Coalgebraic Logic Programming

Katya Komendantskaya, joint work with M. Schmidt, J. Heras

School of Computing, University of Dundee, UK

21 July 2014

Recursion and Corecursion in Logic Programming

1. in the 70s-80s: Apt, van Emden, Kowalski: study of recursion and least Herbrand model semantics of LP

Recursion and Corecursion in Logic Programming

1. in the 70s-80s: Apt, van Emden, Kowalski: study of recursion and least Herbrand model semantics of LP
2. in the 80s; Abdallah, van Emden, Lloyd: "perpetual" computations in LP and the greatest fixed point semantics of LP: incomplete, no finite procedure for computations given

Recursion and Corecursion in Logic Programming

1. in the 70s-80s: Apt, van Emden, Kowalski: study of recursion and least Herbrand model semantics of LP
2. in the 80s; Abdallah, van Emden, Lloyd: "perpetual" computations in LP and the greatest fixed point semantics of LP: incomplete, no finite procedure for computations given
3. 2000s: Gupta, Simon *et al*: CoLP: finite derivation procedure for coinductive programs, soundness and completeness for programs describing regular trees.

Recursion and Corecursion in Logic Programming

1. in the 70s-80s: Apt, van Emden, Kowalski: study of recursion and least Herbrand model semantics of LP
2. in the 80s; Abdallah, van Emden, Lloyd: "perpetual" computations in LP and the greatest fixed point semantics of LP: incomplete, no finite procedure for computations given
3. 2000s: Gupta, Simon *et al*: CoLP: finite derivation procedure for coinductive programs, soundness and completeness for programs describing regular trees.
4. Our work, from 2010, – coalgebraic semantics for LP, and inspired derivation procedures.

Recursion and Corecursion in Logic Programming

Example

$$\begin{array}{rcl} \texttt{bit(0)} & \leftarrow & \\ \texttt{bit(1)} & \leftarrow & \\ \texttt{list(nil)} & \leftarrow & \\ \texttt{list(cons (X, Y))} & \leftarrow & \texttt{bit(X), list(Y)} \end{array}$$

Example

$$\texttt{stream(cons (X,Y))} \leftarrow \texttt{bit(X)}, \texttt{stream(Y)}$$

SLD-resolution (+ unification and backtracking) behind LP derivations.

Example

```
nat(0) ←
nat(s(x)) ← nat(x)
list(nil) ←
list(cons x y) ← nat(x),
                        list(y)
```

$$\leftarrow \mathtt{list}(\mathtt{cons}(x, y))$$
$$|$$
$$\leftarrow \mathtt{nat}(x), \mathtt{list}(y)$$

SLD-resolution (+ unification) is behind LP derivations.

Example

```
nat(0) ←
nat(s(x)) ← nat(x)
list(nil) ←
list(cons x y) ← nat(x),
                    list(y)
```

$$\leftarrow \texttt{list}(\texttt{cons}(x, y))$$
$$|$$
$$\leftarrow \texttt{nat}(x), \texttt{list}(y)$$
$$|$$
$$\leftarrow \texttt{list}(y)$$

SLD-resolution (+ unification) is behind LP derivations.

Example

```
nat(0) ←
nat(s(x)) ← nat(x)
list(nil) ←
list(cons x y) ← nat(x),
                    list(y)
```

$$\leftarrow \texttt{list}(\texttt{cons}(x, y))$$
$$|$$
$$\leftarrow \texttt{nat}(x), \texttt{list}(y)$$
$$|$$
$$\leftarrow \texttt{list}(y)$$
$$|$$
$$\leftarrow \Box$$

The answer is x/O, y/nil, but we can get more substitutions by backtracking. We can backtrack infinitely many times, but each time computation will terminate.

SLD-resolution (+ unification) is behind LP derivations.

Example

```
nat(0) ←
nat(s(x)) ← nat(x)
list(nil) ←
list(cons x y) ← nat(x),
                     list(y)
```

$$\leftarrow \mathtt{list}(\mathtt{cons}(x, y))$$
$$|$$
$$\leftarrow \mathtt{nat}(x), \mathtt{list}(y)$$
$$|$$
$$\leftarrow \mathtt{list}(y)$$
$$|$$
$$\leftarrow \Box$$

The answer is x/O, y/nil, but we can get more substitutions by backtracking. We can backtrack infinitely many times, but each time computation will terminate.

