Coalgebraic Semantics for Parallel Derivation Strategies in Logic Programming.

Ekaterina Komendantskaya, joint work with John Power and Guy McCusker

13th International Conference on Algebraic Methodology and Software Technology

AMAST10,
24 June 2010

Outline

Outline

Outline

Outline

Logic programs

A first-order logic program consists of a finite set of clauses of the form

$$A \leftarrow A_1, \ldots, A_n$$

where A and the A_i's are atomic formulae, typically containing free variables; and $A_1, \ldots, A_n$ is to mean the conjunction of the A_i's.

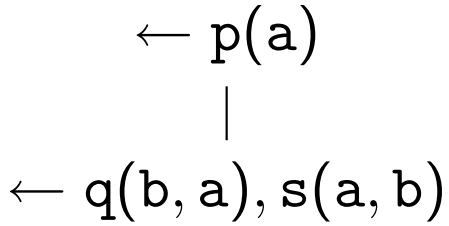
Definition

Let a goal G be $\leftarrow A_1, \ldots, A_m, \ldots, A_k$ and a clause C be $A \leftarrow B_1, \ldots, B_q$. Then G' is *derived* from G and C using mgu θ if the following conditions hold:

- A_m is an atom, called the *selected* atom, in G.
- θ is an *mgu* of A_m and A.
- G' is the goal $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta$.

Example: Goal $\leftarrow$ p(a).

```
q(b,a)  ←  s(a,b)
q(b,a)  ←
s(a,b)  ←
  p(a)  ←  q(b,a),s(a,b)
```



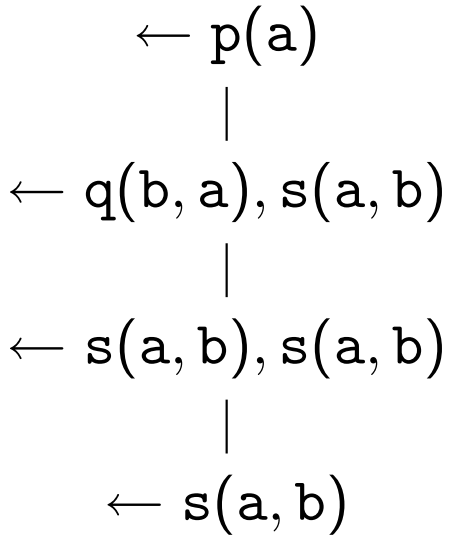
Example: Goal $\leftarrow$ p(a).

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

$$
\begin{array}{c}
\leftarrow \texttt{p(a)} \\
| \\
\leftarrow \texttt{q(b, a), s(a, b)} \\
| \\
\leftarrow \texttt{s(a, b), s(a, b)}
\end{array}
$$

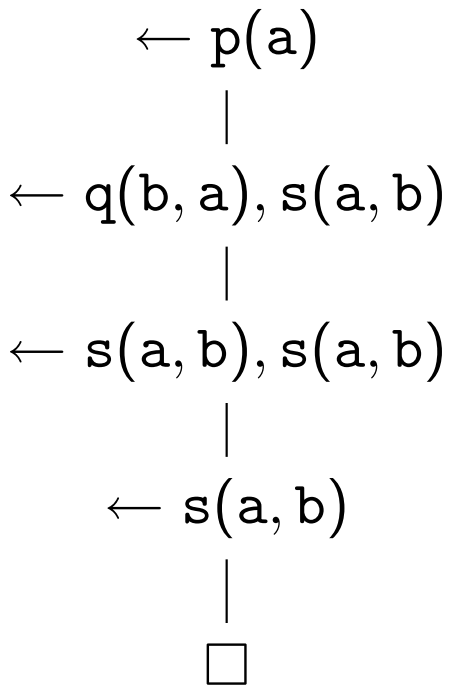
Example: Goal $\leftarrow$ p(a).

