Coalgebra for ATP

Katya Komendantskaya
joint work with H.Basold, Y.Li, J.Power, ...

School of Mathematical and Computer Sciences, Heriot-Watt University

Coalgebra, Now!
8 July 2018

Outline

Problem Statement

Coalgebra for ATP: never? later? or now?

Outline

Outline

Outline

Standard applications of Coalgebra

For a functor F, a *coalgebra* is a pair (U, c) consisting of a set U and a function $c : U \to F(U)$.

two main groups of applications

- coinductive types (sets with structure):
  E.g. *streams of elements of A*
  - functor $T(X) = A \times X$.
  - Final coalgebra for this functor is the set A^N of infinite lists with coalgebra structure $\langle head, tail \rangle : A^N \longrightarrow A \times A^N$.

Standard applications of Coalgebra

For a functor F, a *coalgebra* is a pair (U, c) consisting of a set U and a function $c : U \to F(U)$.

two main groups of applications

- coinductive types (sets with structure):
  E.g. *streams of elements of A*
  - functor $T(X) = A \times X$.
  - Final coalgebra for this functor is the set A^N of infinite lists with coalgebra structure $\langle head, tail \rangle : A^N \longrightarrow A \times A^N$.
- reactive, or transition, systems:
  E.g. *labelled transition system* $(S, A, \longrightarrow_S)$
  - functor: $B(X) = \mathcal{P}(A \times X)$
  - Final coalgebra is the set S of states such that $\alpha_s : S \longrightarrow \mathcal{P}(A \times S)$, with $s \mapsto \{< a, s' > | s \longrightarrow^a s'\}$

Jan J. M. M. Rutten: Universal coalgebra: a theory of systems. Theor. Comput. Sci. 249(1): 3-80 (2000)

What is Automated Theorem Proving

Usually:

- First-order logic as language
- with automated inference
- and set-theoretic models

What is Automated Theorem Proving

Usually:

- First-order logic as language
- with automated inference
- and set-theoretic models

Example

Prolog, SAT, SMT solvers, ... Horn Clause Logic is accepted as *lingua franca*

What is Automated Theorem Proving

Usually:

- First-order logic as language
- with automated inference
- and set-theoretic models

Example

Prolog, SAT, SMT solvers, ... Horn Clause Logic is accepted as *lingua franca*

Applications:

- automated verification
- help in proof search for higher-order provers (esp. HOL, but also Coq)
- type inference in functional and OO languages

Coalgebra and automated theorem proving?

Coalgebra: never?

two main groups of applications of coalgebra

- coinductive types (sets with structure)?
  – no types in the language

Coalgebra and automated theorem proving?

Coalgebra: never?

two main groups of applications of coalgebra

- coinductive types (sets with structure)?
  – no types in the language
- reactive, or transition, systems?
  – tempting to model inference as transition from formulae to formulae (rewriting system), but a lot more goes on there: unification, soundness relative to models

Horn Clause Logic

- Horn clause logic is a fragment of predicate logic, in which all formulae are written in clausal form.

Horn Clause Logic

- Horn clause logic is a fragment of predicate logic, in which all formulae are written in clausal form.
- Turing complete if taken as a programming language.

Why Horn clause Logic?

Well-defined model-theoretic properties:

- clean denotational (least and greatest) fixed point semantics. (Fixpoint construction for a monotone functor á la Knaster-Tarski.)

Why Horn clause Logic?

Well-defined model-theoretic properties:

- clean denotational (least and greatest) fixed point semantics. (Fixpoint construction for a monotone functor á la Knaster-Tarski.)

Sweet spot between expressivity and automation:

- It is long known to yield **efficient proofs by resolution**
  - logic of choice for first implementations of Prolog in 70s and 80s;
  - and many resolution-based provers in the 90s.
- SLD-resolution is not just sound but also complete relative to the least fixed point semantics.

Why Horn clause Logic?

In 2000s, emerged as a unifying language of ATP:

- allows elegant extensions to constraint LP and other enriched variants;
- a neat connection to Hoare Logic was discovered in 1987;
- in 2000s, Horn constraints have been shown to relate to Craig interpolation, which is one of the main techniques used to construct and refine abstractions in verification, and to synthesise inductive loop invariants;
- from 2010 onwards, increasingly used in SMT-solvers, model checkers, abstract interpretation (Bjorner, Rybalchenko);
- higher-order Horn clauses are used in model checkers of functional languages (Ong, Kobayashi)

Why Horn clause logic?

Neat connection to intuitionistic logic.

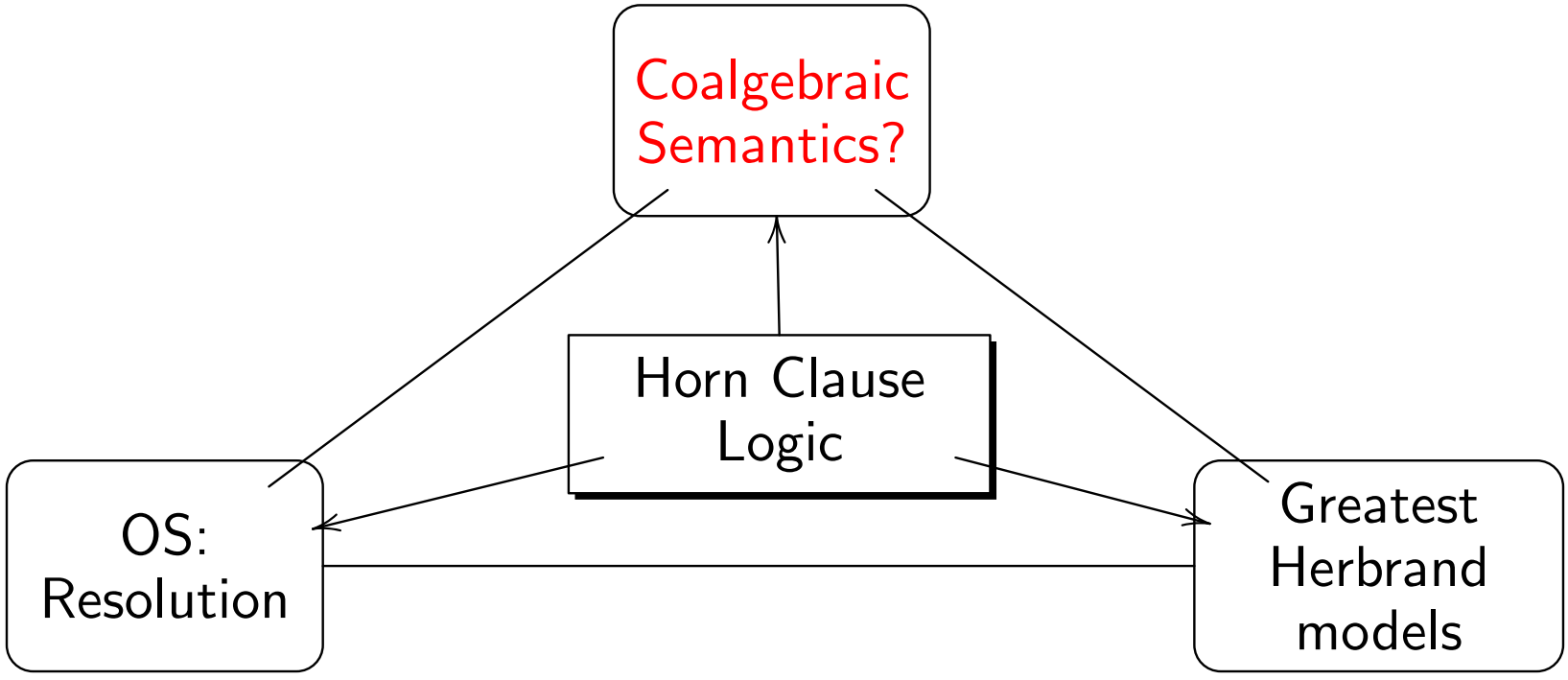
- In 1989, Girard suggested to use the cut rule to model resolution for Horn formulas.
- In the 1990s, Miller et. al. introduce uniform proofs (proof theoretic counterpart of resolution) – and show their soundness relative to intuitinsitic sequent calculus.
- Interactive theorem prover Twelf (by Pfenning et al.) pioneered implementation of proof search for Horn clause logic on top of a dependently typed system called LF (Harper & Licata).
- In 2016, Fu&Komendantskaya gave a Horn clause-as-types (proofs as terms) interpretation to a fragment of Horn clause logic.