Nice, clean semantics: least Herbrand model exists, sound&complete, etc...

Corecursion in LP?

Example

```
bit(0) ←
bit(1) ←
stream(scons(x, y)) ←

          bit(x), stream(y)
```

Corecursion in LP?

Example

```
bit(0) ←
bit(1) ←
stream(scons(x, y)) ←
        bit(x), stream(y)
```

No answer, as derivation never terminates.

Corecursion in LP?

Example

```
bit(0) ←
bit(1) ←
stream(scons(x, y)) ←
            bit(x), stream(y)
```

No answer, as derivation never terminates.
Semantics may go wrong as well: least Herbrand models will contain an infinite term corresponding to stream: so completeness fails.

$$\leftarrow \texttt{stream}(\texttt{scons}(x, y))$$
$$|$$
$$\leftarrow \texttt{bit}(x), \texttt{stream}(y)$$
$$|$$
$$\leftarrow \texttt{stream}(y)$$
$$|$$
$$\leftarrow \texttt{bit}(x_1), \texttt{stream}(y_1)$$
$$|$$
$$\leftarrow \texttt{stream}(y_1)$$
$$|$$
$$\leftarrow \texttt{bit}(x_2), \texttt{stream}(y_2)$$
$$|$$
$$\leftarrow \texttt{stream}(y_2)$$
$$|$$
$$\vdots$$

It can be worse....

Example

```
bit(0) ←
bit(1) ←
list(cons(x, y)) ←
              bit(x), list(y)
list(nil) ←
```

It can be worse....

Example

$\texttt{bit(0)} \leftarrow$

$\texttt{bit(1)} \leftarrow$

$\texttt{list(cons(x, y))} \leftarrow$

$\texttt{bit(x), list(y)}$

$\texttt{list(nil)} \leftarrow$

No answer, as derivation never terminates.

It can be worse....

Example

```
bit(0) ←
bit(1) ←
list(cons(x, y)) ←
                 bit(x), list(y)
list(nil) ←
```

No answer, as derivation never terminates.
Semantics goes wrong: this time, soundness!

$$
\begin{array}{c}
\leftarrow \texttt{list}(\texttt{cons}(x, y)) \\
| \\
\leftarrow \texttt{bit}(x), \texttt{list}(y) \\
| \\
\leftarrow \texttt{list}(y) \\
| \\
\leftarrow \texttt{bit}(x_1), \texttt{list}(y_1) \\
| \\
\leftarrow \texttt{list}(y_1) \\
| \\
\leftarrow \texttt{bit}(x_2), \texttt{list}(y_2) \\
| \\
\leftarrow \texttt{list}(y_2) \\
| \\
\vdots
\end{array}
$$

Solution - 1 [Gupta, Simon et al., 2007 - 2008]

If a formula repeatedly appears as a resolvent (modulo α-conversion), then conclude the proof.

Example

```
bit(0) ←
bit(1) ←
stream(scons (X, Y)) ←
            bit(X), stream(Y)
```

$\leftarrow \texttt{stream(X)}$

|

$\leftarrow \texttt{bit(X1)}, \texttt{stream(X)}$

|

$\leftarrow \texttt{stream(X)}$

|

$\Box^{c}$

Solution - 1 [Gupta, Simon et al., 2007 - 2008]

If a formula repeatedly appears as a resolvent (modulo α-conversion), then conclude the proof.

Example

```
bit(0) ←
bit(1) ←
stream(scons (X, Y)) ←
            bit(X), stream(Y)
```

$\leftarrow \texttt{stream(X)}$

|

$\leftarrow \texttt{bit(X1)}, \texttt{stream(X)}$

|

$\leftarrow \texttt{stream(X)}$

|

$\square^c$

The answer is: $X/cons(0, X)$.

Requires programs to be regular, in order to be sound and complete

CoALP: what is it about?

- syntactically – first-order logic programming;
- operationally – lazy (co)recursion;

CoALP: what is it about?

- syntactically – first-order logic programming;
- operationally – lazy (co)recursion;
- inspired by coalgebraic fibrational semantics;
- uses and-or parallel trees, but restricts unification to matching;

Term-matcher

A substitution θ is a term-matcher for A and B is $A\theta = B$.

CoALP: what is it about?

- syntactically – first-order logic programming;
- operationally – lazy (co)recursion;
- inspired by coalgebraic fibrational semantics;
- uses and-or parallel trees, but restricts unification to matching;

Term-matcher

A substitution θ is a term-matcher for A and B is $A\theta = B$.