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

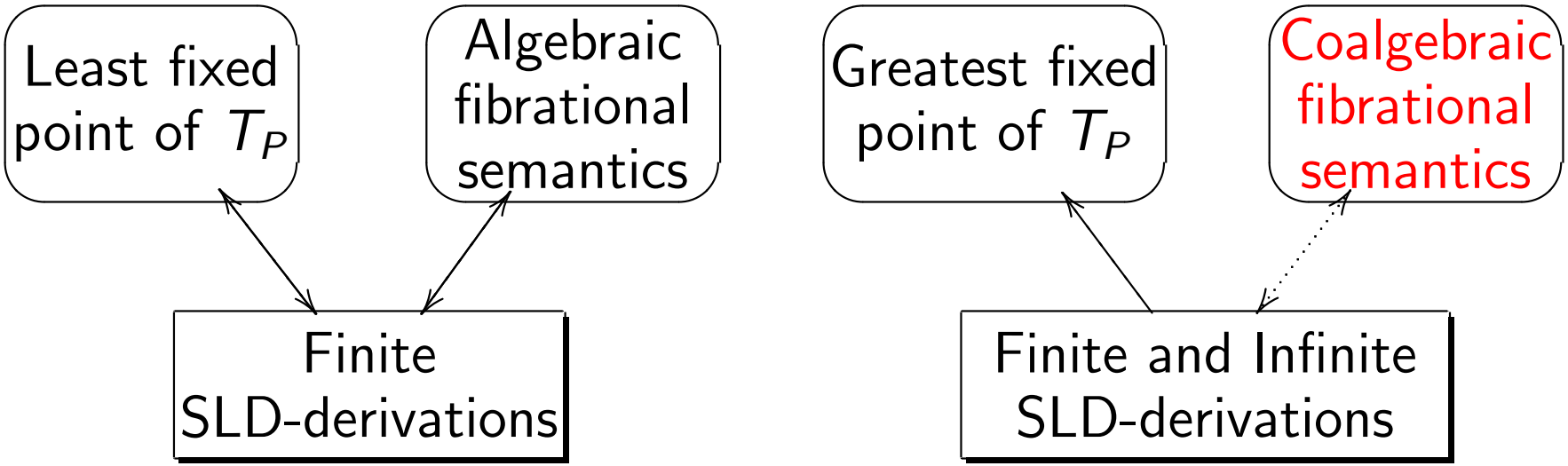


Example: Goal $\leftarrow$ p(a).

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



Algebraic and coalgebraic semantics for LP



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. □

Example

Example

Consider the logic program from the previous Example.

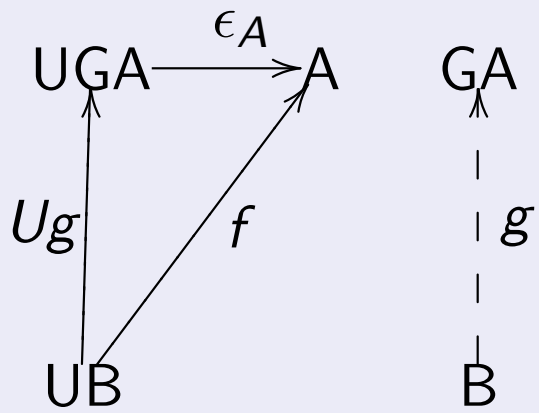
$$\begin{aligned} \texttt{q(b,a)} &\leftarrow \texttt{s(a,b)} \\ \texttt{q(b,a)} &\leftarrow \\ \texttt{s(a,b)} &\leftarrow \\ \texttt{p(a)} &\leftarrow \texttt{q(b,a)}, \texttt{s(a,b)} \end{aligned}$$

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

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:



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

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

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

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

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

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

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-$Coalg$ 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