Why Horn clause Logic?

Applications in Programming languages, via type inference

- Type classes in Haskell (Jones, 90s, ext. 2000s)
- GADTs in Haskell (Stuckey, Schrijvers, et al. late 90s onwards)
- Type Classes in Coq and SSReflect (Ziliani, Gonthier et al. 2014)
- Class inference in Java (Ancona et al, 2000s)

This talk:


Coalgebraic
Semantics?
Horn Clause
Logic
OS:
Resolution
Greatest
Herbrand
models

Outline

Syntax of Horn-clause Logic

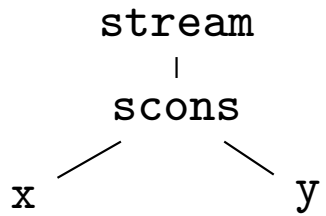
First-order signature Σ and terms, term-trees

- Σ: a set of function symbols with arity;
- V: a set of variables.

Example

- `stream` – arity 1
- `scons` – arity 2

Term-trees are trees over $\Sigma \cup V$, subject to branching $\approx$ arity:



Sets of terms:

$\mathbf{Term}(\Sigma)$	Set of *finite* term trees over Σ
$\mathbf{Term}^{\omega}(\Sigma)$	Set of *finite and infinite* term trees over Σ

Sets of terms:

$\mathbf{Term}(\Sigma)$	Set of *finite* term trees over Σ
$\mathbf{Term}^{\omega}(\Sigma)$	Set of *finite and infinite* term trees over Σ

$\mathbf{GTerm}(\Sigma)$,
$\mathbf{GTerm}^{\omega}(\Sigma)$
will denote sets of ground (variable free) terms over Σ.

Syntax of Horn-clause Logic

$$
\begin{array}{lll}
G & ::= & \top \mid A \mid G \wedge G \mid G \vee G \mid \exists Var\ G \\
D & ::= & A \mid G \to D \mid D \wedge D \mid \forall Var\ D
\end{array}
$$

Horn Clauses

Given $A, B_1, \ldots, B_n \in \mathbf{Term}(\Sigma)$,

- a (definite) Horn clause has the shape $\forall \overline{x}. B_1 \wedge \ldots \wedge B_k \to A$
- a goal clause $?B_1 \wedge \ldots \wedge B_k$

Example: inductive definitions in Horn clauses

Example

1. $\texttt{nat}\ \ \texttt{0}$
2. $\forall\, x.\, \texttt{nat}\ \ \texttt{x} \rightarrow \texttt{nat}\ \ (\texttt{s}\ \texttt{x})$
3. $\texttt{list}\ \ \texttt{nil}$
4. $\forall\, x\ y.\, (\texttt{nat}\ \ \texttt{x} \wedge \texttt{list}\ \ \texttt{y}) \rightarrow \texttt{list}\ \ (\texttt{cons}\ \ \texttt{x}\ \ \texttt{y})$

Example: Coinductive definitions

Example

1. $\texttt{bit}\ \ \texttt{0}$
2. $\texttt{bit}\ \ \texttt{1}$
3. $\forall\, x\ y.\, (\texttt{bit}\ \ \texttt{x} \wedge \texttt{stream}\ \ \texttt{y}) \rightarrow \texttt{stream(scons}\ \ \texttt{x}\ \ \texttt{y)}$

Inductive Semantics of LP

Definition (Big step rule)

$$\frac{P \models \sigma(B_1), \ldots, P \models \sigma(B_n)}{P \models \sigma(A)},$$

for some grounding substitution σ, and $\forall \overline{x}.B_1, \ldots B_n \rightarrow A \in P$.

Definition

The *least Herbrand model* for P is the smallest set $M_P \subseteq \mathbf{GTerm}(\Sigma)$ *closed forward* under the rules.

Example

Taking the logic program *Nat*, we obtain the set $M_{Nat} = \{\texttt{nat 0},\ \texttt{nat(s 0)}, \texttt{nat(s s 0)}, \ldots\}$.

Least Herbrand models as a least fixed point construction

Definition

The function $T_P : \mathcal{P}(\mathbf{GTerm}(\Sigma)) \to \mathcal{P}(\mathbf{GTerm}(\Sigma))$ is defined by $T_P(A) = \{\theta(t) \in \mathbf{GTerm}(\Sigma) \mid \forall \overline{x}.\ t_1, \ldots, t_n \to t \in P$ and each $\theta(t_i) \in A\ \}$.

Definition

The *least Herbrand model* for $P \in \mathbf{LP}(\Sigma)$ is the smallest set $M_P \in \mathcal{P}(\mathbf{GTerm}(\Sigma))$ such that

- $T_P(M_P) = M_P$.

i.e. M_P is the smallest fixed point of T_P.

Maarten H. van Emden, Robert A. Kowalski: The Semantics of Predicate Logic as a Programming Language. J. ACM 23(4): 733-742 (1976)

Kleene Chain Construction

Kleene Fixed-Point Theorem

Suppose $(L, \sqsubseteq)$ is a directed-complete partial order (dcpo) with a least element, and let $f : L \to L$ be a Scott-continuous (and therefore monotone) function. Then f has a least fixed point, which is the supremum of the ascending Kleene chain of f:

$$\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \ldots$$

Applied to least fixed point semantics:

$T_P \uparrow 0 = \emptyset$;
$T_P \uparrow n = T_P(T_P \uparrow (n-1))$

Theorem

$T_P \uparrow \omega = M_P$

CoInductive Semantics of LP

Definition (Big step rule)

$$\frac{P \models \sigma(B_1), \ldots, P \models \sigma(B_n)}{P \models \sigma(A)},$$

for some grounding substitution σ, and $\forall \overline{x}.B_1, \ldots B_n \rightarrow A \in P$.

Definition

The *greatest complete Herbrand model* for P is the largest set $M_P^\omega \subseteq \mathbf{GTerm}^\omega(\Sigma)$ *closed backward* under the rules.

Example

M_{Nat}^ω will now be given by the set:
$\{\texttt{nat 0}, \texttt{nat(s 0)}, \texttt{nat(s s 0)}, \ldots\} \bigcup \{\texttt{nat(s s ...)}\}$.

Greatest Herbrand models as a least fixed point construction

Definition

The function $T_P^\omega : \mathcal{P}(\mathbf{GTerm}^\omega(\Sigma)) \to \mathcal{P}(\mathbf{GTerm}^\omega(\Sigma))$ is defined by $T_P^\omega(A) = \{\theta(t) \in \mathbf{GTerm}^\omega(\Sigma) \mid \forall x.\ t_1, \ldots, t_n \to t \in P$ and each $\theta(t_i) \in A\ \}$.