- explores the tree-structure of partial proofs – "coinductive trees";

Coinductive tree...

is an and-or-parallel tree in which unification is restricted to term-matching;

CoALP: what is it about?

- syntactically – first-order logic programming;
- operationally – lazy (co)recursion;
- inspired by coalgebraic fibrational semantics;
- uses and-or parallel trees, but restricts unification to matching;

Term-matcher

A substitution θ is a term-matcher for A and B is $A\theta = B$.

- explores the tree-structure of partial proofs – "coinductive trees";

Coinductive tree...

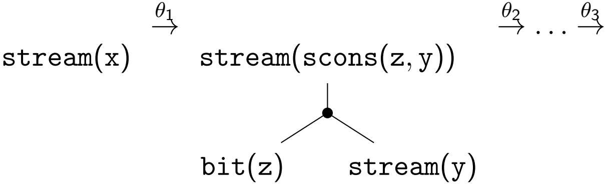
is an and-or-parallel tree in which unification is restricted to term-matching;

- Coinductive trees give a measure for lazy guarded corecursion, (cf. "clocked corecursion")

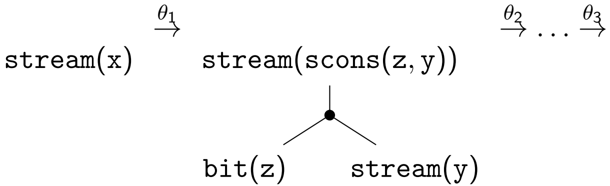
Lazy Corecursion in CoALP: Coinductive trees

$$\texttt{stream(x)} \xrightarrow{\theta_1}$$

Lazy Corecursion in CoALP: Coinductive trees



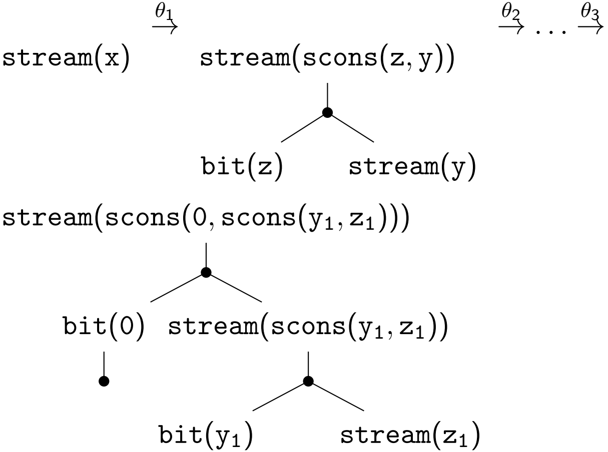
Lazy Corecursion in CoALP: Coinductive trees



Note that transitions θ may be determined in a number of ways:

- using mgus;
- non-deterministically;
- in a distributed/parallel manner.

Lazy Corecursion in CoALP



The above would correspond to one-branch of SLD-derivations we have seen! The main driving force: separation of layers of computations into different dimensions.

Computationally essential:

1. for coinductive `Stream` program, the coinductive-trees are finite!!! – both in depth and in breadth;
2. each tree gives only a partial computation – it is not like eager SLD-trees we have seen earlier;

Computationally essential:

1. for coinductive `Stream` program, the coinductive-trees are finite!!! – both in depth and in breadth;
2. each tree gives only a partial computation – it is not like eager SLD-trees we have seen earlier;

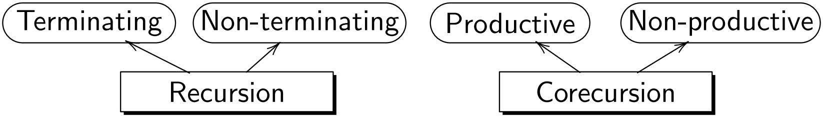
1. $\Rightarrow$ gives hope for a formalism to describe termination and productivity, as in functional languages
2. $\Rightarrow$ hints there may be laziness involved...

What do we gain?

