Coalgebraic Semantics for Parallel Derivation Strategies in Logic Programming.

Ekaterina Komendantskaya, joint work with John Power

School of Computing, University of Dundee

ScotCat,
4th Scottish Category Theory Seminar
13 May 2011

Outline

Outline

1. Logic programs...

2. Coalgebraic Semantics for them...

Outline

1. Logic programs...
2. Coalgebraic Semantics for them...
3. Lawvere theory in this picture

First-order syntax

We fix the *alphabet* **A** to consist of

- Signature Σ:
  - constant symbols a, b, c, possibly with finite subscripts;
  - function symbols $f, g, h, f_1 \ldots$, with arities;
- variables $x, y, z, x_1 \ldots$;
- predicate symbols $P, Q, R, P_1, P_2 \ldots$, with arities;

Term:

$$t = a \mid x \mid f(t)$$

Atomic Formula: $At = P(t_1, \ldots, t_n)$, where P is a predicate symbol of arity n and t_i is a term.

Example

Signature: $\{O, S\}$ - for natural numbers; {a,b,c} for a graph with 3 nodes.
Atoms: nat(x), edge(a,b), etc...

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)
```

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

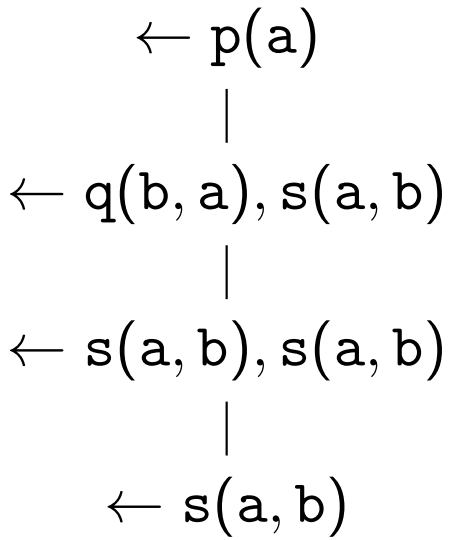
Example: Goal $\leftarrow$ p(a).

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

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

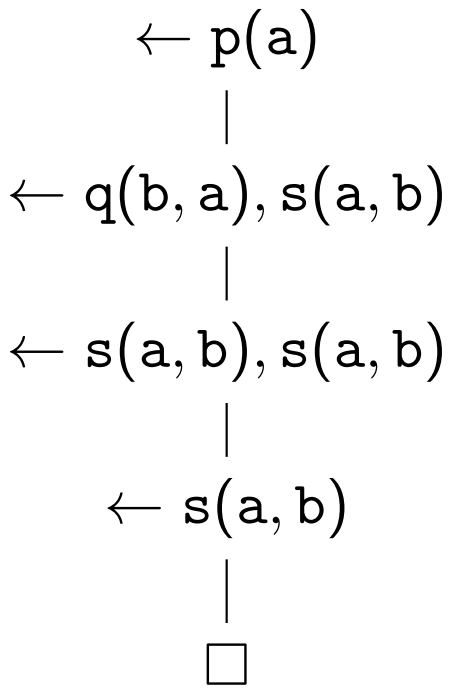
Example: Goal $\leftarrow$ p(a).

```
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).

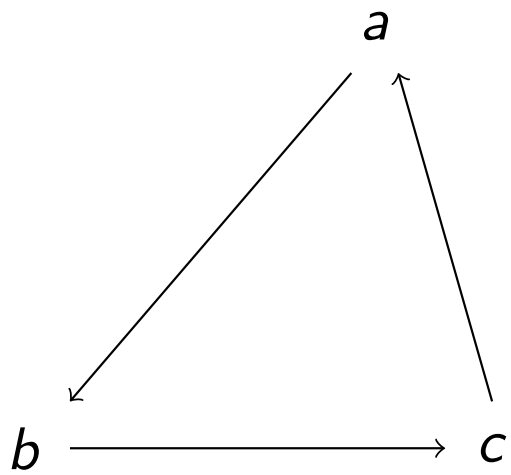
$$
\begin{array}{rcl}
\texttt{q(b,a)} & \leftarrow & \texttt{s(a,b)} \\
\texttt{q(b,a)} & \leftarrow & \\
\texttt{s(a,b)} & \leftarrow & \\
\texttt{p(a)} & \leftarrow & \texttt{q(b,a)}, \texttt{s(a,b)}
\end{array}
$$



Logic program: Graph connectivity

Example

```
connected(x,x)  ←
connected(x,y)  ←  edge(x,z),
connected(z,y).
     edge(a,b)  ←
     edge(b,c)  ←
     edge(c,a)  ←
```



Example: Goal $\leftarrow$ connected(a, c).

Example

```
connected(x,x)  ←
connected(x,y)  ←  edge(x,z),
connected(z,y)
     edge(a,b)  ←
     edge(b,c)  ←
     edge(c,a)  ←
```

$$\leftarrow \texttt{connected}(\texttt{a}, \texttt{c})$$

$$\Big|$$

$$\leftarrow \texttt{edge}(\texttt{a}, \texttt{x}), \texttt{connected}(\texttt{x}, \texttt{c})$$

Example: Goal $\leftarrow$ connected(a, c).

Example

```
connected(x,x)  ←
connected(x,y)  ←  edge(x,z),
connected(z,y)
     edge(a,b)  ←
     edge(b,c)  ←
     edge(c,a)  ←
```

$$\leftarrow \texttt{connected}(\texttt{a},\texttt{c})$$
$$\Big|$$
$$\leftarrow \texttt{edge}(\texttt{a},\texttt{x}),\texttt{connected}(\texttt{x},\texttt{c})$$
$$\Big|$$
$$\leftarrow \texttt{connected}(\texttt{b},\texttt{c})$$

Example: Goal $\leftarrow$ connected(a, c).