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

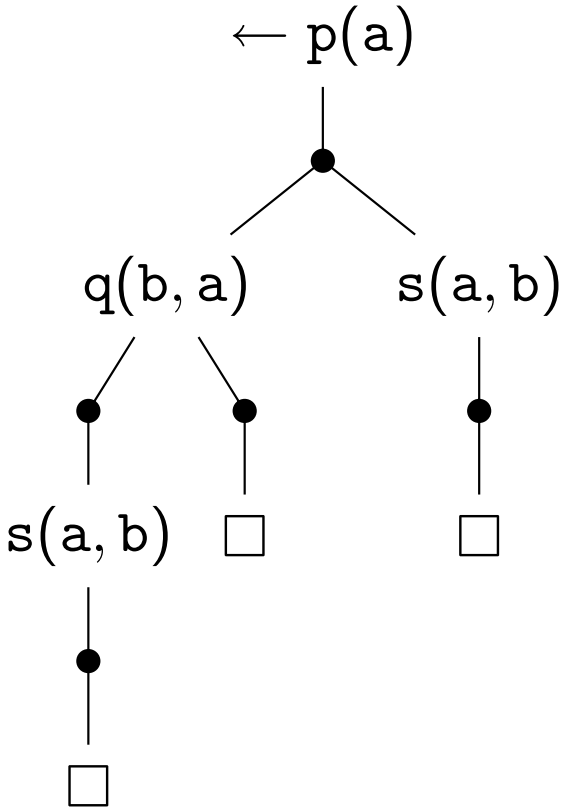
This chain has length ω. As above, we inductively define the objects $\mathrm{At}_0 = \mathrm{At}$ and $\mathrm{At}_{n+1} = \mathrm{At} \times P_f P_f \mathrm{At}_n$, and the cone

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

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

Examples of a derivations

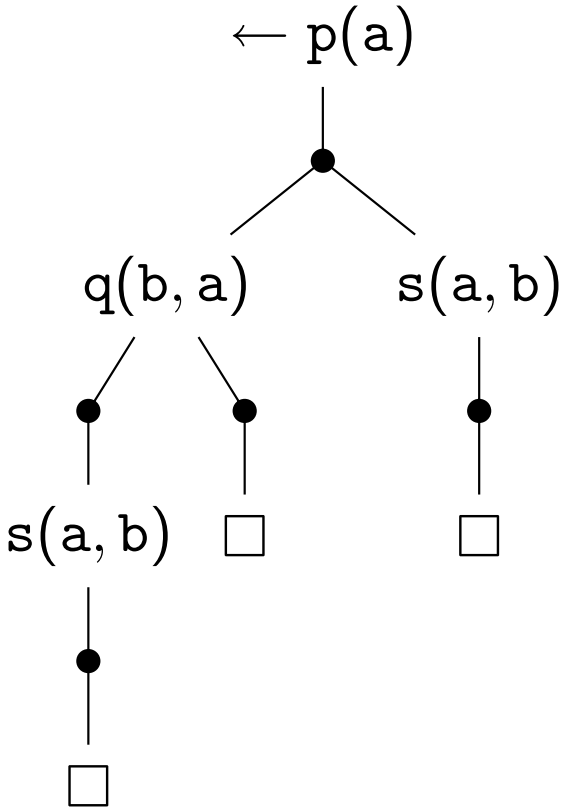
The action of
$\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on
$\mathtt{p(a)}$