Definition

The *greatest complete Herbrand model* for P is the largest set $M_P^\omega \in \mathcal{P}(\mathbf{GTerm}^\omega(\Sigma))$ such that

- $T_P^\omega(M_P^\omega) = M_P^\omega$.

i.e. M_P^ω is the greatest fixed point of T_P^ω.

Kleene Chain Construction

Kleene construction applied to greatest fixed point semantics:

$T_P^\omega \downarrow 0 = \mathbf{GTerm}^\omega(\Sigma)$;
$T_P^\omega \downarrow n = T_P^\omega(T_P^\omega \downarrow (n-1))$

Theorem

$T_P^\omega \downarrow \omega = M_P^\omega$

Coinductive programs

Some programs have only one natural interpretation:

Example

1. $\texttt{bit}\ \ \texttt{0}$
2. $\texttt{bit}\ \ \texttt{1}$
3. $\forall\, x\ y.\, (\texttt{bit}\ \ \texttt{x} \wedge \texttt{stream}\ \ \texttt{y}) \rightarrow \texttt{stream(scons}\ \ \texttt{x}\ \ \texttt{y)}$

$M_{Stream} = \{\texttt{bit}\ \ \texttt{0}, \texttt{bit}\ \ \texttt{1}\}$
$M^{\omega}_{Stream} =$
$\{\texttt{bit}\ \ \texttt{0}, \texttt{bit 1}, \texttt{stream(scons}\ \ \texttt{0}\ \ \texttt{(scons 0 ...))},$
$\texttt{stream(scons}\ \ \texttt{1 (scons}\ \ \texttt{0 ...))}, \ldots\}$

Universal vs Existential Coinductive Models

Example

$\forall x.p(f\ x) \rightarrow p\ x$

	least fixed point	greatest fixed point
finite terms	$\emptyset$	$\{p\ a, p(f\ a), p(f\ f\ a), \ldots\}$
finite and infinite terms	$\emptyset$	$\{p\ a, p(f\ a), p(f\ f\ a), \ldots$ $p\ f^{\omega}\ \}$

Universal vs Existential Coinductive Models

Example

$\forall x.p(f\ x) \rightarrow p\ x$

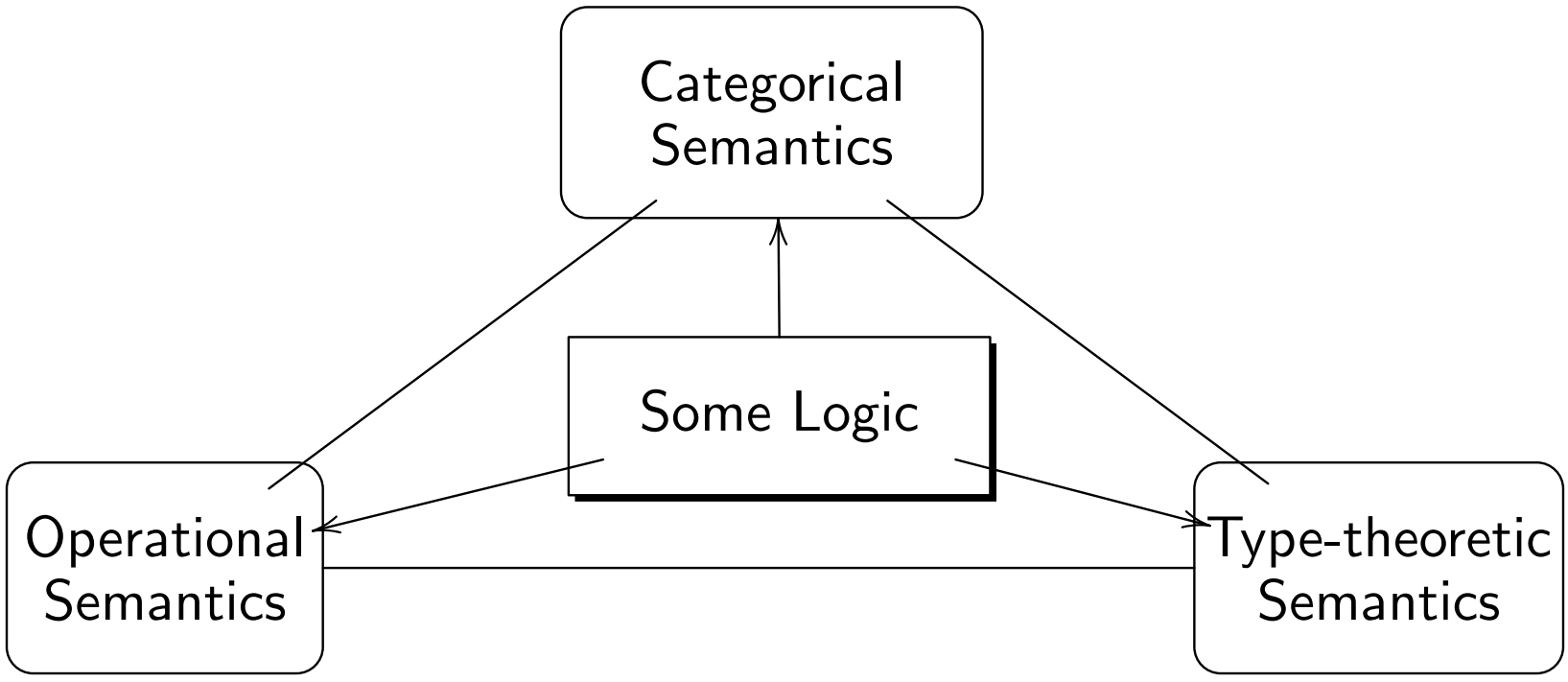
	least fixed point	greatest fixed point
finite terms	$\emptyset$	$\{p\ a, p(f\ a), p(f\ f\ a), \ldots\}$
finite and infinite terms	$\emptyset$	$\{p\ a, p(f\ a), p(f\ f\ a), \ldots p\ f^{\omega}\ \}$