Example

```
connected(x,x)  ←
connected(x,y)  ←  edge(x,z),
connected(z,y)
     edge(a,b)  ←
     edge(b,c)  ←
     edge(c,a)  ←
```

$\leftarrow$ connected(a, c)

|

$\leftarrow$ edge(a, b), connected(b, c)

|

$\leftarrow$ connected(b, c)

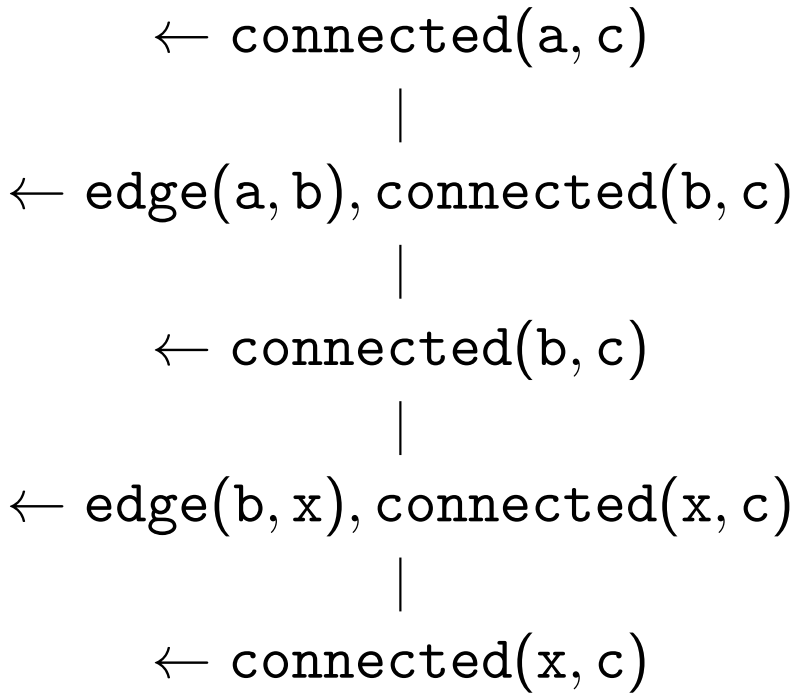
|

$\leftarrow$ edge(b, x), connected(x, c)

Example: Goal $\leftarrow$ connected(a, c).

Example

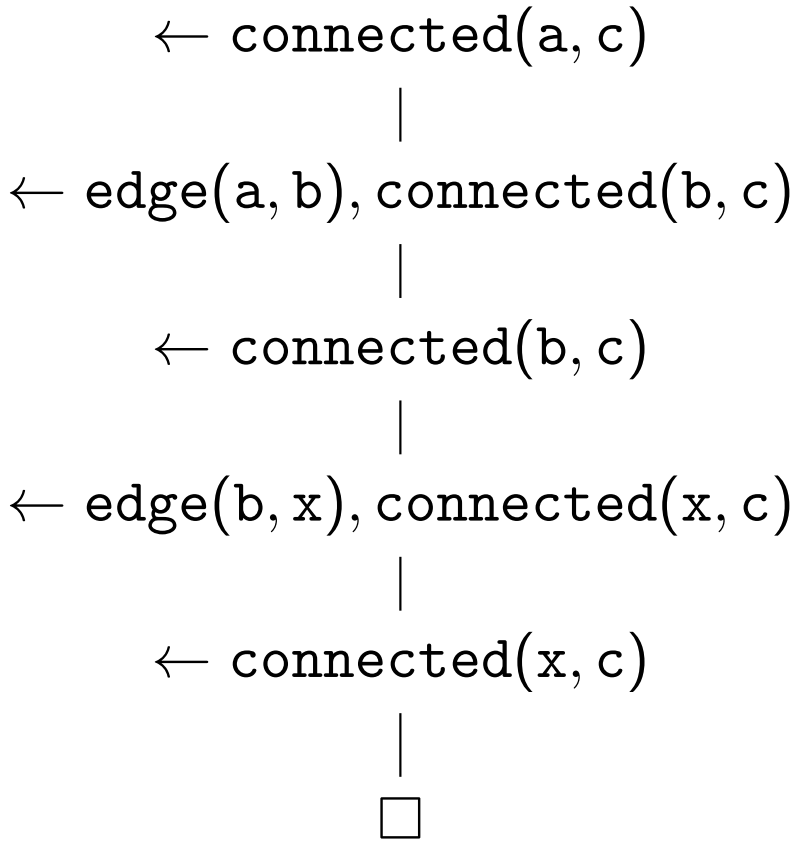
```
connected(x,x)  ←
connected(x,y)  ←  edge(x,z),
connected(z,y).
     edge(a,b)  ←
     edge(b,c)  ←
     edge(c,a)  ←
```



Example: Goal $\leftarrow$ connected(a, c).

Example

```
connected(x,x)   ←
connected(x,y)   ←   edge(x,z),
connected(z,y).
     edge(a,b)   ←
     edge(b,c)   ←
     edge(c,a)   ←
```



