\epsilon h = \epsilon(o(\sigma(\epsilon(\ldots))))$




$CH1 = \emptyset \quad CH2 = \{even\}$

$CH3 = \{even, odd\}$

# Resolution with And-Or Trees (cont.)

University of St Andrews

## Coinductive success

- An infinite subtree; all children in and-nodes any one child in or-nodes.

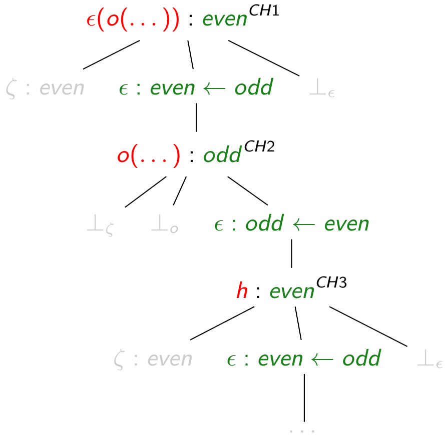
### Example

For a program P:

$$\zeta : even$$
$$\epsilon : even \leftarrow odd$$
$$o : odd \leftarrow even$$

resolve goal *even* with

$\nu h.\epsilon o h = \epsilon(o(\epsilon(o(\dots))))$




$CH1 = \emptyset \quad CH2 = \{even\}$

$CH3 = \{even, odd\}$

# Resolution with And-Or Trees (cont.)

University of St Andrews

## Theorem: Closing Infinite Branches with Coinductive Hypothesis

- For every infinite branch in a resolution tree $T$ there are or nodes $A$ and $A'$ s. t. $A' \in CH_A$, and
- the tree $T_A$ in the node $A$ is isomorphic to the tree $T'_A$ in $A'$.

## Observation: Inductive solutions

- Therefore each coinductively closed hypothesis generates inductive solution of the form

$$\sigma(\mu_i h.\tau(h))\upsilon$$

  where $\sigma$, $\tau$, and $\upsilon$ are finite terms and $\mu_i$ denotes $i$ iterations.
- and a coinductive solution of the form

$$\sigma(\nu h.\tau(h))$$

# Future Work

University of St Andrews

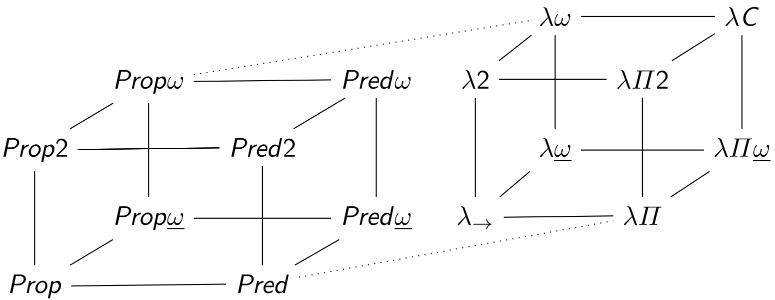
## ~~Future~~ Current work

- Get this worked out formally . . .
- Figure out how to treat nested function types
- Figure out how to treat function types in constructor fields

# Future Work

## Future work

- Second and higher order logic ($\lambda$Prolog - Miller, Nadathur, *et alii* ; $\alpha$Prolog - Cheney, Urban)
  - brings in polymorphism
- Predicate logic (S-resolution - Komendantskaya, Johann *et alii*
  - brings in dependent types
  - see difference in resolution by term matching and by unification gives



# Discussion

University of St Andrews

Thank you