1. A coherent theory of termination and productivity of recursion and corecursion in LP

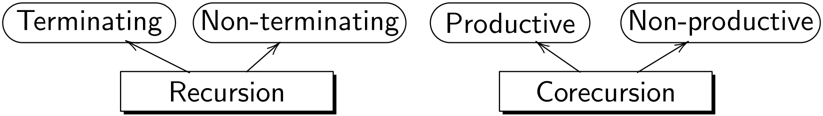
Theory of Productivity in LP

Typeful functional theorem provers:

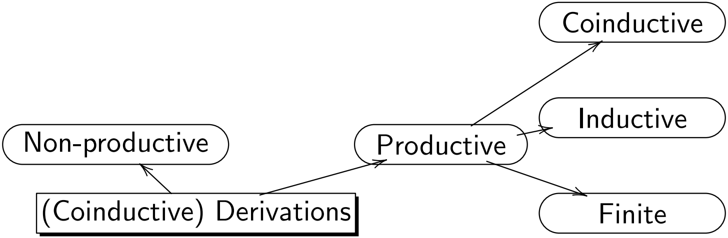


Theory of Productivity in LP

Typeful functional theorem provers:



CoALP



What do we gain?

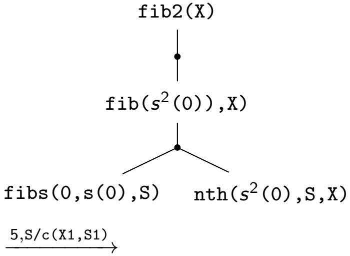
1. A coherent theory of termination and productivity of recursion and corecursion in LP
2. Extension of classes of inductive and coinductive programs we can handle,
3. Mixing induction/coinduction.

Stream of Fibonacci numbers:

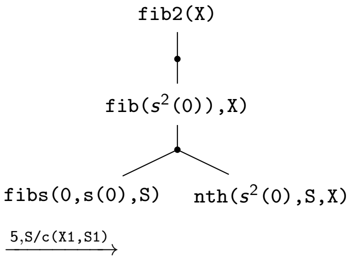
Falls into infinite loops in Prolog and CoLP.

```
1.  add(0,Y,Y).
2.  add(s(X),Y,s(Z)) :- add(X,Y,Z).
3.  fibs(X,Y,cons(X,S)) :- add(X,Y,Z), fibs(Y,Z,S).
4.  nth(0,cons(X,S),X).
5.  nth(s(N),cons(X,S),Y) :- nth(N,S,Y).
6.  fib(N,X) :- fibs(0,s(0),S), nth(N,S,X).
7.  fib2(X) :- fib(s(s(0)),X).
```

Examples of derivations with Fib: lazy step 1

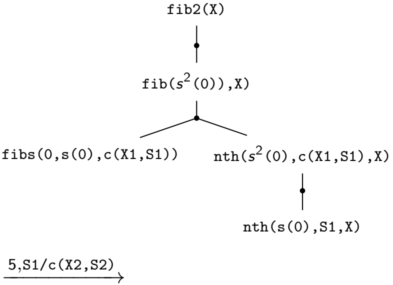


Examples of derivations with Fib: lazy step 1

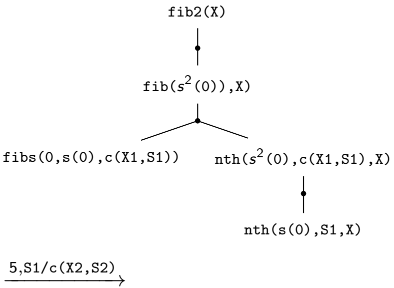



```
1.  add(0,Y,Y).
2.  add(s(X),Y,s(Z)) :-
add(X,Y,Z).
3.  fibs(X,Y,cons(X,S)) :-
add(X,Y,Z), fibs(Y,Z,S).
4.  nth(0,cons(X,S),X).
5.  nth(s(N),cons(X,S),Y) :-
nth(N,S,Y).
6.  fib(N,X) :- fibs(0,s(0),S),
nth(N,S,X).
7.  fib2(X) :- fib(s(s(0)),X).
```