Recursion and Corecursion in Logic Programming

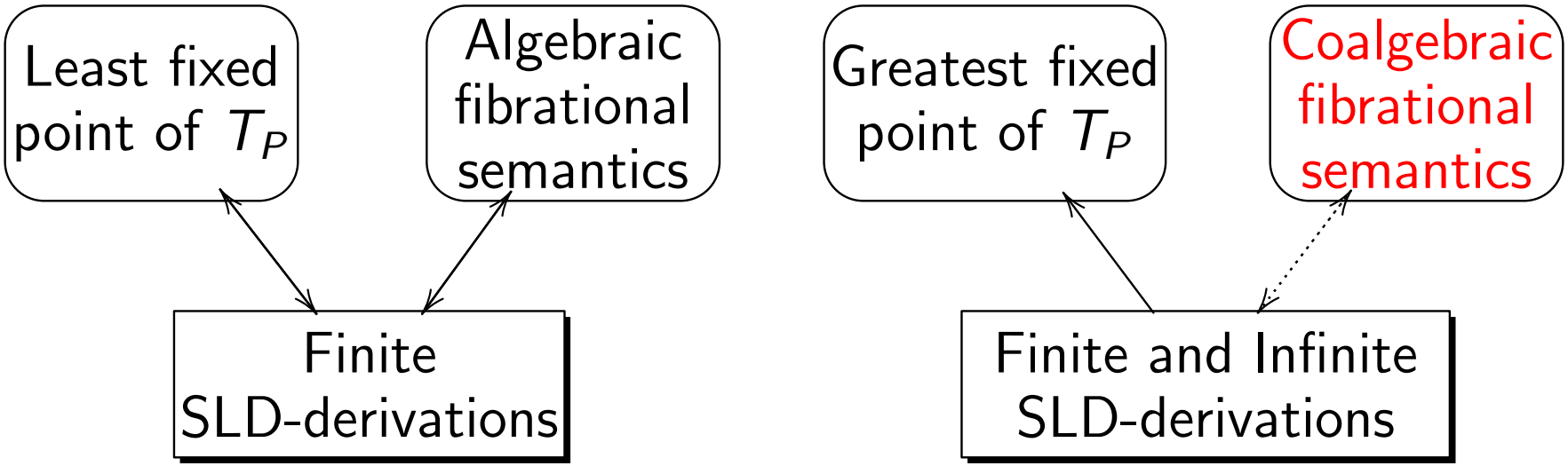
Example

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

Example

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

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.
$p(\texttt{p(a)}) = \{\{\texttt{q(b,a)}, \texttt{s(a,b)}\}\}$.

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

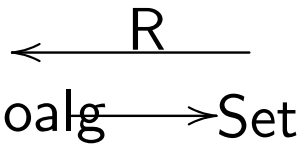
$$Y_{\alpha+1} = Y \times HY_\alpha \overset{Y \times H\delta_{\alpha-1}}{\longrightarrow} Y \times HY_{\alpha-1} = Y_\alpha,$$

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



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

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

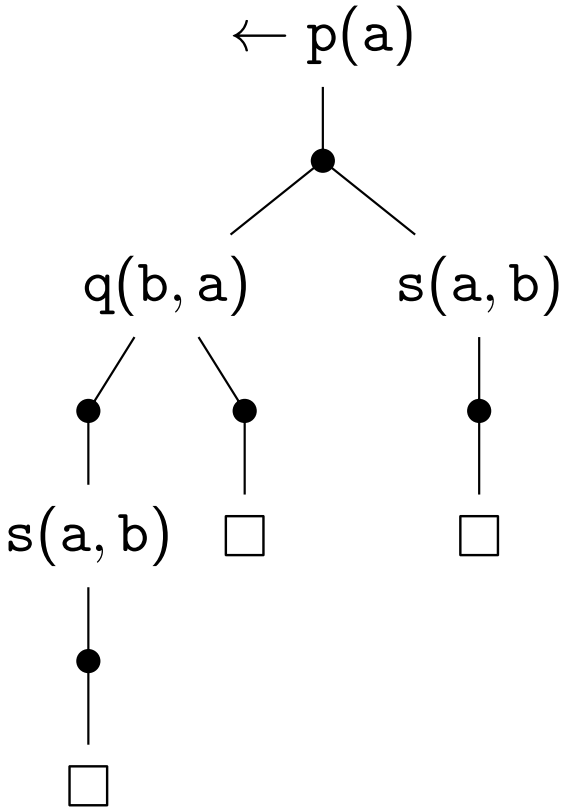
This chain has length ω. As above, we inductively define the objects $\mathrm{At}_0 = \mathrm{At}$ and $\mathrm{At}_{n+1} = \mathrm{At} \times P_f P_f \mathrm{At}_n$, and the cone

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

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

Examples of a derivations

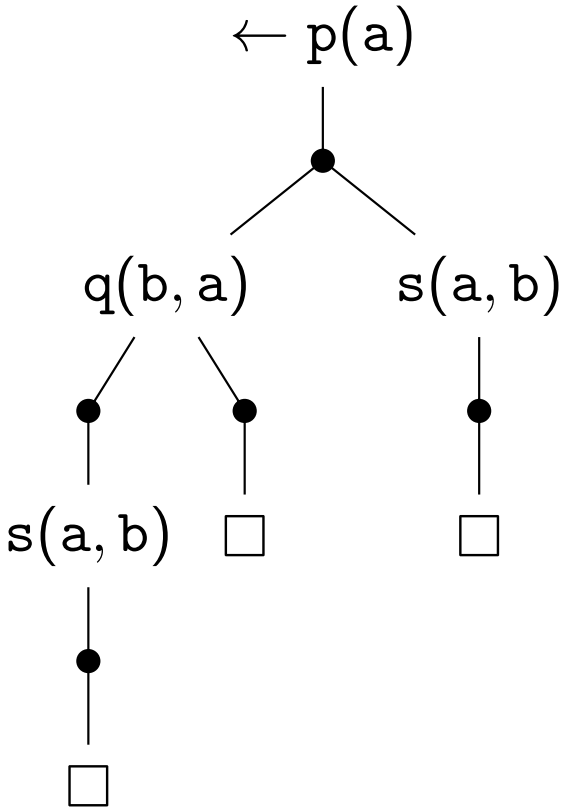
The action of
$\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on
$\mathtt{p(a)}$

