Algebraic Datatypes are Horn-Clause Theories

František Farka

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

26 November 2015

University of St Andrews | FOUNDED 1413

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

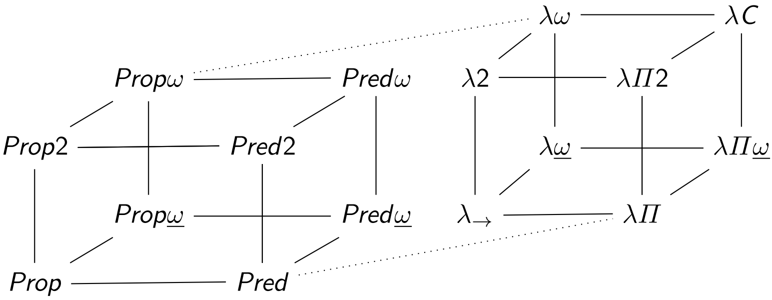
- Program specified by a term
- Type of a term is a proposition
- Interactive theorem proving (ITP)

How are the two related?

Introduction (cont)

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
  $H \leftarrow B_0, \ldots B_n$
- Resolution step:

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

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 $\alpha, \beta_0, \ldots$ equipped with arity ($ar(\alpha) = 1, \ldots$)
- A program is a set of Horn-clauses, i. e. clauses in the form $\alpha : H \leftarrow B_0, \ldots B_n$ where $ar(\alpha) = n$
- Resolution step:

$$\frac{P \vdash \beta_0 : B_0 \qquad , \ldots , \qquad P \vdash \beta_n : B_n}{P \vdash \alpha(\beta_0, \ldots, \beta_n) : A} \; \alpha : A \leftarrow B_0 \ldots B_n \in P$$

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

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

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 $P_{\mathtt{nat}} = \{\zeta : nat \ , \ \sigma : nat \leftarrow nat\}$

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

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

Propositional Logic Programming (cont.)

University of St Andrews

Theorem: Inductive soundness and completeness

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

Theorem: Coinductive soundness

A proposition A is in the greatest Herbrand model M_P^ω of a program P if there is a finite term π 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 Herbrand model M_P of a program P iff there is a finite term π s. t. $P \vdash \pi : A$

Theorem: Coinductive soundness

A proposition A is in the greatest Herbrand model M_P^ω of a program P if there is a finite term π s. t. $P \vdash \pi : A$

Herbrand models are as usual in LP

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: $\alpha, \beta, \ldots$
- 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 $(x, y, \dots)$, and infinite set B of type variables/identifiers: $\alpha, \beta, \dots$, 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 α and a set C of i constructors c_i
- Each constructor is equipped with arity n and with a n-tuple of types
- Inference rules:

$$\frac{\Gamma \vdash t_0 : \beta_{j,0} \qquad , \ldots , \qquad \Gamma \vdash t_{ar(c_j)} : \beta_{j,ar(c_j)}}{\Gamma \vdash c_i t_0 \ldots t_{ar(c_j)} : \alpha} \ \mathrm{CON} c_j$$

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

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

Algebraic Datatypes (cont.)

University of St Andrews

Example (Algebraic Datatypes)

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

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_i s. t. $c_i : (\beta_0, \ldots, \beta_n) \to \alpha$ $|c_i| = \gamma_i : A \leftarrow B_0, \ldots, B_n$

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_i s. t. $c_i : (\beta_0, \ldots, \beta_n) \rightarrow \alpha$ $|c_i| = \gamma_i : A \leftarrow B_0, \ldots, B_n$

Example (Nat and Bool)

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

Lemma: PLP resolution for ADTs

- For an ADT α and for a term t the following holds:

  $\mathit{\Gamma} \vdash_{\lambda_{\rightarrow}} t : \alpha \quad \text{iff} \quad |\mathit{\Gamma}| \vdash_{PLP} \tau : A$ for some proof τ
- and further $\mathit{\Gamma} \vdash_{\lambda_{\rightarrow}} |\tau|^{-1}_{Cl} : \alpha$