Examples of derivations with Fib: lazy step 2

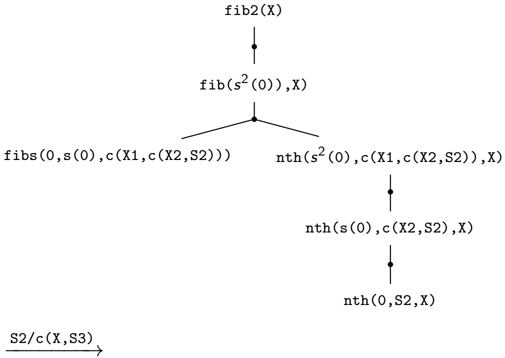


Examples of derivations with Fib: lazy step 2

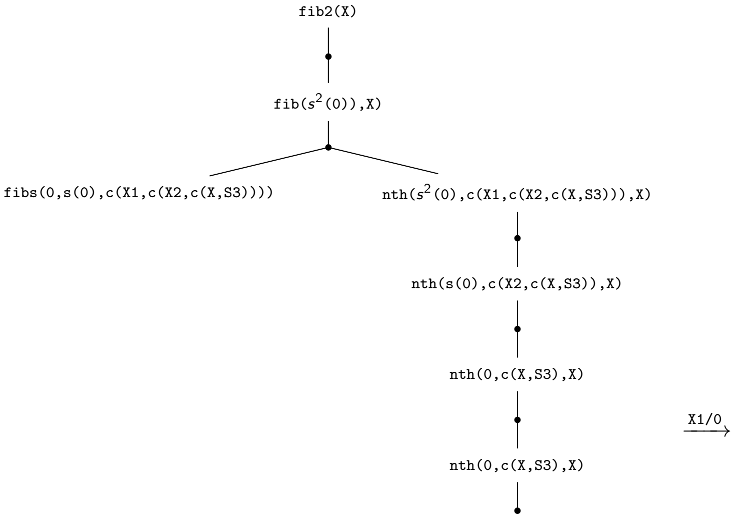



```
1.  add(0,Y,Y).
2.  add(s(X),Y,s(Z)) :-
add(X,Y,Z).
3.  fibs(X,Y,cons(X,S)) :-
add(X,Y,Z), fibs(Y,Z,S).
4.  nth(0,cons(X,S),X).
5.  nth(s(N),cons(X,S),Y) :-
nth(N,S,Y).
6.  fib(N,X) :- fibs(0,s(0),S),
nth(N,S,X).
7.  fib2(X) :- fib(s(s(0)),X).
```

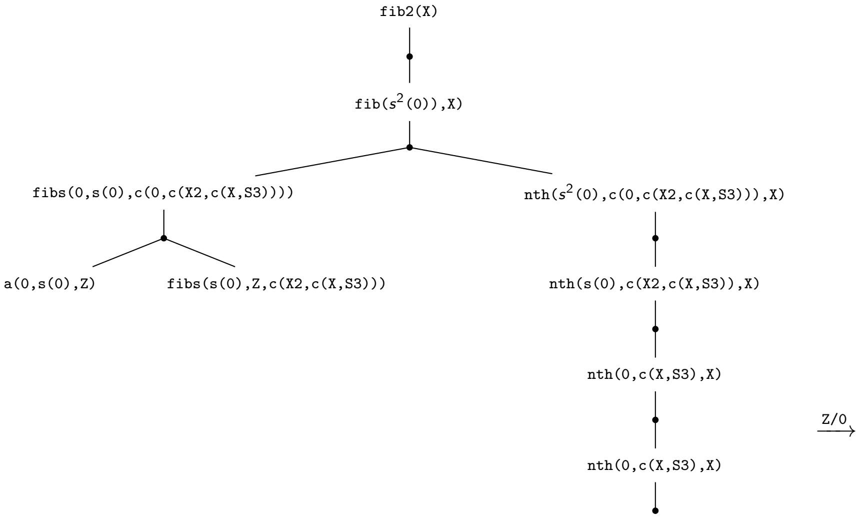
Examples of derivations with Fib: lazy step 3



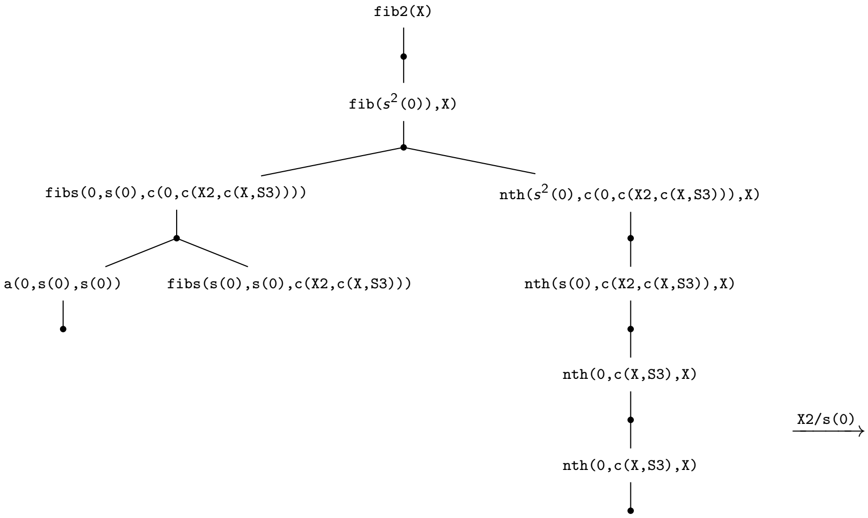
Examples of derivations with Fib: lazy step 4



