Coalgebraic Derivations in Logic Programming

Ekaterina Komendantskaya, joint work with John Power

Automated Reasoning Workshop, Glasgow, 2011

ARW'11,
11 April 2011

Recursion and Corecursion in Logic Programming

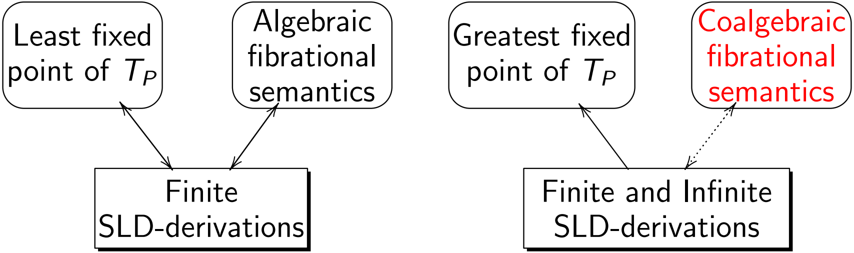
Example

$$
\begin{aligned}
\mathtt{nat(0)} &\leftarrow \\
\mathtt{nat(s(x))} &\leftarrow \mathtt{nat(x)} \\
\mathtt{list(nil)} &\leftarrow \\
\mathtt{list(cons\ x\ y)} &\leftarrow \mathtt{nat(x),\ list(y)}
\end{aligned}
$$

Example

$$
\begin{aligned}
\mathtt{bit}(0) &\leftarrow \\
\mathtt{bit}(1) &\leftarrow \\
\mathtt{stream(cons\ (x,y))} &\leftarrow \mathtt{bit(x)}, \mathtt{stream(y)}
\end{aligned}
$$

Algebraic and coalgebraic semantics for LP



In one slide,

It is the sory of how one started with looking for a suitable semantics for an existing derivation algorithm, and ended up proposing a new derivation algorithm for the semantics.

Part 1

Colagebraic semantics for Logic programming.

Coalgebraic Analysis of Logic Programs

Generally, given a functor F, an F-coalgebra is a pair (S, α) consisting of a set S and a function $\alpha : S \longrightarrow F(S)$. We will take a powerset functor P_f.

Proposition

For any set At, there is a bijection between the set of variable-free logic programs over the set of atoms At and the set of $P_f P_f$-coalgebra structures on At.

Proof.

Given a variable-free logic program P, let At be the set of all atoms appearing in P. Then P can be identified with a $P_f P_f$-coalgebra (At, p), where $p : At \longrightarrow P_f(P_f(\mathrm{At}))$ sends an atom A to the set of bodies of those clauses in P with head A, each body being viewed as the set of atoms that appear in it. $\square$

Example

Example

Consider the logic programbelow .

$$
\begin{array}{rcl}
\mathtt{q(b,a)} & \leftarrow & \mathtt{s(a,b)} \\
\mathtt{q(b,a)} & \leftarrow & \\
\mathtt{s(a,b)} & \leftarrow & \\
\mathtt{p(a)} & \leftarrow & \mathtt{q(b,a)}, \mathtt{s(a,b)}
\end{array}
$$

The program has three atoms, namely $\mathtt{q(b,a)}$, $\mathtt{s(a,b)}$ and $\mathtt{p(a)}$. So $At = \{\mathtt{q(b,a)}, \mathtt{s(a,b)}, \mathtt{p(a)}\}$. And the program can be identified with the $P_f P_f$-coalgebra structure on At given by
$p(\mathtt{q(b,a)}) = \{\{\}, \{\mathtt{s(a,b)}\}\}$, where $\{\}$ is the empty set.
$p(\mathtt{s(a,b)}) = \{\{\}\}$, i.e., the one element set consisting of the empty set.
$p(\mathtt{p(a)}) = \{\{\mathtt{q(b,a)}, \mathtt{s(a,b)}\}\}$.