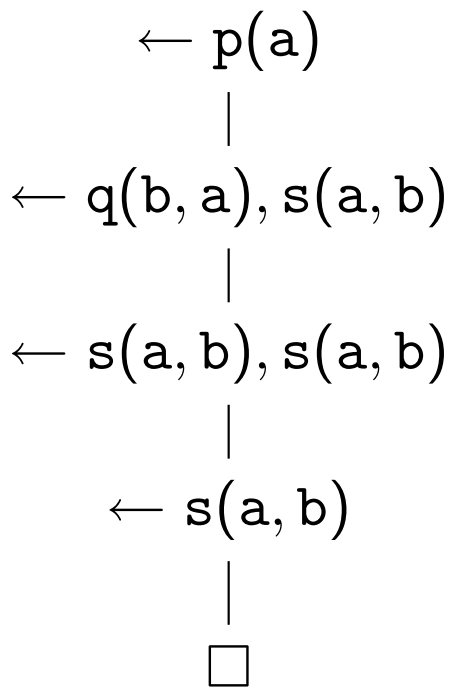


Examples of a derivations

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

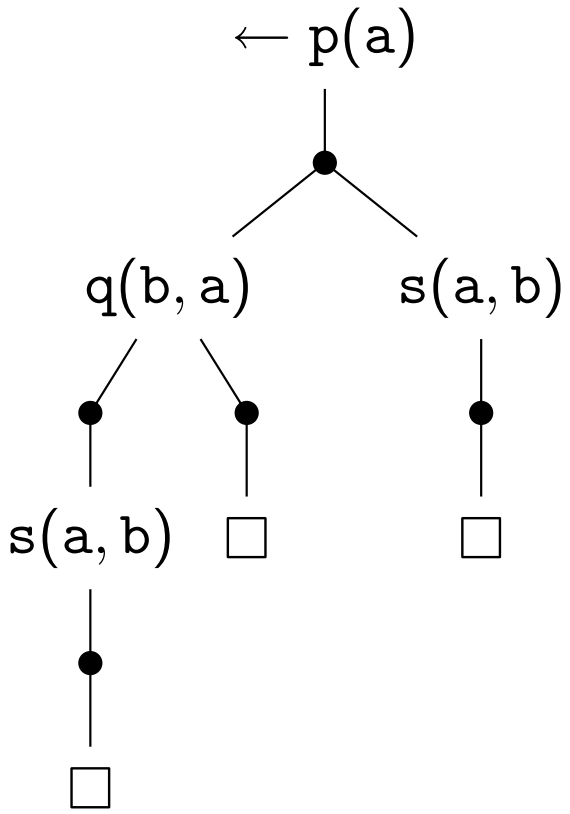
The SLD derivation from a previous example