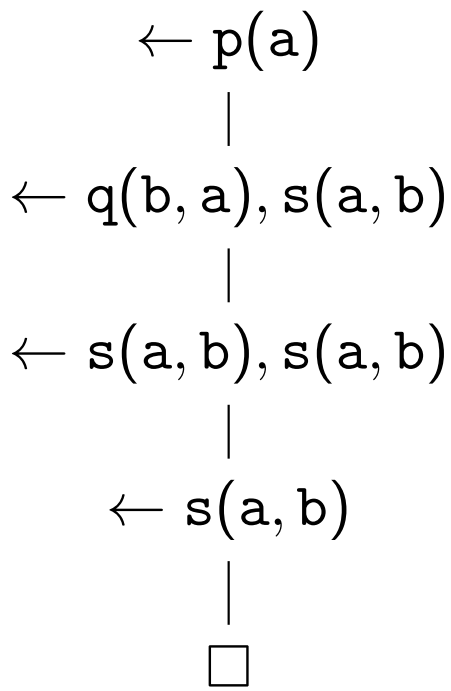


Examples of a derivations

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

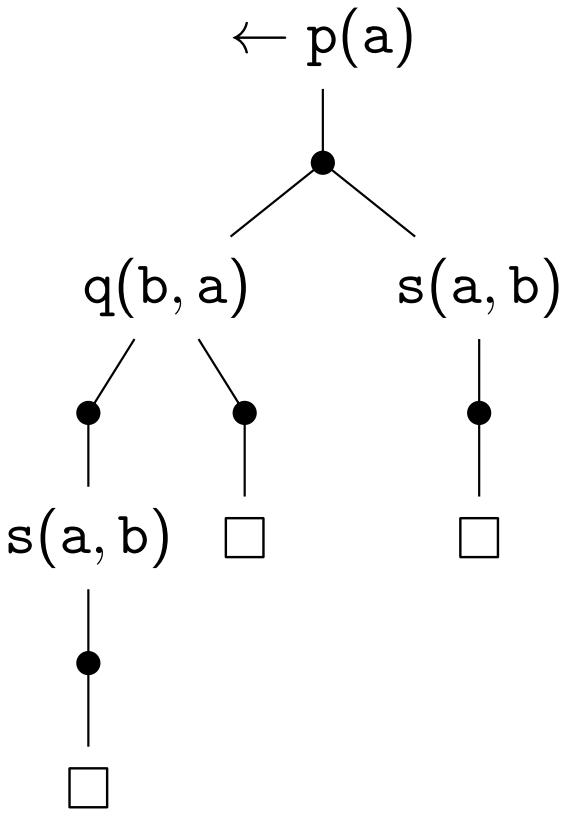
The SLD derivation from a previous example