Example

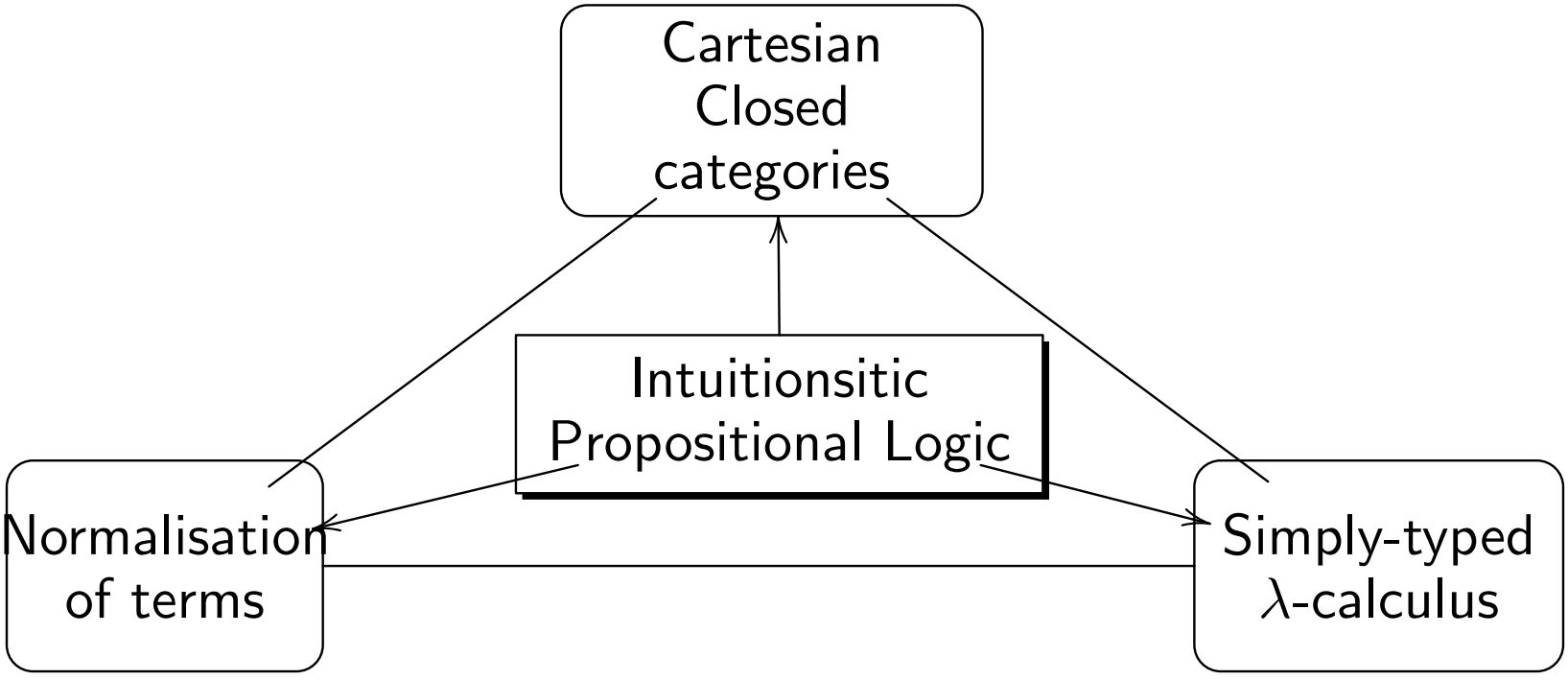
$\forall x.p\ x \rightarrow\ p(f\ x)$

	least fixed point	greatest fixed point
finite terms	$\emptyset$	$\emptyset$
finite and infinite terms	$\emptyset$	$\{p\ f^{\omega}\ \}$

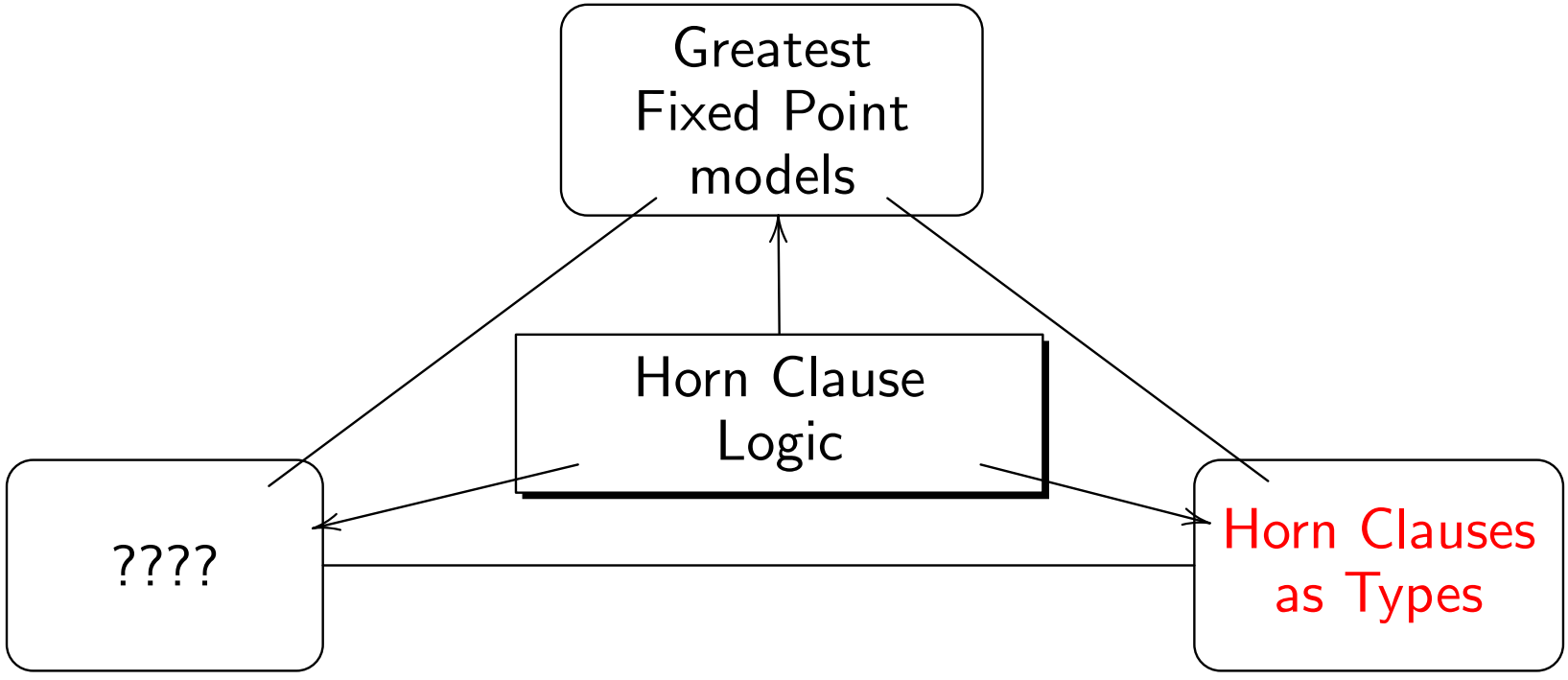
The Semantic triangle (cf. Curry-Howard-Lambek)



Curry - Howard - Lambek correspondence



Triangle, for Coinductive Horn clauses



Horn clauses as coinductive types?

Horn clauses as (dependent) coinductive types

Example

```
  Variable O : A.
  Variable I : A.

CoInductive bit: A -> Prop :=
 | bit_O: bit O
 | bit_I: bit I.

CoInductive bit_stream: Stream -> Prop :=
| bs : forall (x : A) (y: Stream),
     bit x /\ bit_stream y -> bit_stream (Cons x y).
```

The types define coinductive predicates, that determine the sets described by Greatest Herbrand models.

Coming back to our earlier discussion

Two main groups of applications of coalgebra

1. coinductive types (sets with structure):
   E.g. *streams of elements of A*
   - functor $T(X) = A \times X$.
   - Final coalgebra for this functor is the set A^N of infinite lists with coalgebra structure $\langle head, tail \rangle : A^N \longrightarrow A \times A^N$.
- But there are no types in Horn clause logic

Coming back to our earlier discussion

Two main groups of applications of coalgebra

1. coinductive types (sets with structure):
   E.g. *streams of elements of A*
   - functor $T(X) = A \times X$.
   - Final coalgebra for this functor is the set A^N of infinite lists with coalgebra structure $\langle head, tail \rangle : A^N \longrightarrow A \times A^N$.
- But there are no types in Horn clause logic

Horn clauses can be seen as coinductive types themselves!