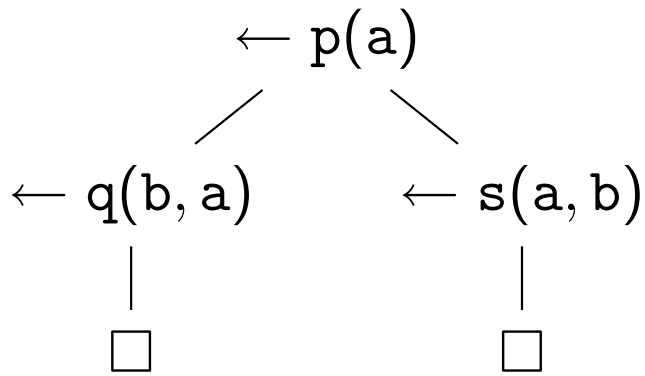


Examples of a derivations

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



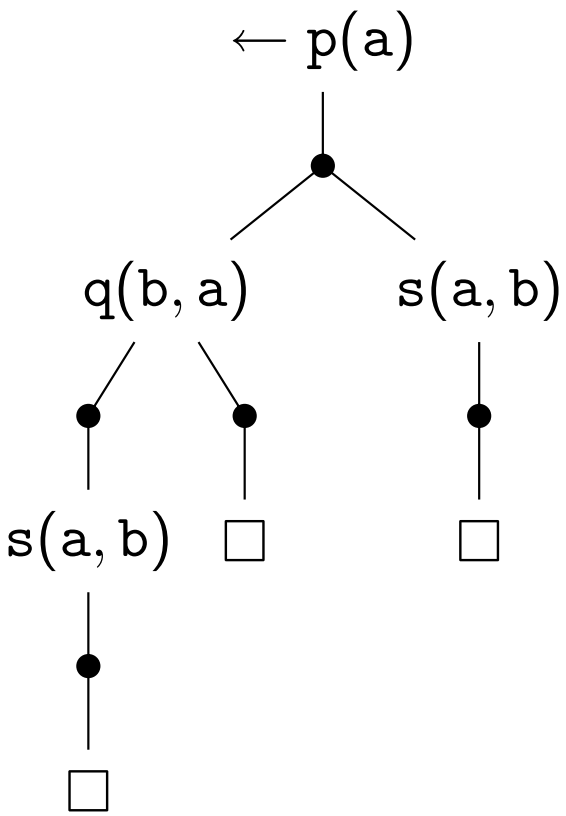
The proof tree

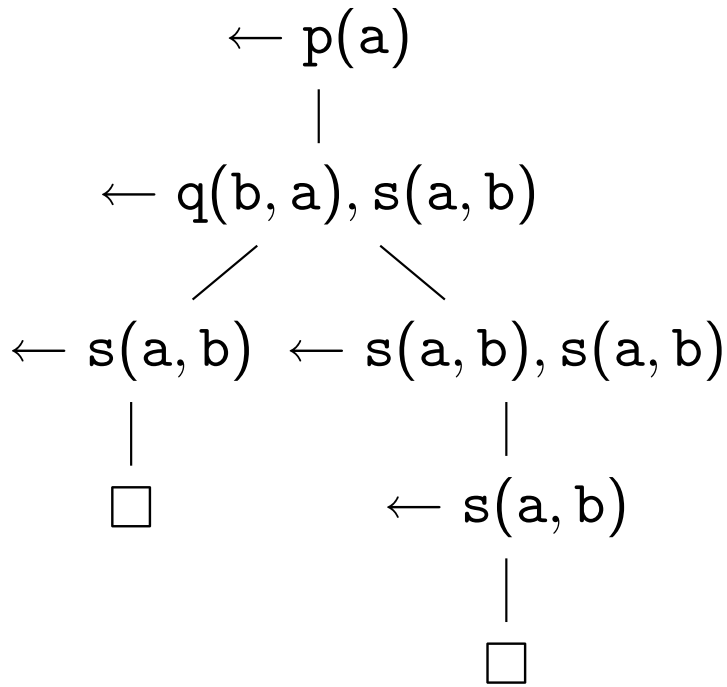


Examples of a derivations

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

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.