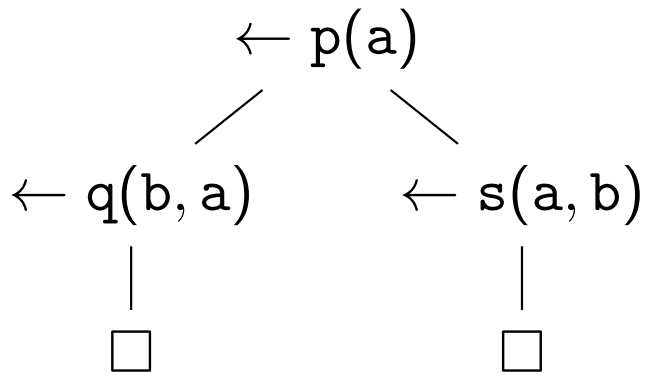


Examples of a derivations

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



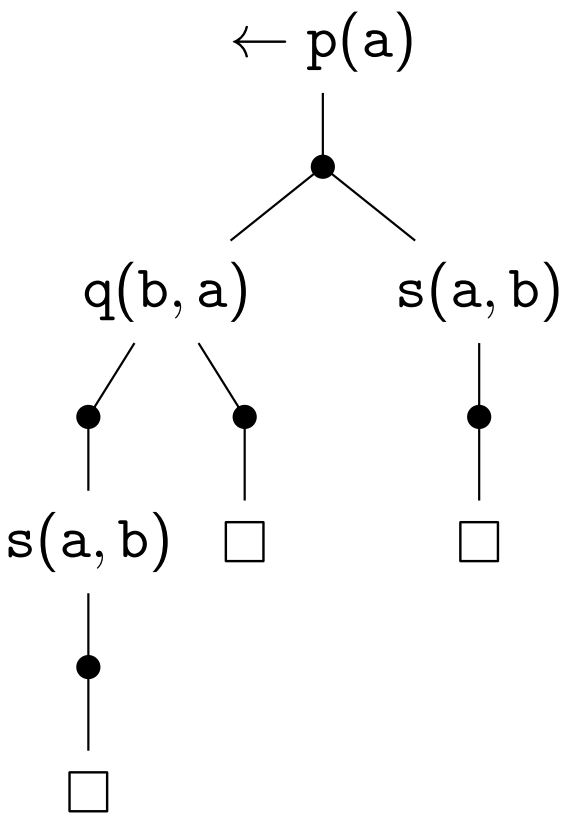
The proof tree

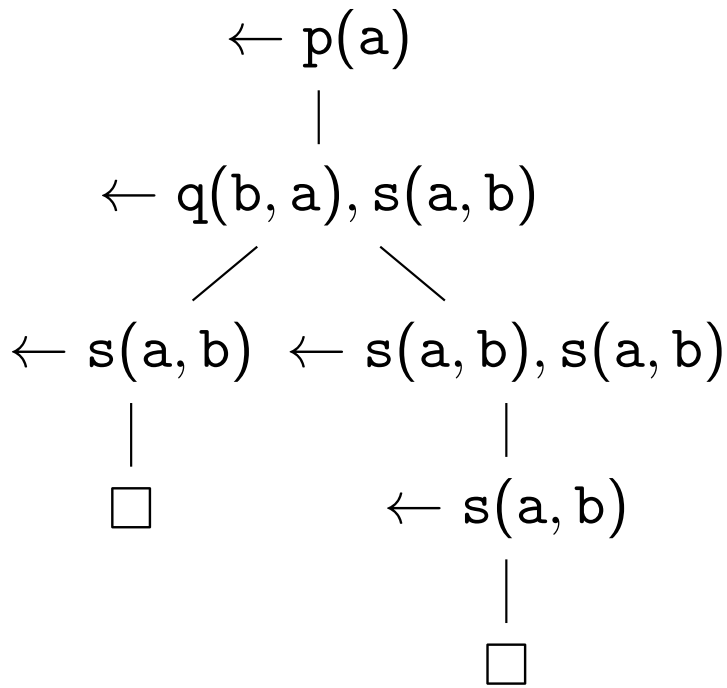


Examples of a derivations

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.

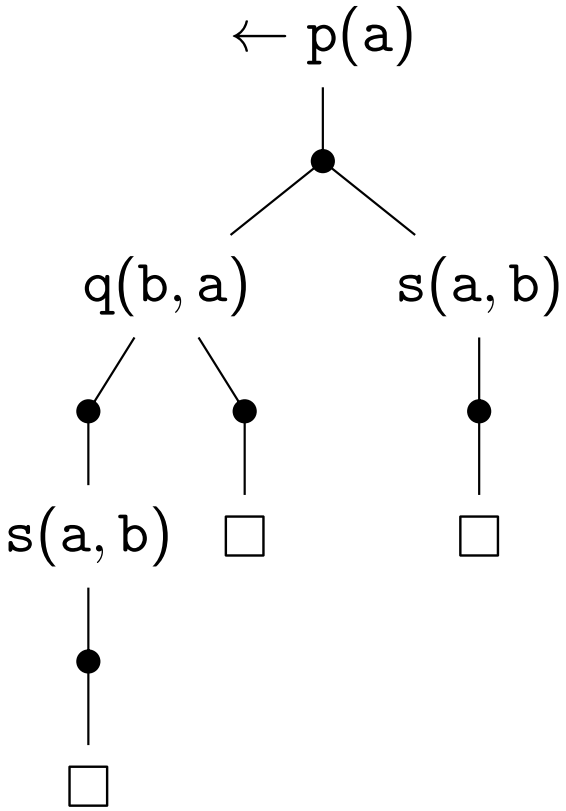
SLD-trees

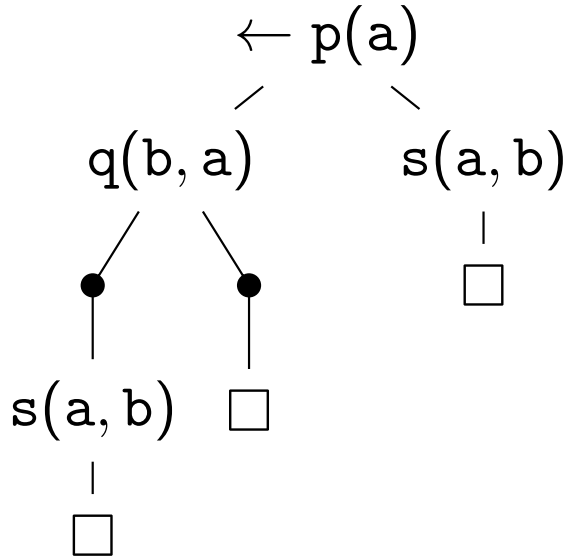
exhibit an or-parallelism in derivations - that is, they show different possibilities of derivations if there are multiple clauses that unify with a goal; but they process conjuncts in a goal sequentially.

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)

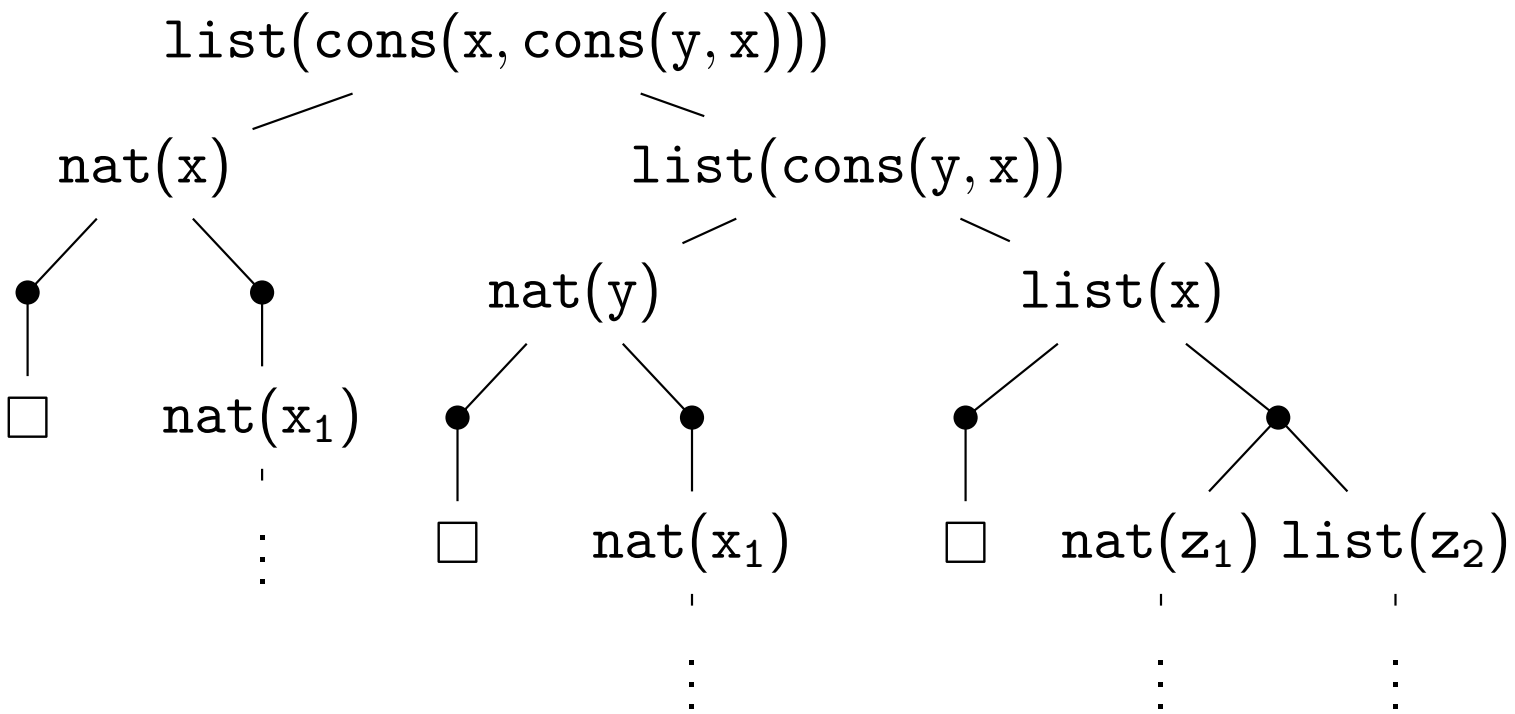
The and-or parallel tree (as defined in the literature)