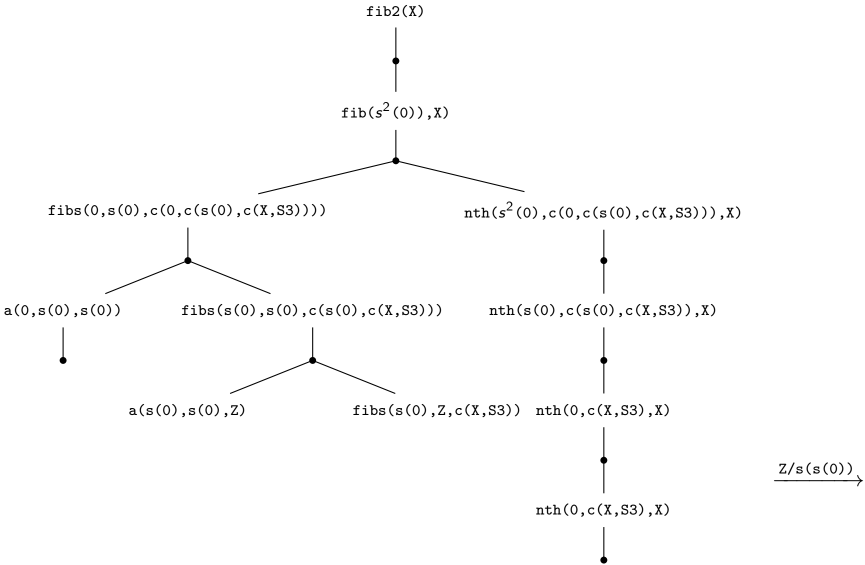
Examples of derivations with Fib: lazy step 5



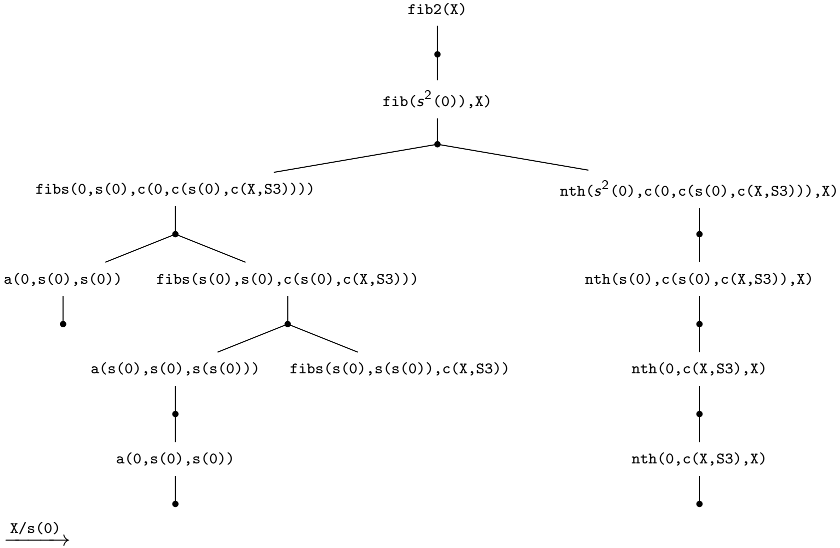
Examples of derivations with Fib: lazy step 6



