Can CoALP be useful for CoCo?

Katya Komendantskaya, joint work with J. Power, M. Schmidt,
J. Heras, V. Komendantsky

School of Computing, University of Dundee, UK

CoCo'14,
8 May 2014

CoALP-TI

"Coalgebraic Logic Programming for Type Inference" – EPSRC grant, Sep 2013 – Sep 2016. (Joint with J.Power, U. Bath)
"Coalgebraic Logic Programming" is an LP dialect with added corecursion and parallelism.

Year1 (September 2013 - September 2014) Well-tuned Haskell implementation

Year2 Experiments on using CoALP for type-inference in various languages;

Year3 Informed implementation of CoALP-TI in one of the above.

Outline

Outline

Outline

Milner, 1978

"A theory of Type Polymorphism in Programming"

Milner, 1978

"A theory of Type Polymorphism in Programming"
An elegant match between polymorphic λ-calculus and type inference by means of Robinson's unification/resolution algorithm.

Trend in type inference:

improvement in **expressiveness** of the underlying type system, e.g., in terms of

- *Dependent Types*,
- *Type Classes* [Wadler&Blott 89],
- *Generalised Algebraic Types* (GADTs) [Peyton Jones & al, 2006]
- *Dependent Type Classes* [Sozeau & al 08] and
- *Canonical Structures* [Gonthier& al 11].

Milner-style decidable type inference does not always suffice (e.g. the *principal type* may no longer exist), and TI requires additional inference algorithms.

Trend in type inference:

improvement in **expressiveness** of the underlying type system, e.g., in terms of

- *Dependent Types*,
- *Type Classes* [Wadler&Blott 89],
- *Generalised Algebraic Types* (GADTs) [Peyton Jones & al, 2006]
- *Dependent Type Classes* [Sozeau & al 08] and
- *Canonical Structures* [Gonthier& al 11].

Milner-style decidable type inference does not always suffice (e.g. the *principal type* may no longer exist), and TI requires additional inference algorithms.

Implementations of new type inference algorithms include a variety of first-order decision procedures, notably Unification and Logic Programming (LP) [Peyton Jones & al, 2006], Constraint LP [Odersky Sulzmann, Vytiniotis & many more 1999-], LP embedded into interactive tactics (Coq's *eauto*) Sozeau & al. 08], and LP supplemented by rewriting [Gonthier & al, 11].

Example: type inference with Polymorphic types

List Length in Haskell

```
length ::  [a] -> Integer
length [] = 0
length (x:xs) = 1 + length xs
```

Logic program for type inference

$$
\begin{aligned}
&\texttt{cons(X)} \leftarrow \texttt{X} = \texttt{Y} \rightarrow \texttt{list(Y)} \rightarrow \texttt{list(Y)}.\\
&\texttt{plus(X)} \leftarrow \texttt{X} = \texttt{int} \rightarrow \texttt{int} \rightarrow \texttt{int}.\\
&\texttt{nil(X)} \leftarrow \texttt{X} = \texttt{list(Y)}.\\
&\texttt{length(X)} \leftarrow (\texttt{X} = \texttt{Y} \rightarrow \texttt{Z})\ \&\ \texttt{nil(Y)}\ \&\ \texttt{Z} = \texttt{int}\ \&\ \texttt{cons(W)}\ \&\\
&\qquad\qquad (\texttt{W} = \texttt{W1} \rightarrow \texttt{W2} \rightarrow \texttt{Y})\ \&\ \texttt{plus(U)}\ \&\\
&\qquad\qquad (\texttt{U} = \texttt{int} \rightarrow \texttt{Z} \rightarrow \texttt{Z})\ \&\ \texttt{W2} = \texttt{Y}.
\end{aligned}
$$

Query: length(X)?

Answer (any existing PROLOG version): $X = \mathit{list}(\_) \rightarrow \mathit{int}$.

Trend to do more by type-inference:

... session types,
... writing contracts by means of types:

Example

Vytiniotis et al. "HALO: Haskell to Logic Through Denotational Semantics" [POPL'13]

```
f xs = head (reverse (True :  xs))
g xs = head (reverse xs)
```

Both f and g are well typed and "'can't go wrong"' in Milner's sense, but g will crash for empty list, and `f` will never crash.
Contract:

$$\texttt{reverse} \in (\texttt{xs} : \texttt{CF}) \to \{\texttt{ys} \mid \texttt{null xs} <=> \texttt{null ys}\}$$

Requires strong first-order type inference engines: Z3, Vampire, E...