Except for... they are unsound for first-order programs.

Why unsound?


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

Lawvere theories and the first-order signature

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

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

Define the set $\mathrm{ob}(\mathcal{L}_\Sigma)$ to be the set of natural numbers.
For each natural number n, let $x_1, \ldots, x_n$ be a specified list of distinct variables. Define $\mathrm{ob}(\mathcal{L}_\Sigma)(n, m)$ to be the set of m-tuples $(t_1, \ldots, t_m)$ of terms generated by the function symbols in Σ and variables $x_1, \ldots, x_n$. Define composition in $\mathcal{L}_\Sigma$ by substitution. The interpretation $\| \ \|_\Sigma$ sends an n-ary function symbol f to $f(x_1, \ldots, x_n)$.

Example

The constants `0` and `nil` are modelled by maps from 0 to 1 in $\mathcal{L}_\Sigma$, `s` is modelled by a map from 1 to 1, and `cons` is modelled by a map from 2 to 1. The term `s(0)` is therefore modelled by the map from 0 to 1 given by the composite of the maps modelling `s` and `0`; similarly for the term `s(nil)`, although the latter does not make semantic sense.

Coalgebraic semantics for the first-order case

Assume we have a signature Σ of function symbols and, for each natural number n, a specified list of variables $x_1, \ldots, x_n$. Then, given an arbitrary logic program with signature Σ,

we can consider the functor $At : \mathcal{L}_\Sigma^{op} \to Set$ that sends a natural number n to the set of all atomic formulae with variables among $x_1, \ldots, x_n$ generated by the function symbols in Σ and the predicate symbols appearing in the logic program. A map $f : n \to m$ in $\mathcal{L}_\Sigma$ is sent to the function $At(f) : At(m) \to At(n)$ that sends an atomic formula $A(x_1, \ldots, x_m)$ to $A(f_1(x_1, \ldots, x_n)/x_1, \ldots, f_m(x_1, \ldots, x_n)/x_m)$, i.e., $At(f)$ is defined by substitution.

Attempt to model P by $[\mathcal{L}_\Sigma^{op}, P_f P_f]$-coalgebra

$p : At \longrightarrow P_f P_f At$

Naturality fails: ListNat

There is a map in $\mathcal{L}_\Sigma$ of the form $0 \to 1$ that models the nullary function symbol 0. So, naturality of the map $p : At \longrightarrow P_f P_f At$ in $[\mathcal{L}_\Sigma^{op}, Set]$ would yield commutativity of the diagram

$$\begin{array}{ccc} At(1) & \longrightarrow & P_f P_f At(1) \\ \downarrow & & \downarrow \\ At(0) & \longrightarrow & P_f P_f At(0) \end{array}$$

No clause "$\mathtt{nat(x)} \leftarrow$", but the commutativity of the diagram would in turn imply that there cannot be a clause in ListNat of the form $\mathtt{nat(0)} \leftarrow$. In fact, there is!

The solution: Lax naturality, Posets and and P_c

The diagram need not commute, but rather the composite via $P_f P_f At(m)$ need only yield a subset of that via $At(n)$.

$$\begin{array}{ccc} At(m) & \longrightarrow & P_f P_f At(m) \\ \downarrow & & \downarrow \geq \\ At(n) & \longrightarrow & P_f P_f At(n) \end{array}$$