Proof-relevant resolution

Peng Fu, Ekaterina Komendantskaya, Tom Schrijvers, Andrew Pond: Proof Relevant Corecursive Resolution. FLOPS 2016: 126-143

$$\frac{\Phi \vdash e_1 : \sigma B_1 \quad \cdots \quad \Phi \vdash e_n : \sigma B_n}{\Phi \vdash \kappa\ e_1 \cdots e_n : \sigma A} \text{ if } (\kappa : B_1, ..., B_n \Rightarrow A) \in \Phi$$

Proof-relevant resolution

Peng Fu, Ekaterina Komendantskaya, Tom Schrijvers, Andrew Pond: Proof Relevant Corecursive Resolution. FLOPS 2016: 126-143

$$\frac{\Phi \vdash e_1 : \sigma B_1 \quad \cdots \quad \Phi \vdash e_n : \sigma B_n}{\Phi \vdash \kappa\ e_1 \cdots e_n : \sigma A} \text{ if } (\kappa : B_1, ..., B_n \Rightarrow A) \in \Phi$$

Example

1. `nat 0`
2. $\forall\, x.$ `nat x` $\rightarrow$ `nat (s x)`
3. `list nil`
4. $\forall\, x\ y.$ `(nat x` $\wedge$ `list y)` $\rightarrow$ `list (cons x y)`

We Prove $P \vdash \kappa_4\kappa_1\kappa_3 : \mathit{list}(\mathit{cons}\ 0\ \mathit{nil})$

Proof-relevant corecursive resolution

Peng Fu, Ekaterina Komendantskaya, Tom Schrijvers, Andrew Pond: Proof Relevant Corecursive Resolution. FLOPS 2016: 126-143

$$\frac{\Phi, (\alpha : \underline{A} \Rightarrow B) \vdash e : \underline{A} \Rightarrow B \quad \mathrm{HNF}(e)}{\Phi \vdash \nu\alpha.e : \underline{A} \Rightarrow B}\ (\textsc{Mu}) \qquad \frac{\Phi, (\underline{\alpha} : \underline{A}) \vdash e : B}{\Phi \vdash \lambda\underline{\alpha}.e : \underline{A} \Rightarrow B}\ (\textsc{Lam})$$

Proof-relevant corecursive resolution

Peng Fu, Ekaterina Komendantskaya, Tom Schrijvers, Andrew Pond: Proof Relevant Corecursive Resolution. FLOPS 2016: 126-143

$$\frac{\Phi, (\alpha : \underline{A} \Rightarrow B) \vdash e : \underline{A} \Rightarrow B \quad \mathrm{HNF}(e)}{\Phi \vdash \nu\alpha.e : \underline{A} \Rightarrow B}\ (\textsc{Mu}) \qquad \frac{\Phi, (\underline{\alpha} : \underline{A}) \vdash e : B}{\Phi \vdash \lambda\underline{\alpha}.e : \underline{A} \Rightarrow B}\ (\textsc{Lam})$$

- Can prove coinductive properties of universal coinductive theories

Example

$\kappa : \forall x.p(f\ x) \to p\ x$
Prove
$\nu\alpha.\kappa\alpha : \forall x.p(x)$.

Current work

- Coalgebraic restatement of the greatest Herbrand model construction (H.Basold)
  - **GTerm**$^{\omega}(\Sigma)$ is defined as a coinductive data type
  - Kleene construction on top of it.
- Constructive calculus for coinductive proofs in Horn clause logic (general approach)

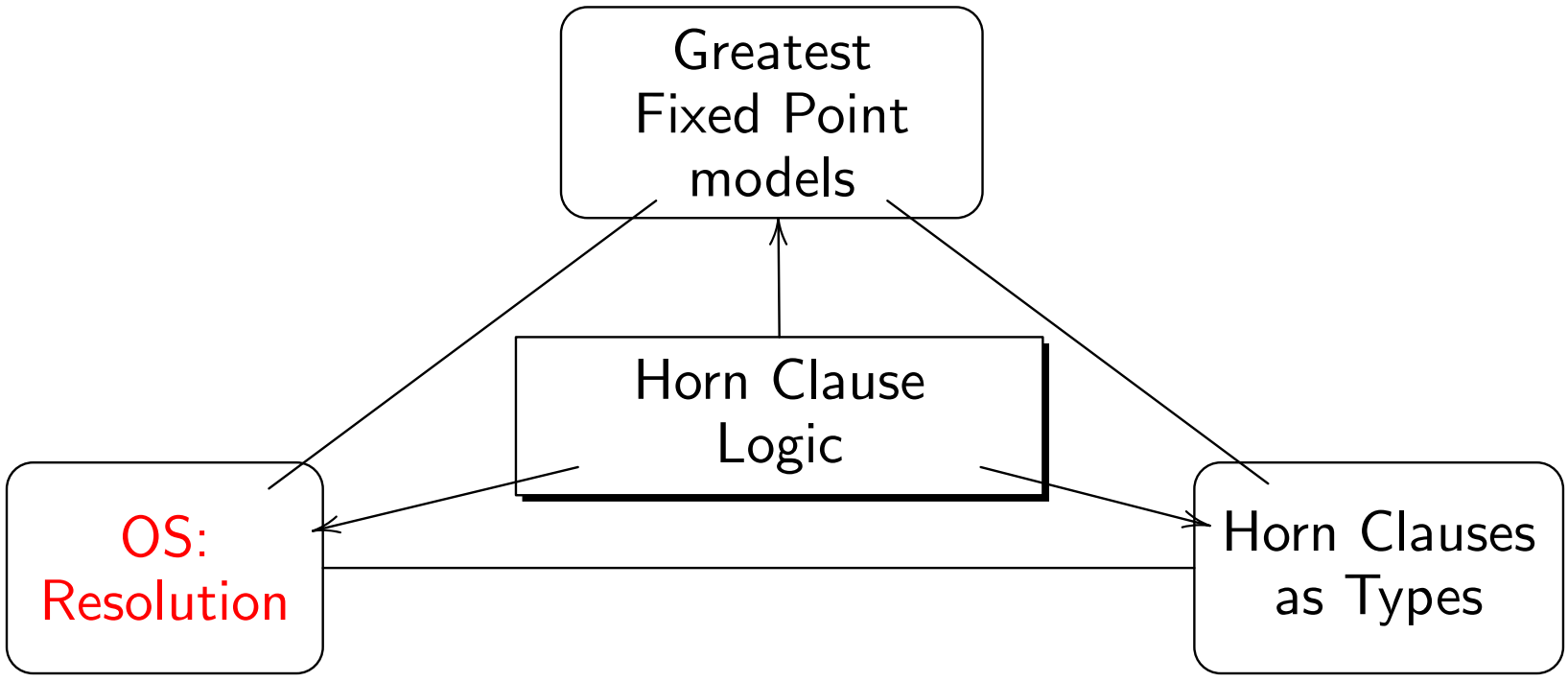
Example

1. `bit 0`
2. `bit 1`
3. $\forall\, x\ y.$ `(bit x` $\wedge$ `stream y)` $\rightarrow$ `stream(scons x y)`

How to prove $P \vdash ? : \exists x.\mathit{stream}\ x$

Outline

Triangle, for Coinductive Horn clauses



Standard applications of Coalgebra

For a functor F, a *coalgebra* is a pair (U, c) consisting of a set U and a function $c : U \to F(U)$.

two main groups of applications

1. coinductive types ...
2. reactive, or transition, systems:
   E.g. *labelled transition system* $(S, A, \longrightarrow_S)$
   - functor: $B(X) = \mathcal{P}(A \times X)$
   - Final coalgebra is the set S of states such that $\alpha_s : S \longrightarrow \mathcal{P}(A \times S)$, with $s \mapsto \{< a, s' > | s \longrightarrow^a s'\}$

Intuition: if we do not have labels, than just $F(X) = \mathcal{P}(X)$ should work...