Extending PLP resolution

University of St Andrews

Extending resolution step

We add a new resolution step for a goal of the shape $A \leftarrow B_0, \ldots, B_n$: Let C be all the clauses $\gamma_0, \ldots, \gamma_i$ of P with head A and let $B_{\gamma_i, j}$ be the j-th preposition in the body of the clause γ_i

$$\frac{\begin{array}{c} P, x_0 : B_{\gamma_0,0}, \ldots x_{ar(\gamma_0)} : B_{\gamma_0, ar(\gamma_0)}, \vdash \delta_0 : D \\ \ldots \\ P, x_0 : B_{\gamma_i,0}, \ldots x_{ar(\gamma_i)} : B_{\gamma_i, ar(\gamma_i)}, \vdash \delta_i : D \end{array}}{P \vdash \lambda h.\text{case } h \text{ of } (\delta_0, \ldots, \delta_i) : D \leftarrow A} \; \gamma_0 \ldots \gamma_i$$

Extending PLP resolution

University of St Andrews

Extending resolution step

We add a new resolution step for a goal of the shape $A \leftarrow B_0, \ldots, B_n$:
Let C be all the clauses $\gamma_0, \ldots, \gamma_i$ of P with head A and let $B_{\gamma_i, j}$ be the j-th preposition in the body of the clause γ_i

$$\frac{\begin{array}{c} P, x_0 : B_{\gamma_0,0}, \ldots x_{ar(\gamma_0)} : B_{\gamma_0, ar(\gamma_0)}, \vdash \delta_0 : D \\ \ldots \\ P, x_0 : B_{\gamma_i,0}, \ldots x_{ar(\gamma_i)} : B_{\gamma_i, ar(\gamma_i)}, \vdash \delta_i : D \end{array}}{P \vdash \lambda h.\text{case } h \text{ of } (\delta_0, \ldots, \delta_i) : D \leftarrow A} \; \gamma_0 \ldots \gamma_i$$

Observation

- Under the translation $|\cdot|$ the set C is exactly the set of constructors of an ADT

ADTs ad Horn-Clause Theories (cont.)

University of St Andrews

Extending map $|\cdot|$

- For an extension of a program P with a hypothesis - ADT constructor $c_i : (\beta_0, \ldots, \beta_n) \to \alpha$ let

$$P, |c_i| = P \cup \bigcup_{j=0}^{n} \{x_{c_i,j} : |\beta_j|\}$$

Lemma: PLP resolution for functions

- For a function $\alpha \to \beta$ where α, β are ADTs, and for a term t the following holds:

$$\Gamma \vdash_{\lambda_\to} t : \alpha \to \beta \text{ iff } |\Gamma| \vdash_{PLP} \tau : B \leftarrow A$$

  for some proof τ
- and further $\Gamma \vdash_{\lambda_\to} |\tau|^{-1} : \alpha \to \beta$

Resolution with And-Or Trees

University of St Andrews

And-Or Trees

- Due to Komendantskaya and Johann, 2015

Example

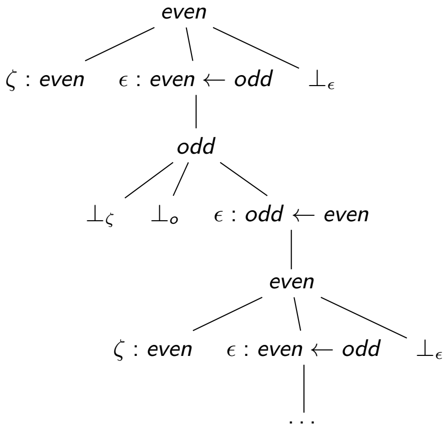
For a program P:

$\zeta : even$

$\epsilon : even \leftarrow odd$

$o : odd \leftarrow even$

resolve goal *even*



Resolution with And-Or Trees (cont.)

University of St Andrews

Inductive success

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

Example

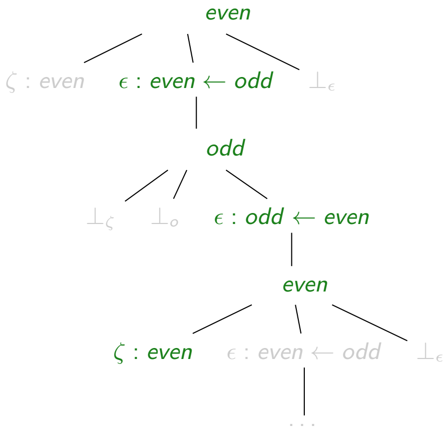
For a program P:

$\zeta : even$

$\epsilon : even \leftarrow odd$

$o : odd \leftarrow even$

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

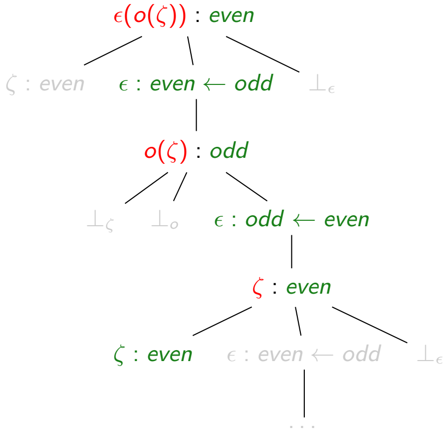
For a program P:

$$\zeta : even$$

$$\epsilon : even \leftarrow odd$$

$$o : odd \leftarrow even$$

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



Resolution with And-Or Trees (cont.)

University of St Andrews

Coinductive success

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

Example

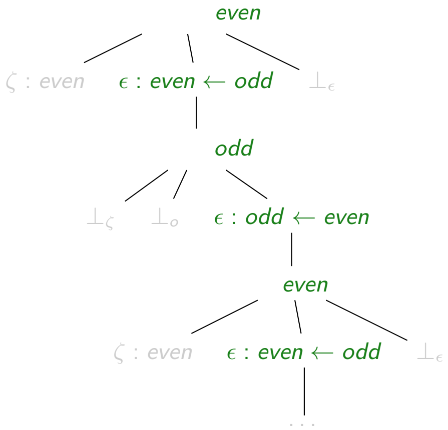
For a program P:

$\zeta : even$

$\epsilon : even \leftarrow odd$

$o : odd \leftarrow even$

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

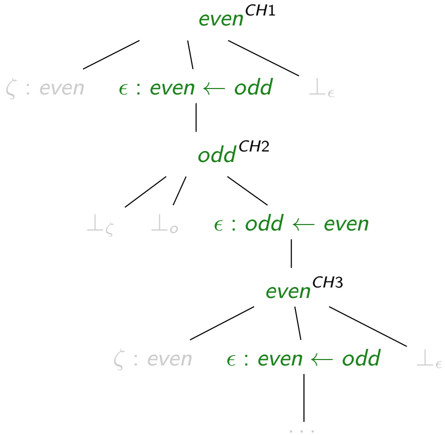
For a program P:

$\zeta : even$

$\epsilon : even \leftarrow odd$

$o : odd \leftarrow even$

resolve goal *even*



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

$CH3 = \{even, odd\}$

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.

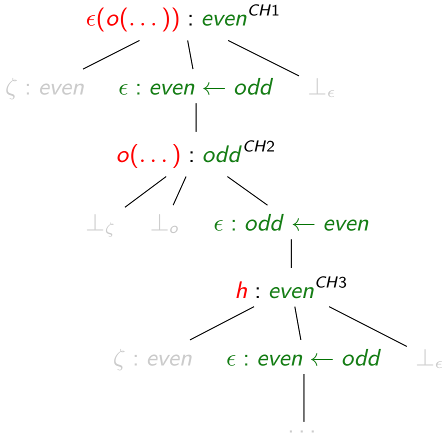
Example

For a program P:

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

resolve goal *even* with

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



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

$CH3 = \{even, odd\}$

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 is an or node A s. t. $A \in CH_A$, and
- the subtree T_A in the node A is isomorphic to T.

Observation: Inductive solutions

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

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

where σ, τ, and υ are finite terms and μ_i denotes i iterations.

Future Work

~~Future~~ Current work

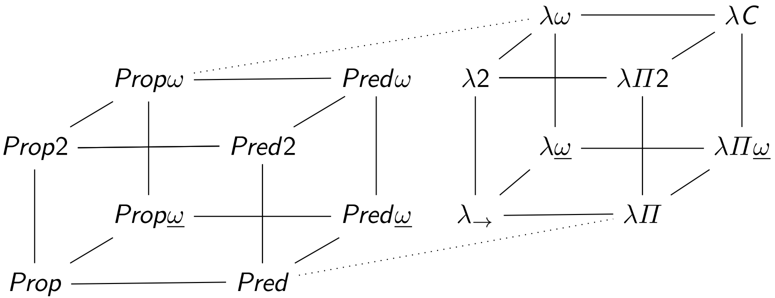
- Get this worked out formally ...
- Figure out how to treat nested function types
- Figure out how to treat function types in constructor fields

Future Work

University of St Andrews

Future work

- Second and higher order logic (λProlog - Miller, Nadathur, *et alii* ; αProlog - 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