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

\[
\begin{align*}
\text{nat}(0) & \leftarrow \\
\text{nat}(s(x)) & \leftarrow \text{nat}(x) \\
\text{list}(\text{nil}) & \leftarrow \\
\text{list}(\text{cons } x \; y) & \leftarrow \text{nat}(x), \text{list}(y)
\end{align*}
\]

Example

\[
\begin{align*}
\text{bit}(0) \leftarrow \\
\text{bit}(1) \leftarrow \\
\text{stream}(\text{cons } (x,y)) \leftarrow \text{bit}(x), \text{stream}(y)
\end{align*}
\]
Algebraic and coalgebraic semantics for LP

Least fixed point of $T_P$

Algebraic fibrational semantics

Finite SLD-derivations

Greatest fixed point of $T_P$

Coalgebraic fibrational semantics

Finite and Infinite SLD-derivations
In one slide,

It is the story 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.
Colagebraic semantics for Logic programming.
Coalgebraic Analysis of Logic Programs

Generally, given a functor $F$, an $F$-coalgebra is a pair $(S, \alpha)$ consisting of a set $S$ and a function $\alpha : S \rightarrow F(S)$. 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 $P_fP_f$-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 $P_fP_f$-coalgebra $(At, p)$, where $p : At \rightarrow P_f(P_f(At))$ 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.
Consider the logic program below.

\[
\begin{align*}
q(b,a) & \leftarrow s(a,b) \\
q(b,a) & \leftarrow \\
s(a,b) & \leftarrow \\
p(a) & \leftarrow q(b,a), s(a,b)
\end{align*}
\]

The program has three atoms, namely \(q(b,a)\), \(s(a,b)\) and \(p(a)\). So \(At = \{q(b,a), s(a,b), p(a)\}\). And the program can be identified with the \(P_fP_f\)-coalgebra structure on \(At\) given by

\[
\begin{align*}
p(q(b,a)) &= \{\{\}, \{s(a,b)\}\}, \text{ where } \{\} \text{ is the empty set.} \\
p(s(a,b)) &= \{\{\}\}, \text{ i.e., the one element set consisting of the empty set.} \\
p(p(a)) &= \{\{q(b,a), s(a,b)\}\}.
\end{align*}
\]
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:

\[
\begin{array}{ccc}
UGA & \xrightarrow{\epsilon_A} & A \\
\downarrow{Ug} & & \downarrow{f} \\
UB & \xrightarrow{g} & GA \\
\end{array}
\]
Coalgebraic Analysis of derivations in Logic Programs

**Theorem**

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

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

$$Y_{\alpha+1} = Y \times H Y_\alpha \xrightarrow{Y \times H \delta_{\alpha-1}} Y \times H Y_{\alpha-1} = Y_\alpha,$$

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

$$Y_{\beta+1} \xrightarrow{\delta_\beta} Y_\beta.$$

If $H$ has a rank, there exists $\alpha$ such that $Y_\alpha$ is isomorphic to $Y \times H Y_\alpha$. This $Y_\alpha$ forms the cofree coalgebra on $Y$. 
Coalgebraic Analysis of derivations in Logic Programs

\[ \text{Corollary} \]

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 \( p : Y \rightarrow HY \), we construct maps \( p_\alpha : Y \rightarrow Y_\alpha \) for each ordinal \( \alpha \) as follows. The map \( p_0 : Y \rightarrow Y \) is the identity, and for a successor ordinal, \( p_{\alpha+1} = \langle id, Hp_\alpha \circ p \rangle : Y \rightarrow Y \times HY_\alpha \). For limit ordinals, \( p_\alpha \) is given by the appropriate limit. By definition, the object \( GY \) is given by \( Y_\alpha \) for some \( \alpha \), and the corresponding \( p_\alpha \) is the required \( G \)-coalgebra.
Taking $p : \text{At} \rightarrow P_f P_f(\text{At})$, 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: $C(P_f P_f)(\text{At})$ is given by a limit of the form

$$\ldots \rightarrow \text{At} \times P_f P_f(\text{At} \times P_f P_f(\text{At})) \rightarrow \text{At} \times P_f P_f(\text{At}) \rightarrow \text{At}.$$ 