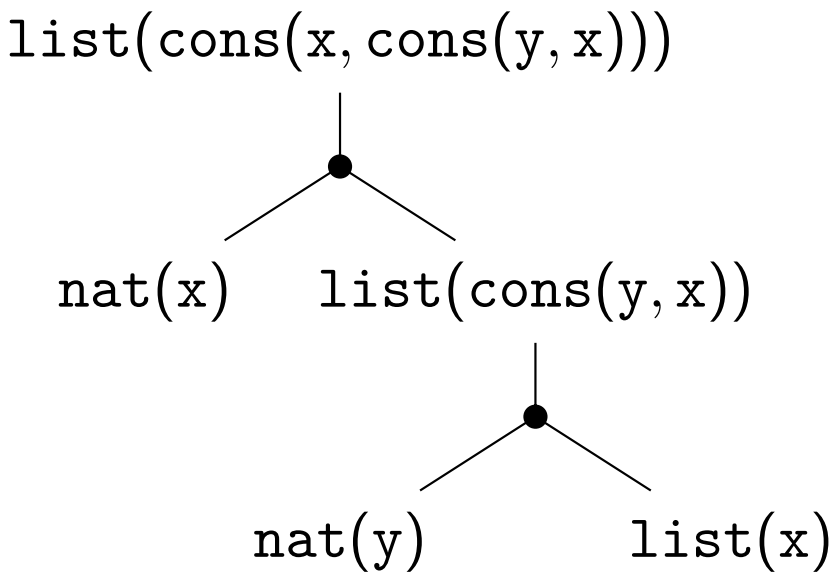
$p : At \longrightarrow P_c P_f At$ gives a $Lax(\mathcal{L}_\Sigma^{op}, P_c P_f)$-coalgebra structure on At; and p determines a $Lax(\mathcal{L}_\Sigma^{op}, C(P_c P_f))$-coalgebra structure $\bar{p} : At \longrightarrow C(P_c P_f)(At)$.

Note that we extend *Set* to *Poset* in $At : \mathcal{L}_\Sigma^{op} \to Set$, and change P_f to P_c for the endofunctor $P_c P_f$.

Examples of first-order coinductive trees determined by the semantics:

$A(x, y) \in At(2)$

Then apply At to the map $(s, s) : 1 \to 2$ in $\mathcal{L}_\Sigma$



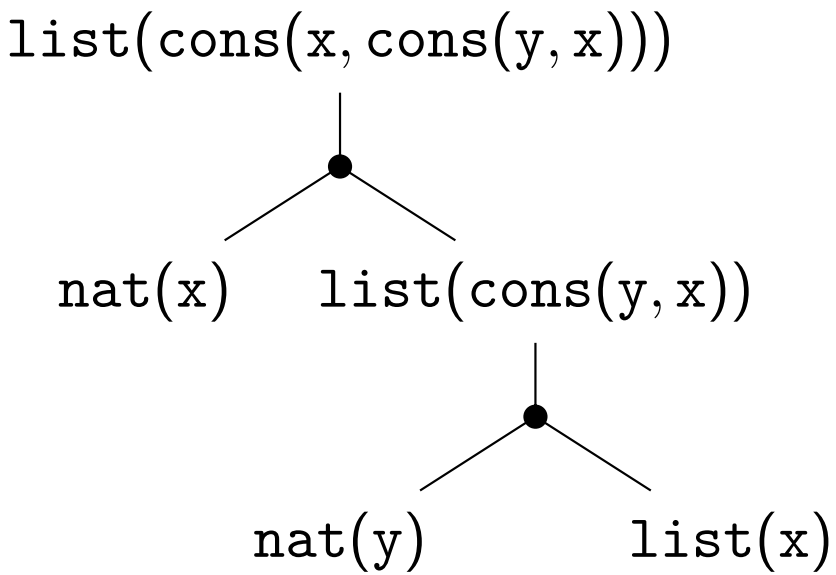
Examples of first-order coinductive trees determined by the semantics:

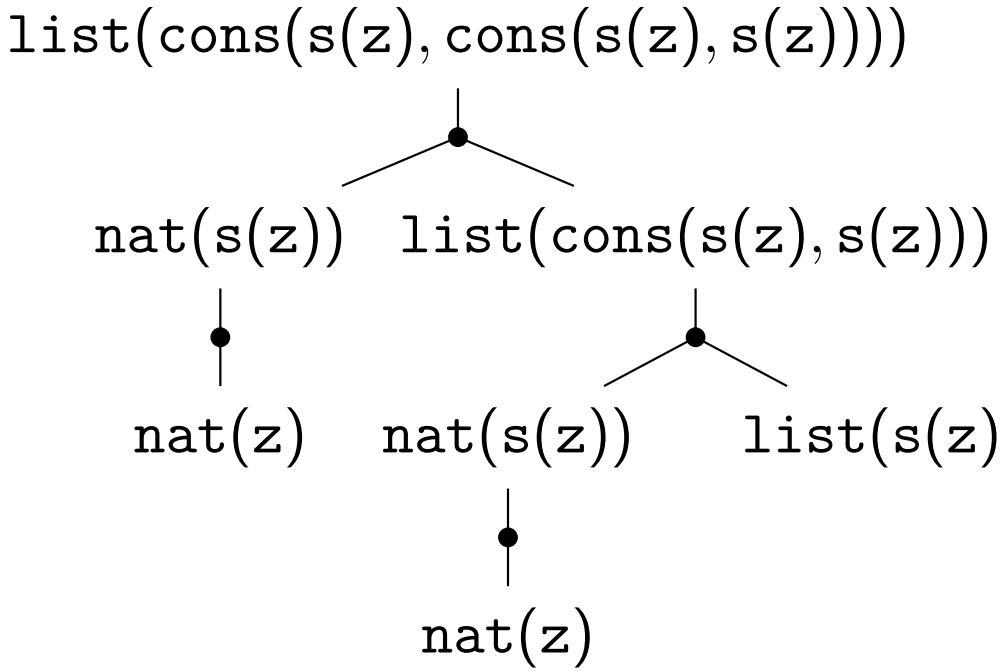
$A(x, y) \in At(2)$

Then apply At to the map $(s, s) : 1 \to 2$ in $\mathcal{L}_\Sigma$

$A(z) \in At(1)$

$At((s, s))(A(x, y))$ is an element of $P_c P_f At(1)$.