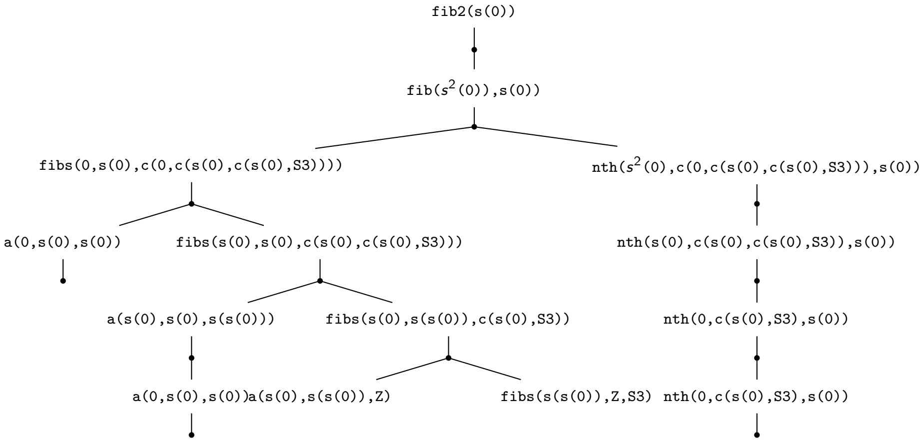
Examples of derivations with Fib: lazy step 7



Examples of derivations with Fib: lazy step 8



Examples of derivations with Fib: lazy step 9



CoALP Properties:

Komendantskaya, Power, Schmidt: Coalgebraic Logic Programming: from Semantics to Implementation, Journal of Logic and Computation, 2014.

- Sound and complete with respect to the coalgebraic semantcs;
- Finite computations are sound and complete with respect to the least Herbrand model semantics (so, we can do as much as standard Prolog).
- Adequacy result for observational semantics.

Logic Programming dialects, compared

	Prolog	Parallel Prolog	Co-LP	CoALP
Fib example	No	No	No	Yes
Execution	Eager	Eager	Eager	Lazy
Corecursion				
Mode of execution				
Declarative semantics				
Operational semantics				

Logic Programming dialects, compared

	Prolog	Parallel Prolog	Co-LP	CoALP
Fib example	No	No	No	Yes
Execution	Eager	Eager	Eager	Lazy
Corecursion	No	No	by Regular Loop detection	Guardedness by constructors
Mode of execution				
Declarative semantics				
Operational semantics				

Logic Programming dialects, compared

	Prolog	Parallel Prolog	Co-LP	CoALP
Fib example	No	No	No	Yes
Execution	Eager	Eager	Eager	Lazy
Corecursion	No	No	by Regular Loop detection	Guardedness by constructors
Mode of execution	Sequential	Parallel	Sequential	Parallel
Declarative semantics				
Operational semantics				

Logic Programming dialects, compared

	Prolog	Parallel Prolog	Co-LP	CoALP
Fib example	No	No	No	Yes
Execution	Eager	Eager	Eager	Lazy
Corecursion	No	No	by Regular Loop detection	Guardedness by constructors
Mode of execution	Sequential	Parallel	Sequential	Parallel
Declarative semantics	lfp	lfp	gfp (restricted)	coalgebraic
Operational semantics				

Logic Programming dialects, compared

	Prolog	Parallel Prolog	Co-LP	CoALP
Fib example	No	No	No	Yes
Execution	Eager	Eager	Eager	Lazy
Corecursion	No	No	by Regular Loop detection	Guardedness by constructors
Mode of execution	Sequential	Parallel	Sequential	Parallel
Declarative semantics	lfp	lfp	gfp (restricted)	coalgebraic
Operational semantics	transitions; states: lists of formulae	transitions; states: lists of formulae	transitions; states: lists of formulae	transitions; states: coinductive trees

Current and future work

1. Using CoALP to formally define a general theory of Termination and Productivity for Recursion and Corecursion in LP

Current and future work

1. Using CoALP to formally define a general theory of Termination and Productivity for Recursion and Corecursion in LP
2. Finalise guardedness conditions

Current and future work

1. Using CoALP to formally define a general theory of Termination and Productivity for Recursion and Corecursion in LP
2. Finalise guardedness conditions
3. Establish soundness criteria for termination of coinductive derivations.

Current and future work

1. Using CoALP to formally define a general theory of Termination and Productivity for Recursion and Corecursion in LP
2. Finalise guardedness conditions
3. Establish soundness criteria for termination of coinductive derivations.
4. Extension of CoALP with constraints

Current and future work

1. Using CoALP to formally define a general theory of Termination and Productivity for Recursion and Corecursion in LP
2. Finalise guardedness conditions
3. Establish soundness criteria for termination of coinductive derivations.
4. Extension of CoALP with constraints
5. Applications to type inference

... join us!

Thank you!

Download your copy of CoALP today:

CoALP webpage: http://staff.computing.dundee.ac.uk/katya/CoALP/