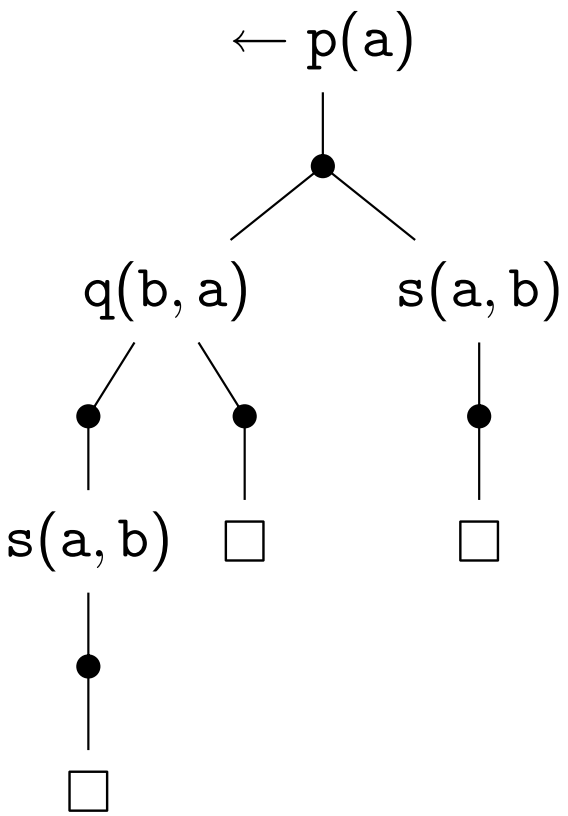
SLD-trees

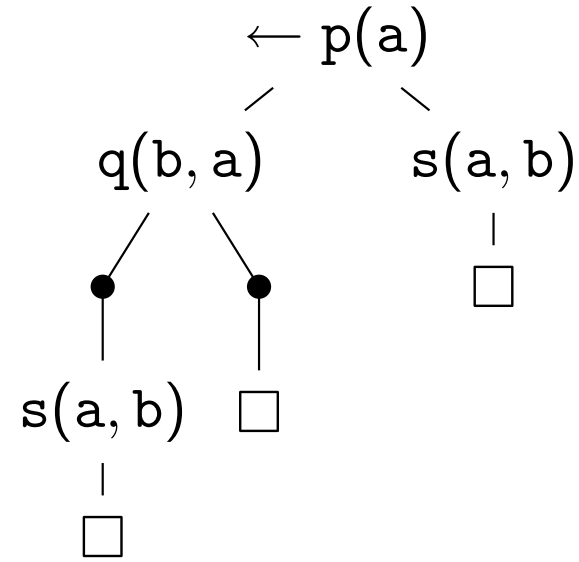
exhibit an or-parallelism in derivations - that is, they show different possibilities of derivations if there are multiple clauses that unify with a goal; but they process conjuncts in a goal sequentially.

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... the order and repetitions of branches is important for and-or trees.

$P_f P_f$ and $P_f List$-coalgebras.

There are different approaches to the meaning of goals and bodies given by $A, A_1, \ldots A_n$. One approach arises from first-order logic semantics, and treats them as finite conjunctions, in which case the order of atoms is not important, and repetitions of atoms are not considered. Another — practical — approach is to treat bodies as sequences of atoms, in which case repetitions and order can play a role in computations.

Coalgebraic semantics based on $P_f P_f$-coalgebras will identify clauses
$B_1 \leftarrow A_1, A_2, A_3$ and
$B_1 \leftarrow A_3, A_2, A_1$ and
$B_1 \leftarrow A_1, A_2, A_3, A_1$ and
$B_1 \leftarrow A_3, A_2, A_1, A_1$.
$C(P_f P_f)$ coalgebras will identify derivations that involve such clauses.

$P_f P_f$ and $P_f List$-coalgebras.

There are different approaches to the meaning of goals and bodies given by $A, A_1, \ldots A_n$. One approach arises from first-order logic semantics, and treats them as finite conjunctions, in which case the order of atoms is not important, and repetitions of atoms are not considered. Another — practical — approach is to treat bodies as sequences of atoms, in which case repetitions and order can play a role in computations.

Coalgebraic semantics based on $P_f P_f$-coalgebras will identify clauses
$B_1 \leftarrow A_1, A_2, A_3$ and
$B_1 \leftarrow A_3, A_2, A_1$ and
$B_1 \leftarrow A_1, A_2, A_3, A_1$ and
$B_1 \leftarrow A_3, A_2, A_1, A_1$.
$C(P_f P_f)$ coalgebras will identify derivations that involve such clauses.

From the point of view of practice of logic programming, it is much more natural to give coalgebraic semantics in terms of $P_f List$-coalgebras, rather than $P_f P_f$-coalgebras.