Operationally...

SLD resolution = Unification + Search

(small step semantics)

Given a logic program P, and terms $t_1, \ldots, t_i, \ldots, t_n$ we define

- SLD-reduction:
  $[t_1, \ldots, t_i, \ldots, t_n] \rightsquigarrow [\sigma(t_1), \ldots, \sigma(B_0), \ldots, \sigma(B_m), \ldots, \sigma(t_n)]$
  if $\forall \overline{x}.B_1, \ldots, B_m \rightarrow A \in P$, and σ is a unifier of t and A.

Example

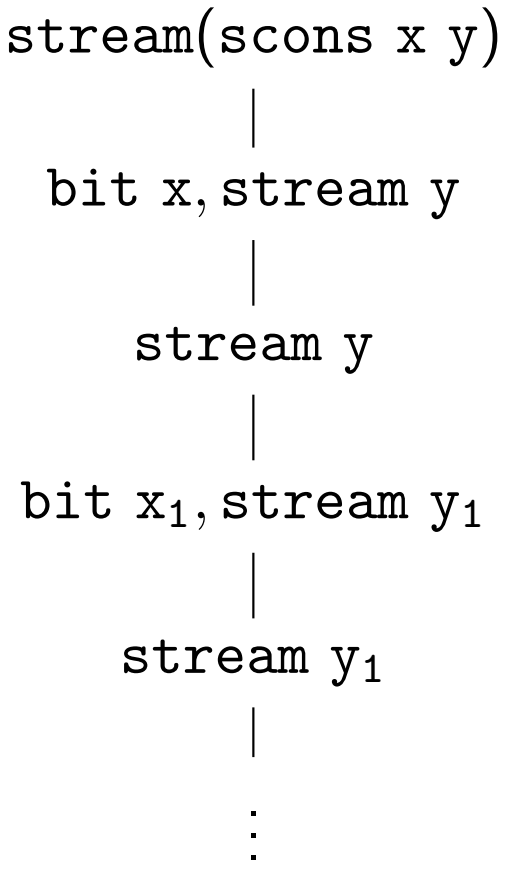
Program **Stream**:

Example

$\texttt{bit}\ \ \texttt{0}$

$\texttt{bit}\ \ \texttt{1}$

$\forall\, x\ y.\ (\texttt{bit}\ \ x \wedge \texttt{stream}\ \ y) \rightarrow \texttt{stream(scons}\ \ x\ \ y\texttt{)}$



So far, we have computed that $x \mapsto 0$, $y \mapsto \texttt{scons}\ x_1 y_1$, and $x_1 \mapsto 0$. At infinity, the term $\texttt{scons}(0\ (\texttt{scons}(0\ \ldots))$ is computed.

Fibrational Coalgebraic Semantics in 3 ideas

Idea 1: Logic programs as coalgebras (propositional case)

1. Let At be the set of all propositions appearing in a program P. Then P can be identified with a $P_f P_f$-coalgebra (At, p), where $p : At \longrightarrow P_f(P_f(At))$ sends an atom A to the set of bodies of those clauses in P with head A.

Example

$Q, R \to T$
$S \to T$
$p(T) = \{\{Q, R\}, \{S\}\}$

Fibrational Coalgebraic Semantics in 3 ideas

Idea 2: Derivations as Comonads

In general, if $U : H\text{-}coalg \longrightarrow C$ has a right adjoint G, the composite functor $UG : C \longrightarrow C$ possesses the canonical structure of a *comonad* $C(H)$, called the *cofree* comonad on H. One can form a *coalgebra* for a comonad $C(H)$.

Fibrational Coalgebraic Semantics in 3 ideas

Idea 2: Derivations as Comonads

In general, if $U : H\text{-}coalg \longrightarrow C$ has a right adjoint G, the composite functor $UG : C \longrightarrow C$ possesses the canonical structure of a *comonad* $C(H)$, called the *cofree* comonad on H. One can form a *coalgebra* for a comonad $C(H)$.

- Taking $p : At \longrightarrow P_f P_f(At)$, the corresponding $C(P_f P_f)$-coalgebra where $C(P_f P_f)$ is the cofree comonad on $P_f P_f$ is given as follows: $C(P_f P_f)(At)$ is given by a limit of the construction

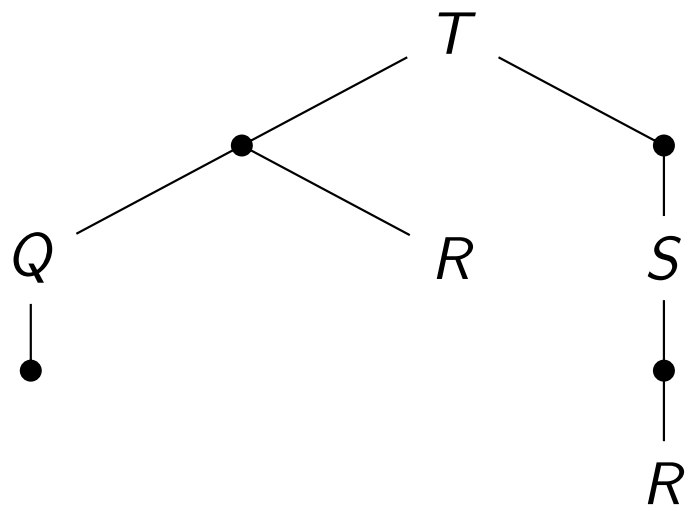
  $$\ldots \longrightarrow At \times P_f P_f(At \times P_f P_f(At)) \longrightarrow At \times P_f P_f(At) \longrightarrow At.$$

  This gives a "tree-like" structure: $\&V$-trees.

Example

Example

$T \leftarrow Q, R$
$T \leftarrow S$
$Q \leftarrow$
$S \leftarrow R$



These are also known as and-or parallel trees

Ekaterina Komendantskaya, Guy McCusker, John Power: Coalgebraic Semantics for Parallel Derivation Strategies in Logic Programming. AMAST 2010: 111-127

Fibrational Coalgebraic Semantics in 3 ideas

Idea 3: Add Lawvere Theories to model first-order signature

A *Lawvere theory* consists of a small category L with strictly associative finite products, and a strict finite-product preserving functor $I : \mathbb{N}^{op} \to L$.

- Take *Lawvere Theory* $\mathcal{L}_\Sigma$ to model the terms over Σ
  - $\mathrm{ob}(\mathcal{L}_\Sigma)$ is $\mathbb{N}$.
  - For each $n \in Nat$, let $x_1, \ldots, x_n$ be a specified list of distinct variables.
  - $\mathrm{ob}(\mathcal{L}_\Sigma)(n, m)$ is the set of m-tuples $(t_1, \ldots, t_m)$ of terms generated by the function symbols in Σ and variables $x_1, \ldots, x_n$.
  - composition in $\mathcal{L}_\Sigma$ is first-order substitution.

Fibrational Coalgebraic Semantics in 3 ideas

Idea 3: Add Lawvere Theories to model first-order signature

A *Lawvere theory* consists of a small category L with strictly associative finite products, and a strict finite-product preserving functor $I : \mathbb{N}^{op} \to L$.