Right adjoint

Definition

Given two categories $\mathcal{C}$ and $\mathcal{D}$, the functor $U : \mathcal{C} \to \mathcal{D}$ *has a right adjoint* if for all $A \in \mathcal{D}$ there exists $GA \in \mathcal{C}$ and there exists $\epsilon_A : UGA \to A$ such that for all $B \in \mathcal{C}$ and for all $f : UB \to A$ there exists a unique $g : B \to GA$ such that the following diagram commutes:

$$\begin{tikzcd}
\mathrm{UGA} \arrow[r, "\epsilon_A"] & \mathrm{A} & \mathrm{GA} \\
\mathrm{UB} \arrow[u, "Ug"] \arrow[ur, "f"'] & & \mathrm{B} \arrow[u, dashed, "g"']
\end{tikzcd}$$

Coalgebraic Analysis of derivations in Logic Programs

Theorem

Given an endofunctor $H : Set \longrightarrow Set$ with a rank, the forgetful functor $U : H\text{-}Coalg \longrightarrow Set$ has a right adjoint R.

R is constructed as follows. Given $Y \in Set$, we define a transfinite sequence of objects as follows. Put $Y_0 = Y$, and $Y_{\alpha+1} = Y \times H(Y_\alpha)$. We define $\delta_\alpha : Y_{n+1} \longrightarrow Y_n$ inductively by

$$Y_{\alpha+1} = Y \times HY_\alpha \stackrel{Y \times H\delta_{\alpha-1}}{\longrightarrow} Y \times HY_{\alpha-1} = Y_\alpha,$$

with the case of $\alpha = 0$ given by the map $Y_1 = Y \times HY \stackrel{\pi_1}{\longrightarrow} Y$. For a limit ordinal, let $Y_\alpha = \lim_{\beta<\alpha}(Y_\beta)$, determined by the sequence

$$Y_{\beta+1} \stackrel{\delta_\beta}{\longrightarrow} Y_\beta.$$

If H has a rank, there exists α such that Y_α is isomorphic to $Y \times HY_\alpha$. This Y_α forms the cofree coalgebra on Y.

Coalgebraic Analysis of derivations in Logic Programs

$$U: \text{H-Coalg} \underset{}{\overset{R}{\underset{\longrightarrow}{\longleftarrow}}} \text{Set}$$

Corollary

If H has a rank, U has a right adjoint R and putting $G = RU$, G possesses a canonical comonad structure and there is a coherent isomorphism of categories

$$G\text{-}Coalg \cong H\text{-}Coalg,$$

where $G\text{-}Coalg$ is the category of G-coalgebras for the comonad G.

Given an H-coalgebra $p : Y \longrightarrow HY$, we construct maps $p_\alpha : Y \longrightarrow Y_\alpha$ for each ordinal α as follows. The map $p_0 : Y \longrightarrow Y$ is the identity, and for a successor ordinal, $p_{\alpha+1} = \langle id, Hp_\alpha \circ p\rangle : Y \longrightarrow Y \times HY_\alpha$. For limit ordinals, p_α is given by the appropriate limit. By definition, the object GY is given by Y_α for some α, and the corresponding p_α is the required G-coalgebra.

Coalgebraic Analysis of derivations in Logic Programs

Taking $p : \mathrm{At} \longrightarrow P_f P_f(\mathrm{At})$, by the proof of Theorem 1, 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: $C(P_f P_f)(\mathrm{At})$ is given by a limit of the form

$$\ldots \longrightarrow \mathrm{At} \times P_f P_f(\mathrm{At} \times P_f P_f(\mathrm{At})) \longrightarrow \mathrm{At} \times P_f P_f(\mathrm{At}) \longrightarrow \mathrm{At}.$$

This chain has length ω. As above, we inductively define the objects $\mathrm{At}_0 = \mathrm{At}$ and $\mathrm{At}_{n+1} = \mathrm{At} \times P_f P_f \mathrm{At}_n$, and the cone

$$\begin{aligned} p_0 &= id : \mathrm{At} \longrightarrow \mathrm{At}(= \mathrm{At}_0) \\ p_{n+1} &= \langle id, P_f P_f(p_n) \circ p \rangle : \mathrm{At} \longrightarrow \mathrm{At} \times P_f P_f \mathrm{At}_n(= \mathrm{At}_{n+1}) \end{aligned}$$

and the limit determines the required coalgebra $\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$.