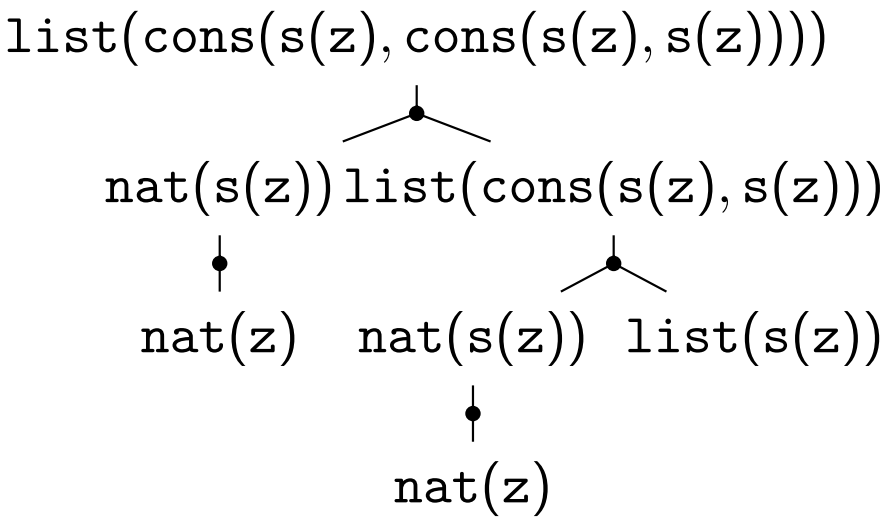




Examples of first-order coinductive trees determined by the semantics:

$A(z) \in At(1)$

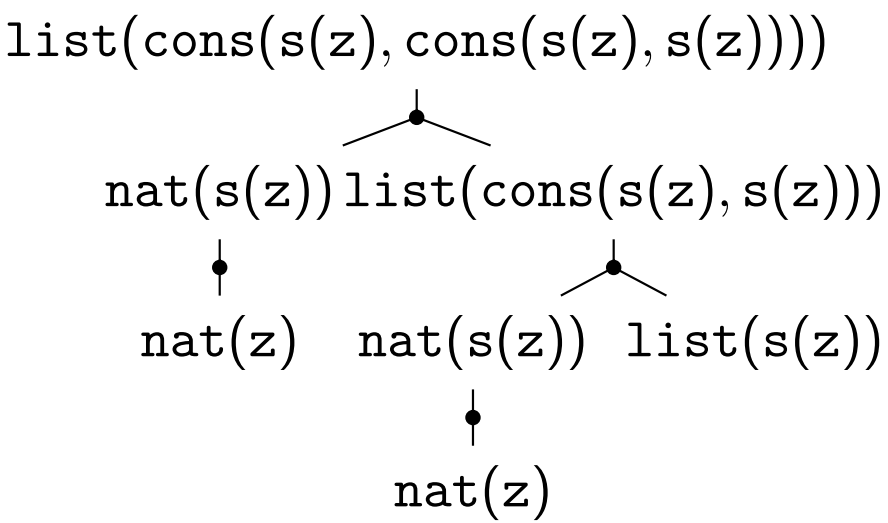
Then apply At to the map $O : 0 \to 1$ in $\mathcal{L}_\Sigma$.



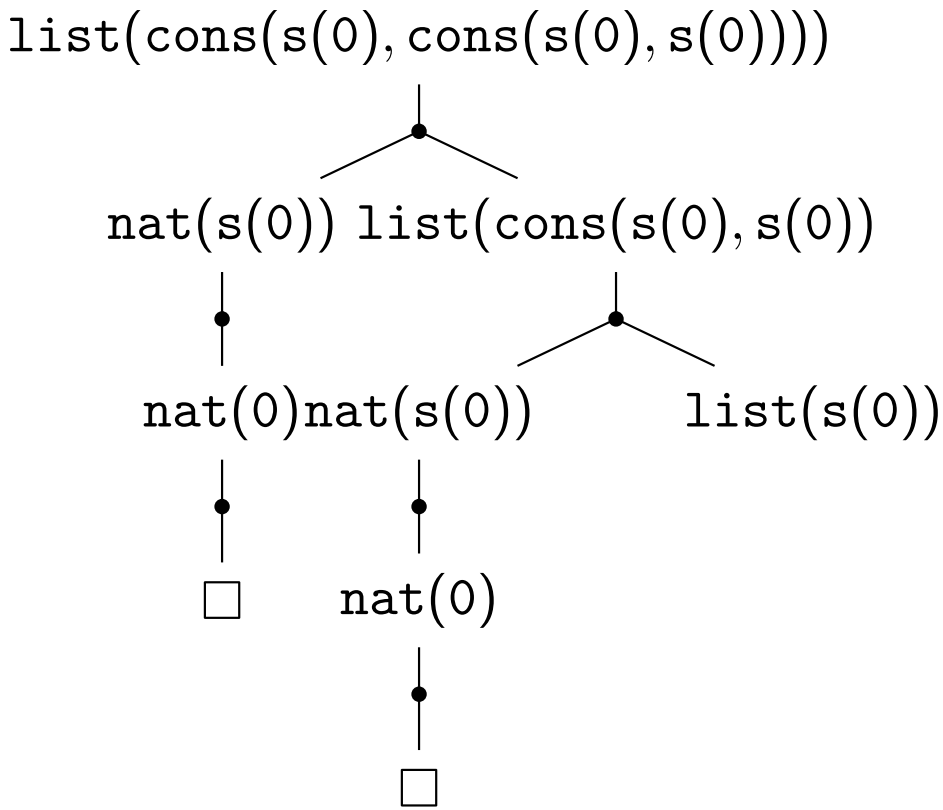
Examples of first-order coinductive trees determined by the semantics:

$A(z) \in At(1)$

Then apply At to the map $O : 0 \to 1$ in $\mathcal{L}_\Sigma$.



$pAt(0)At((s,s))(A(x,y))$



Thank you!