- Take *Lawvere Theory* $\mathcal{L}_\Sigma$ to model the terms over Σ
  - $\mathrm{ob}(\mathcal{L}_\Sigma)$ is $\mathbb{N}$.
  - For each $n \in Nat$, let $x_1, \ldots, x_n$ be a specified list of distinct variables.
  - $\mathrm{ob}(\mathcal{L}_\Sigma)(n, m)$ is the set of m-tuples $(t_1, \ldots, t_m)$ of terms generated by the function symbols in Σ and variables $x_1, \ldots, x_n$.
  - composition in $\mathcal{L}_\Sigma$ is first-order substitution.
- take the functor $At : \mathcal{L}_\Sigma^{op} \to Set$ that sends a natural number n to the set of all atomic formulae generated by Σ and n vars.

Fibrational Coalgebraic Semantics in 3 ideas

Idea 3: Add Lawvere Theories to model first-order signature

A *Lawvere theory* consists of a small category L with strictly associative finite products, and a strict finite-product preserving functor $I : \mathbb{N}^{op} \to L$.

- Take *Lawvere Theory* $\mathcal{L}_\Sigma$ to model the terms over Σ
  - $\mathrm{ob}(\mathcal{L}_\Sigma)$ is $\mathbb{N}$.
  - For each $n \in Nat$, let $x_1, \ldots, x_n$ be a specified list of distinct variables.
  - $\mathrm{ob}(\mathcal{L}_\Sigma)(n, m)$ is the set of m-tuples $(t_1, \ldots, t_m)$ of terms generated by the function symbols in Σ and variables $x_1, \ldots, x_n$.
  - composition in $\mathcal{L}_\Sigma$ is first-order substitution.
- take the functor $At : \mathcal{L}_\Sigma^{op} \to Set$ that sends a natural number n to the set of all atomic formulae generated by Σ and n vars.
- model a program P by the $[\mathcal{L}_\Sigma^{op}, P_f P_f]$-coalgebra $p : At \longrightarrow P_f P_f At$ on the category $[\mathcal{L}_\Sigma^{op}, Set]$.

Further remarks and discussion

Actually, some modifications are needed:

- we need to extend *Set* to *Poset*,
- natural transformations to *lax natural transformations*, and
- replace the outer instance of P_f by P_c - the countable powerset functor.

Further remarks and discussion

Actually, some modifications are needed:

- we need to extend *Set* to *Poset*,
- natural transformations to *lax natural transformations*, and
- replace the outer instance of P_f by P_c - the countable powerset functor.

Then $p : At \longrightarrow P_c P_f At$ gives a $Lax(\mathcal{L}_\Sigma^{op}, P_c P_f)$-coalgebra structure on At.


Filippo Bonchi, Fabio Zanasi: Bialgebraic Semantics for Logic Programming. Logical Methods in Computer Science 11(1) (2015)


Ekaterina Komendantskaya, John Power: Category Theoretic Semantics for Theorem Proving in Logic Programming: Embracing the Laxness. CMCS 2016: 94-113

Examples

Program **Stream**: "fibers" given by term arities. Take the fiber of 1. $\&V$-trees:

Examples

Program **Stream**: "fibers" given by term arities. Take the fiber of 1. $\&V$-trees:

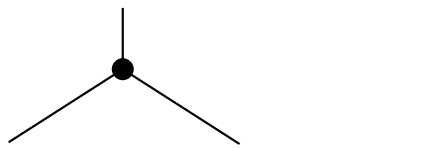
```
stream x
```

Examples

Program **Stream**: "fibers" given by term arities. Take the fiber of 1. $\&V$-trees:

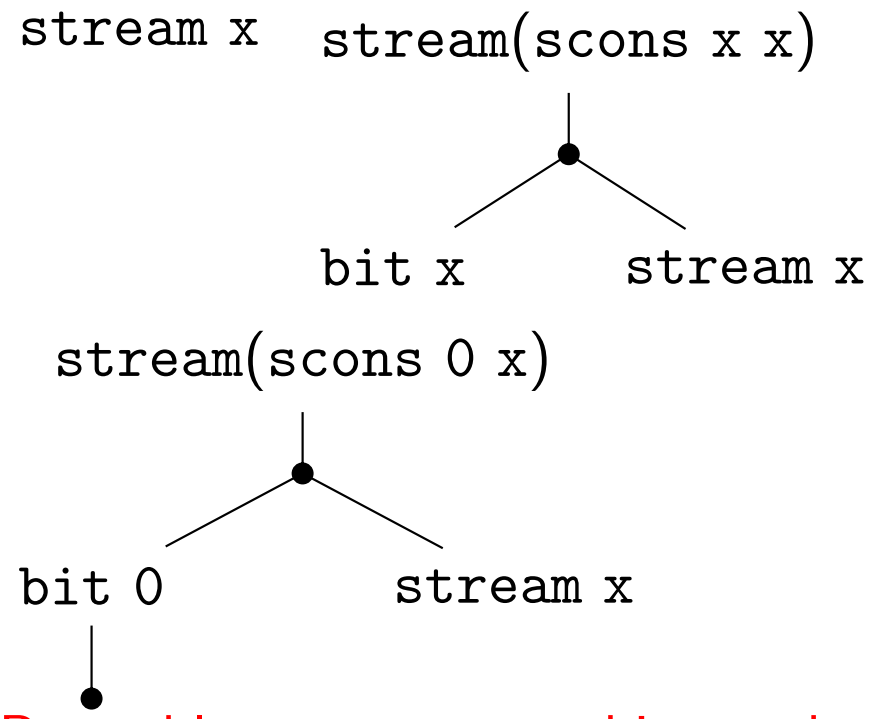
`stream x`

`stream(scons x x)`



Examples

Program **Stream**: "fibers" given by term arities. Take the fiber of 1. $\&V$-trees:



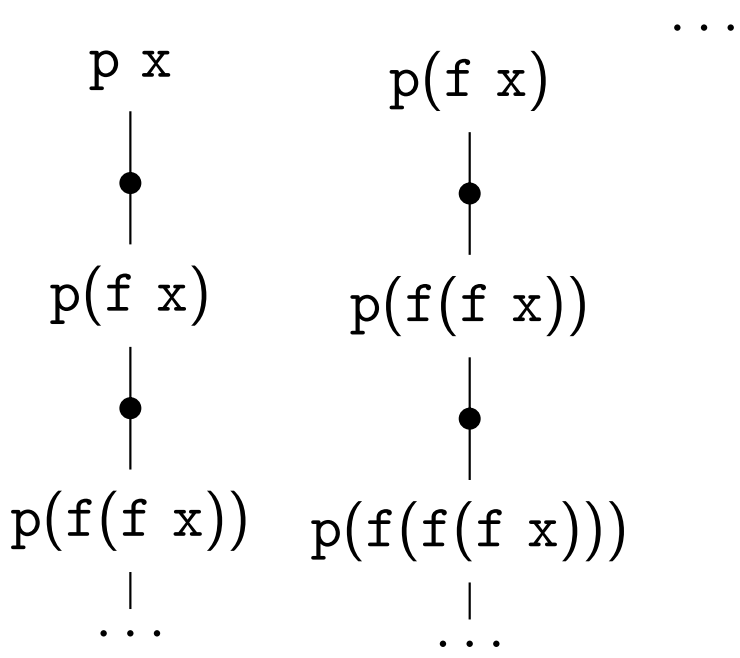
Resembles pattern matching and normalisation, rather than resolution