Success!

We prove soundness and completeness results for the SLD-derivations relative to the Coalgebraic semantics.

However, the observational semantics does not come naturally to this kind of derivations.

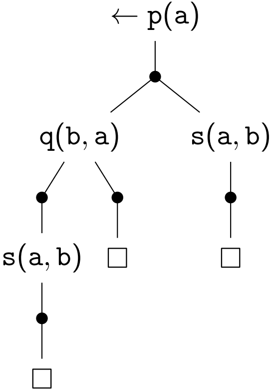
One of the main purposes of giving an observational semantics to logic programs is its ability to observe equal behaviors of logic programs and distinguish logic programs with different computational behavior. Therefore, the choice of observables and semantic models is closely related to the choice of equivalence relation defined over logic programs.

Part 2

Coalgebraic derivations in Logic programming.

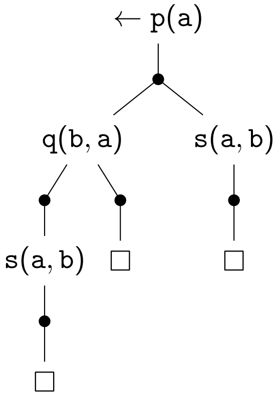
Examples of a derivations

The action of
$\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on
p(a)



Examples of a derivations

The action of
$\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on $\mathtt{p(a)}$

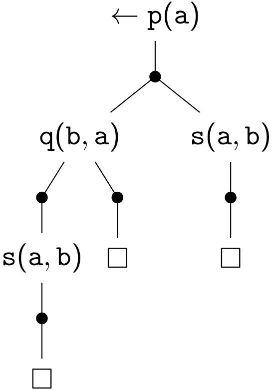


The SLD derivation

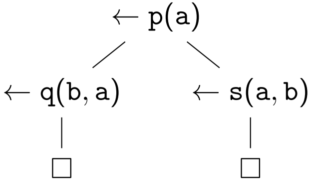
$$\leftarrow \mathtt{p(a)}$$
$$\leftarrow \mathtt{q(b,a)}, \mathtt{s(a,b)}$$
$$\leftarrow \mathtt{s(a,b)}, \mathtt{s(a,b)}$$
$$\leftarrow \mathtt{s(a,b)}$$
$$\Box$$

Examples of a derivations

The action of
$\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on
$\mathtt{p(a)}$

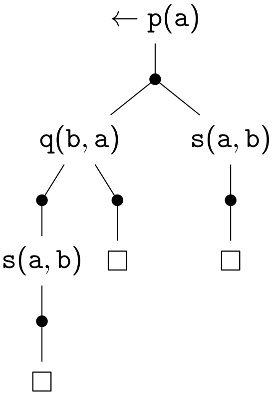


The proof tree

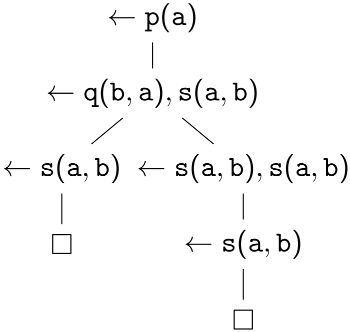


Examples of a derivations

The action of $\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on p(a)



The SLD tree



Is there anything at all in practice of Logic Programming that corresponds to the action of $C(P_f P_f)$-comonad?

From the examples above, it's clear that:

Sequential SLD-derivation

is the least suitable for the model given by $C(P_f P_f)$-comonad.

Proof trees

exhibit an and-parallelism in derivations - that is, parallel proof search over conjuncts in a goal, but the choices of different clauses to use in the process are not reflected - except for - one can use a sequence of proof-trees for this purpose.