Soundness and completeness

Theorem (Soundness and completeness)

Let P be a variable-free logic program, At the set of all atoms appearing in P, and $\bar{p} : At \longrightarrow C(P_f\,List)(At)$ the $C(P_f\,List)$-coalgebra generated by P. (Recall that $\bar{p}$ is constructed as a limit of a cone p_n over an ω-chain.) Then, for any atom A, $\bar{p}(A)$ expresses precisely the same information as that given by the parallel and-or derivation tree for A, that is, the following holds:

- *For a derivation step n of the parallel and-or tree for A, $p_n(A)$ is isomorphic to the and-or parallel tree for A of depth n.*
- *The and-or tree for A is of finite size and has the depth n iff $\bar{p}(A) = p_n(A)$.*
- *The and-or tree for A is infinite iff $\bar{p}(A)$ is given by the element of the limit $\lim_\omega(p_n)(At)$ of an infinite chain given by Construction of $\bar{p} : At \longrightarrow C(P_f\,List)(At)$.*

The Theory of Observables

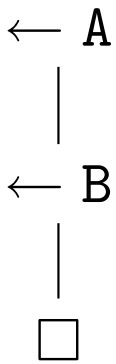
The distinction between the use of P_f and *List* has a deep connection to the Theory of Observables for logic programming, introduced by [Comini, Levi, Meo, 1995-2009].

Traditional model-theoretic (fixed point) semantics for logic programming treats atoms in the bodies as conjunctions - hence their order and repetitions play no role in the semantics. As a result, the traditional semantics identifies programs that behave very differently from the point of view of an observer. And vice versa, it may distinguish programs that are observationally equal.

Example of different behavior of model-theoretically "equal" programs

Example

$$A \leftarrow B$$
$$B \leftarrow$$
$$B \leftarrow B$$



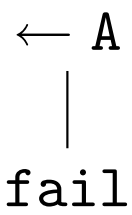
Example

$$A \leftarrow B$$
$$B \leftarrow B$$
$$B \leftarrow$$

Example of different behavior of model-theoretically "equal" programs

Example

$$A \leftarrow false, B$$
$$B \leftarrow B$$
$$B \leftarrow$$



Example

$$A \leftarrow B, false$$
$$B \leftarrow B$$
$$B \leftarrow$$

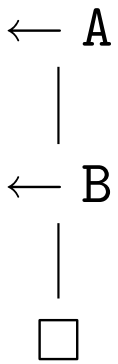
Example of (model-theoretically) "equal" programs with identical behavior

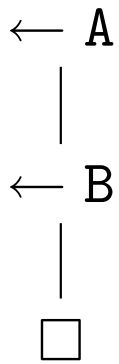
Example

$$A \leftarrow B, \mathit{false}, C, D$$
$$B \leftarrow$$

Example

$$A \leftarrow B, \mathit{false}$$
$$B \leftarrow$$





Observational equivalence

One of the main purposes of giving a 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.

Definition

Let P_1 and P_2 be ground logic programs. Then we define $P_1 \approx P_2$ if and only if, for any goal G, the following four conditions hold:

1. G has a refutation in P_1 if and only if G has a refutation in P_2;
2. G has the same set of computed answers in P_1 and P_2.
3. G has the same set of (correct) partial answers in P_1 and P_2.
4. G has the same set of call patterns in P_1 and P_2.

Correctness of coalgebraic semantics relative to observational semantics

Theorem

For ground logic programs P_1 and P_2, if parallel and-or tree for P_1 is equal to the parallel and-or tree for P_2, then $P_1 \approx P_2$.

Note that the Theorem concerns the and-or trees embedded in the plane, and so it relates observational equivalence with $C(P_f List)$-coalgebra.

The converse of the Theorem does not hold. That is, there can be observationally equivalent programs that have different and-or parallel trees.

Example

Consider two logic programs, P_1 and P_2, whose clauses are exactly the same, with the exception of one clause: P_1 contains
$A \leftarrow B_1, \ldots, B_i, \texttt{false}, \ldots, B_n$; and P_2 contains the clause
$A \leftarrow B_1, \ldots B_i, \texttt{false}$ instead.

Conclusions

- We have considered several different derivation strategies that arise in practice of logic programming, and determined that $P_f\,List$-coalgebra gives a sound and complete semantics to and-or parallel derivations.
- We found that the coalgebraic semantics has a close relation to the theory of observables, and we established *correctness result*.