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

$$p_0 = \text{id} : \text{At} \rightarrow \text{At}(= \text{At}_0)$$

$$p_{n+1} = \langle \text{id}, P_f P_f(p_n) \circ p \rangle : \text{At} \rightarrow \text{At} \times P_f P_f \text{At}_n(= \text{At}_{n+1})$$

and the limit determines the required coalgebra $\overline{p} : \text{At} \rightarrow C(P_f P_f)(\text{At})$.
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 $\bar{p} : At \rightarrow C(P_f P_f)(At)$ on $p(a)$
Examples of derivations

The action of $\bar{p} : \text{At} \rightarrow C(P_f P_f)(\text{At})$ on $p(a)$

The SLD derivation

\[ \leftarrow p(a) \]
\[ \leftarrow q(b, a), s(a, b) \]
\[ \leftarrow s(a, b), s(a, b) \]
\[ \leftarrow s(a, b) \]
Examples of a derivations

The action of $\bar{p} : \text{At} \rightarrow C(P_f P_f)(\text{At})$ on $p(a)$

$$p(a) \leftarrow q(b, a) \leftarrow s(a, b)$$

The proof tree

$$\leftarrow p(a)$$

$$\leftarrow q(b, a)$$

$$\leftarrow s(a, b)$$
Examples of a derivations

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

\[
\begin{align*}
&\leftarrow p(a) \\
&\quad \leftarrow q(b, a), s(a, b) \\
&\quad \quad \leftarrow s(a, b) \\
&\quad \quad \quad \leftarrow s(a, b)
\end{align*}
\]

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

From the examples above, it’s clear that:

**Sequential SLD-derivation**

is the least suitable for the model given by $C(P_fP_f)$-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:

\[ \overline{p} : \text{At} \longrightarrow C(P_f P_f)(\text{At}) \text{ on } p(a) \]

The and-or parallel tree

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

\[
\text{list(cons(x, cons(y, x)))}
\]

\[
\text{nat(x)} \quad \text{list(cons(y, x))}
\]

\[
\text{nat(x_1)} \quad \text{nat(y)} \quad \text{list(x)}
\]

\[
\text{nat(x_1)} \quad \text{nat(x_1)} \quad \text{nat(z_1) list(z_2)}
\]

\[
\text{...} \quad \text{...} \quad \text{...}
\]
Coalgebraic semantics for the first-order case

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

Given a signature $\Sigma$, the Lawvere theory $L_{\Sigma}$ generated by $\Sigma$

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

We would like to model $P$ by the putative $[L_{\Sigma}^{op}, P_fP_f]$-coalgebra $p : At \rightarrow P_fP_fAt$ 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.

$p : At \rightarrow P_cP_fAt$ gives a $Lax(L_{\Sigma}^{op}, P_cP_f)$-coalgebra structure on $At$; and $p$ determines a $Lax(L_{\Sigma}^{op}, C(P_cP_f))$-coalgebra structure $\bar{p} : At \rightarrow C(P_cP_f)(At)$. 
Example of a first-order coinductive tree (Sound derivations!):

```
list(cons(x, cons(y, x)))
```

```
  nat(x)  list(cons(y, x))
```

```
    nat(y)  list(x)
```

Katya (ARW'11)
Coalgebraic Derivations in Logic Programming
ARW'11 21 / 24
Example of a first-order coinductive tree (Sound derivations!):

```
list(cons(s(z), cons(s(z), s(z)))))
```

Diagram:

```
  list(cons(s(z), cons(s(z), s(z)))))
   /         \                       /
  nat(s(z))  list(cons(s(z), s(z)))/  \               /
     \                                /       /
    nat(z)  nat(s(z))  list(s(z))   /       /
       \                                /         
        nat(z)                         }
```
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!