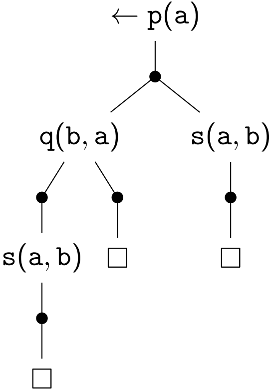
SLD-trees

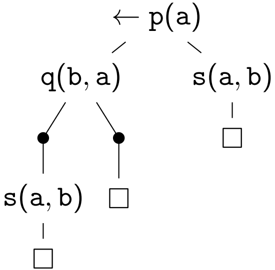
exhibit an or-parallelism in derivations - that is, they show different possibilities of derivations if there are multiple clauses that unify with a goal; but they process conjuncts in a goal sequentially.

It turns out that the answer lies in the combination of the two kinds of parallelism:

$\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on p(a)

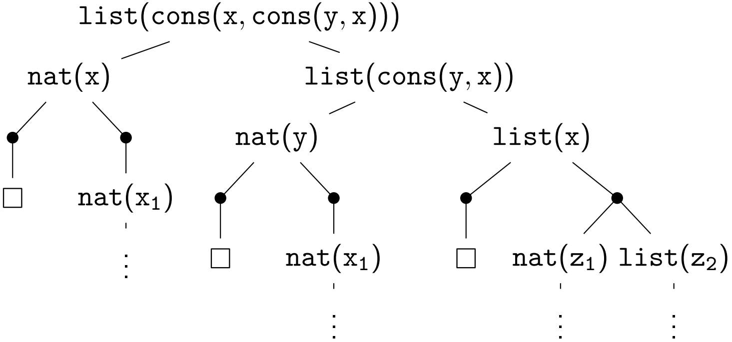
The and-or parallel tree





Except for... and-or trees are unsound in the first-order case.

Why unsound?


list(cons(x, cons(y, x)))
nat(x)
list(cons(y, x))
□
nat(x1)
nat(y)
list(x)
□
nat(x1)
□
nat(z1)
list(z2)

Coalgebraic semantics for the first-order case

- We use *Lawvere theories*,
- model most general unifiers (mgu's) by *equalisers*,

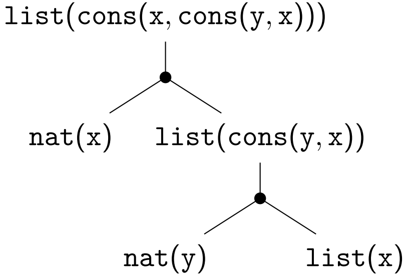
Given a signature Σ, the Lawvere theory $\mathcal{L}_\Sigma$ generated by Σ

has objects given by natural numbers and maps from n to m given by equivalence classes of substitutions θ of m variables by terms generated by the function symbols in Σ applied to n variables.

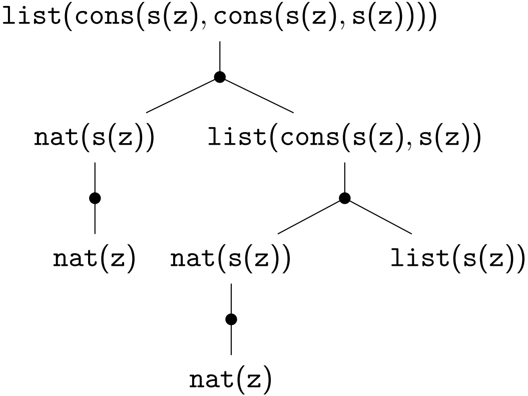
We would like to model P by the putative $[\mathcal{L}_\Sigma^{op}, P_f P_f]$-coalgebra $p : At \longrightarrow P_f P_f At$ that, at n, takes an atomic formula $A(x_1, \ldots, x_n)$ with at most n variables, considers all substitutions of clauses in P whose head agrees with $A(x_1, \ldots, x_n)$, and gives the set of sets of atomic formulae in antecedents.

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

Example of a first-order coinductive tree (Sound derivations!):



Example of a first-order coinductive tree (Sound derivations!):



The main results

- We propose a new coinductive derivation algorithm inspired by the coalgebraic semantics.
- The algorithm provides an elegant solution to the problem of implementing both corecursion and concurrency in logic programming.
- We prove soundness and completeness,
- ... and correctness and full abstraction results for the new coinductive derivations relative to the coalgebraic semantics.

Thank you!