We plan to extend this work to first-order logic programs.

Future work - 1. Completeness.

Completeness results in logic programs

The traditional least fixed point semantics does not give an account to infinite derivations, but it is sound and complete. The greatest fixed point semantics can handle infinite derivations, but completeness fails.

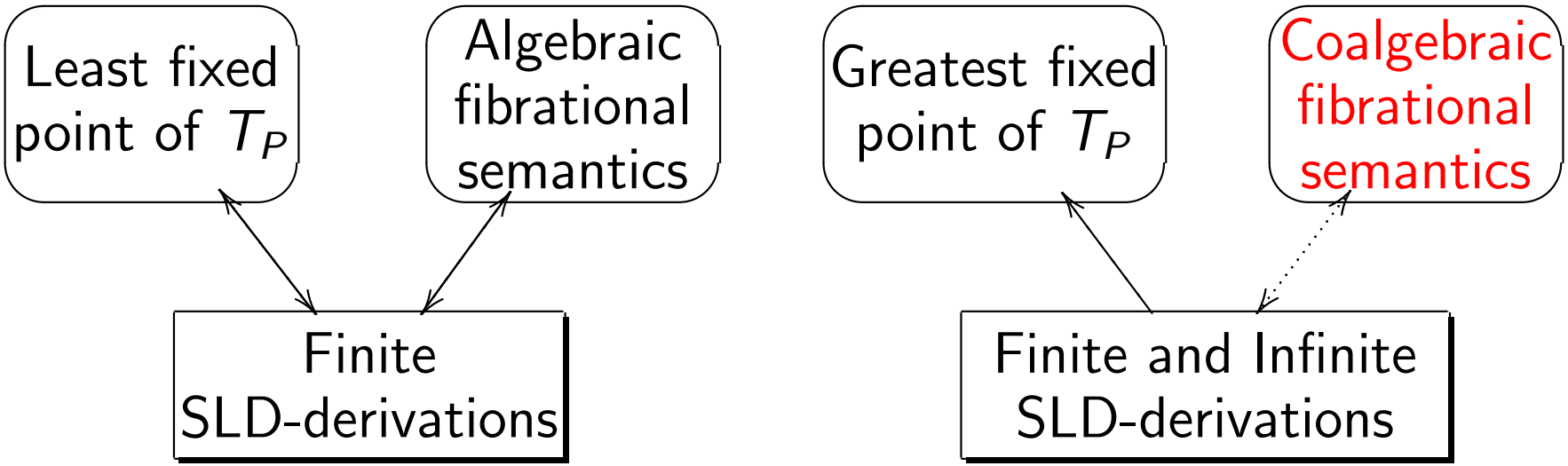
Example

The program below will be characterised by the greatest fixed point of the T_P operator, which contains $R(f^{\omega}(a))$; whereas no infinite term will be computed via SLD-resolution.

$$R(x) \quad \leftarrow \quad R(f(x))$$

We are hopeful that this can be resolved when we achieve a coalgebraic semantics for first-order logic programs.

Soundness and completeness of different-style semantics for LP



Future work - 2. Coinductive Logic Programming

In [Simon, Gupta et al. 2006 - 2007], there was proposed an implementation of logic programs that describe infinite objects. The resolution procedure was changed, so that circular derivations could result in finite computed answers.

Example

The following program `stream` defines the infinite stream of binary bits:

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

We hope to relate the coalgebraic semantics to this implementation.

Challenge - 3. Soundness

Parallel strategies are unsound?

In general first-order case, the parallel derivation strategies are not sound - because they ignore variable dependencies that exist between different atoms in a goal. To make parallel logic programming sound, more technically sophisticated derivation strategies have to be introduced.

We are not entirely sure whether the coalgebraic semantics will be able to describe those; it may need to be restricted to sequential derivations.

Thank you!