Examples

Program $\kappa : \forall x.p(f\ x) \to p\ x$: "fibers" given by term arities. Take the fiber of 1. $\&V$-trees:

Examples

Program $\kappa : \forall x.p(f\ x) \to p\ x$: "fibers" given by term arities. Take the fiber of 1. &V-trees:



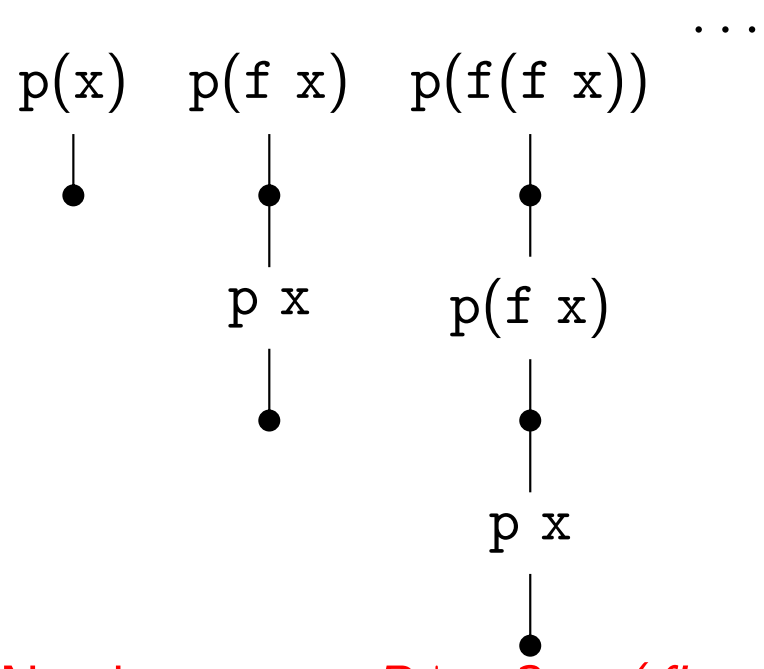
Recall: we know how to prove $P \vdash \nu\alpha.\kappa\alpha : \forall x.p(x)$

Examples

Program $\forall x.p\ x \leftarrow p(f\ x)$: "fibers" given by term arities. Take the fiber of 1. $\&V$-trees:

Examples

Program $\forall x.p\ x \leftarrow p(f\ x)$: "fibers" given by term arities. Take the fiber of 1. $\&V$-trees:



Need to prove $P \vdash e? : p(\mathit{fix}\ x.f\ x)$ – Current work

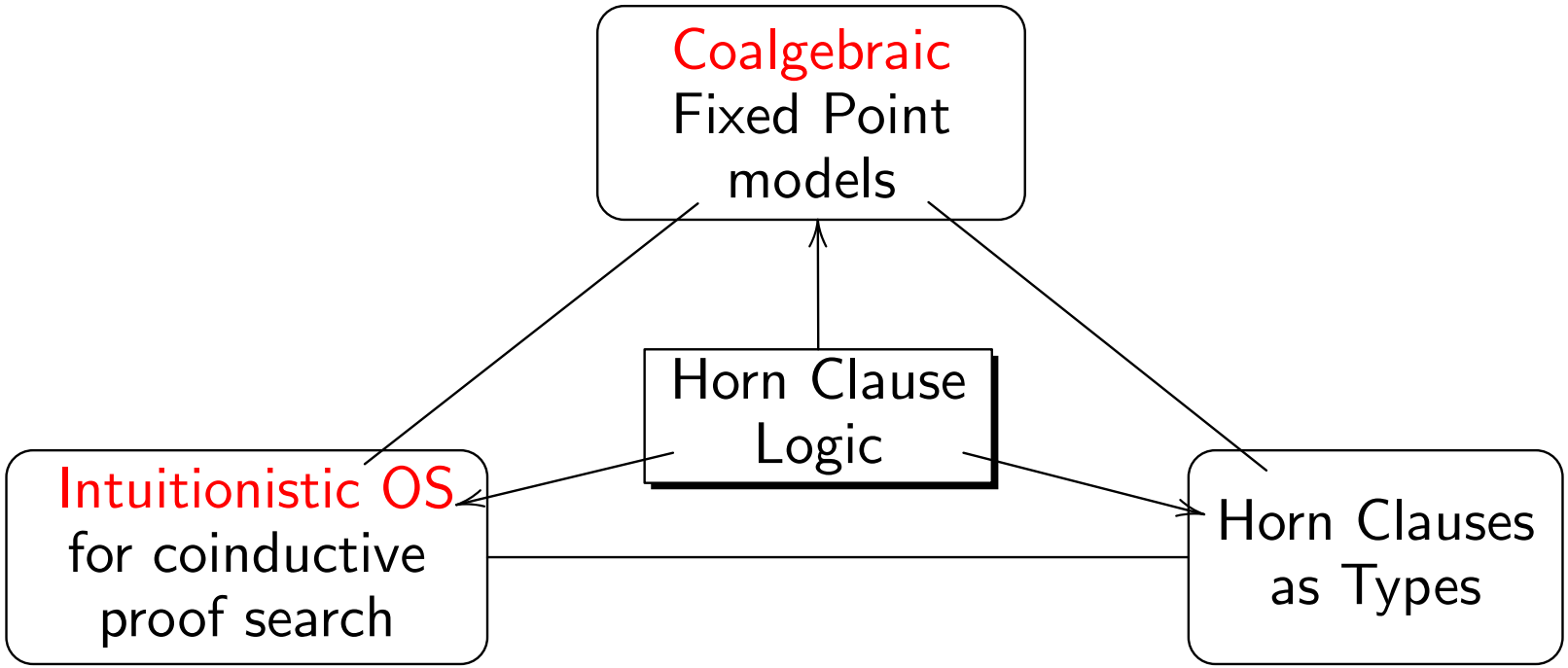
Conclusion: coalgebra for ATP, now!

- Horn clause logic is a special fragment of FOL that provides a *lingua franca* to many Automated Theorem Proving systems: Turing complete and constructive.
- ATP has no reason not to be in Coalgebraic family, and broadly fits into two main coalgebraic directions: coinductive types and transition systems
  - Horn clauses as coinductive types (declaratively)
  - Resolution as transition system (operationally)

Conclusion: coalgebra for ATP, now!

- Horn clause logic is a special fragment of FOL that provides a *lingua franca* to many Automated Theorem Proving systems: Turing complete and constructive.
- ATP has no reason not to be in Coalgebraic family, and broadly fits into two main coalgebraic directions: coinductive types and transition systems
  - Horn clauses as coinductive types (declaratively)
  - Resolution as transition system (operationally)
- It just gets very complicated very quickly for two reasons:
  - Horn clauses as coinductive types are very rich and general, and can define any first-order property or relation. They correspond to dependent types, rather than well-studied simply typed lambda calculus in Lambek-Curry-Howard triangle
  - Study of resolution as transition system is complicated by the need to consider not only non-termination/confluence but also coinductive soundness relative to the models. Unification / existential properties of theories /break of compositionality add to the mix.

What is still missing in the magic triangle?



- Finalise Coalgebraic view on infinite terms and Kleene construction (with H. Basold)
- Formulate a uniform, general, comprehensive operational semantics for constructive and coinductive proof search:
  - involving coinduction both on the level of terms and proofs
  - allowing full generality of coinductive properties (universal, existential, mixed) definable by Horn clause logic