Could it get any better?

- Clear trend on Type theory side: increase in type expressivenes (dependent types, GADTs, type classes, session types, etc etc)

Could it get any better?

- Clear trend on Type theory side: increase in type expressivenes (dependent types, GADTs, type classes, session types, etc etc)
- Chaotic use of type-inference engines, also known in the literature as "using off-the-shelf" first order TPs.

Could it get any better?

- Clear trend on Type theory side: increase in type expressivenes (dependent types, GADTs, type classes, session types, etc etc)
- Chaotic use of type-inference engines, also known in the literature as "using off-the-shelf" first order TPs.
- Would it pay-off to get more conceptually elegant on type inference side? – especially bearing in mind the big emphasis on type inference in more expressive type systems.

Could it get any better?

- Clear trend on Type theory side: increase in type expressivenes (dependent types, GADTs, type classes, session types, etc etc)
- Chaotic use of type-inference engines, also known in the literature as "using off-the-shelf" first order TPs.
- Would it pay-off to get more conceptually elegant on type inference side? – especially bearing in mind the big emphasis on type inference in more expressive type systems.
- Would our "Coalgebraic Logic programming" grow to become a type-inference specific theorem prover (with stronger theoretical background and motivation than state-of-the-art SAT/SMT-solvers)?

CoALP: what is it about?

- inspired by coalgebraic fibrational semantics;
- explores the tree-structure of partial proofs – "coinductive trees";
- uses lazy guarded corecursion using measures of corecursive steps given by coinductive trees (cf. "clocked corecursion");
- parallel...

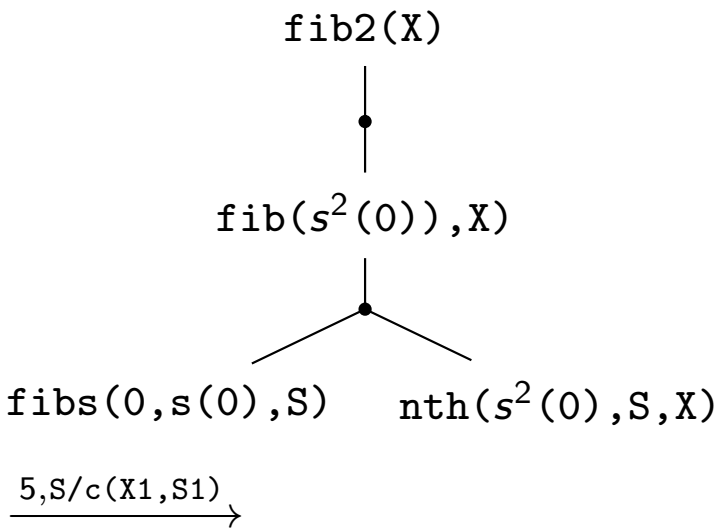
CoALP in one example

Stream of Fibonacci numbers:

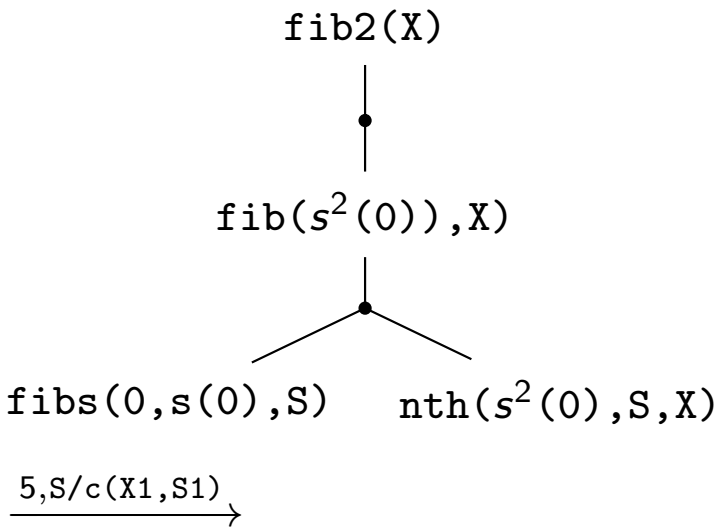
Falls into infinite loops in Prolog and Prolog-like version of CoLP [Gupta et al. 2007] [Both are eager...] Those powerful SAT/SMT solvers would not do it either.

```
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

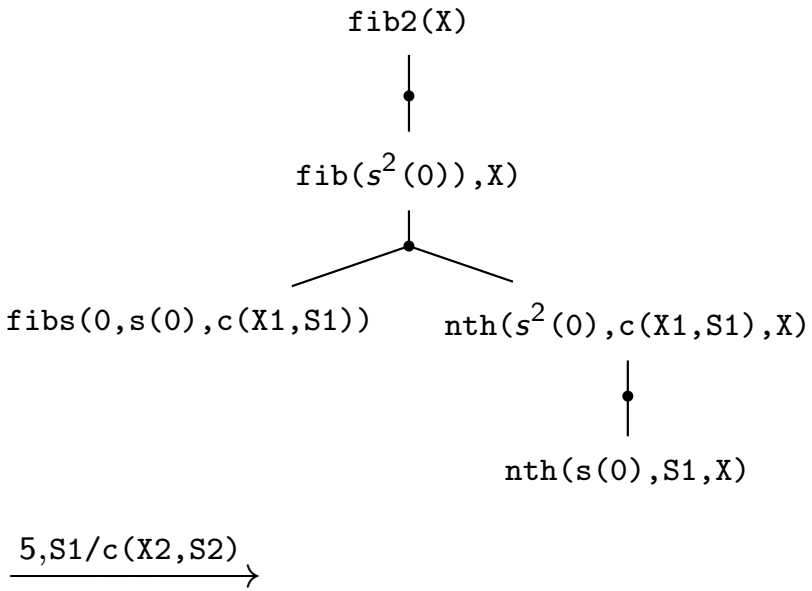


Examples of derivations with Fib: lazy step 1

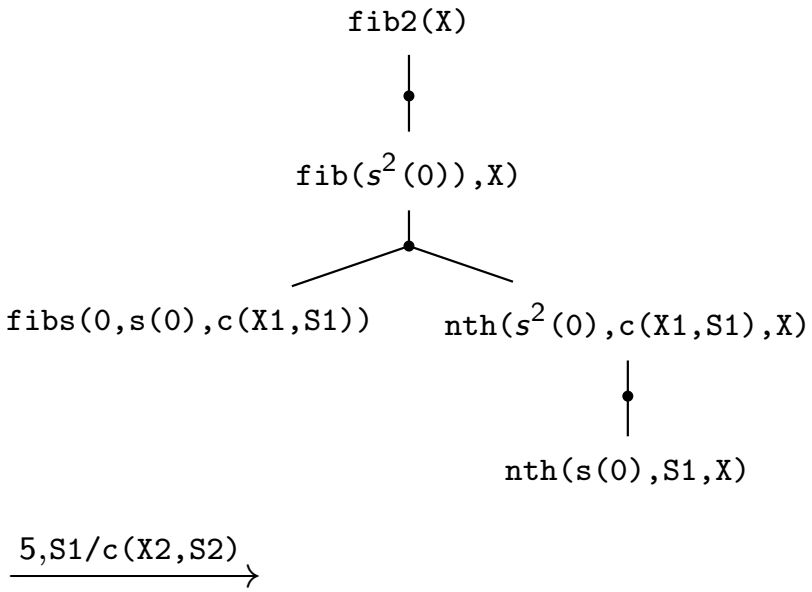



```
1.  add(0,Y,Y).
2.  add(s(X),Y,s(Z)) :-
add(X,Y,Z).
3.  fibs(X,Y,cons(X,S)) :-
add(X,Y,Z), fibs(Y,Z,S).
4.  nth(0,cons(X,S),X).
5.  nth(s(N),cons(X,S),Y) :-
nth(N,S,Y).
6.  fib(N,X) :- fibs(0,s(0),S),
nth(N,S,X).
7.  fib2(X) :- fib(s(s(0)),X).
```

Examples of derivations with Fib: lazy step 2

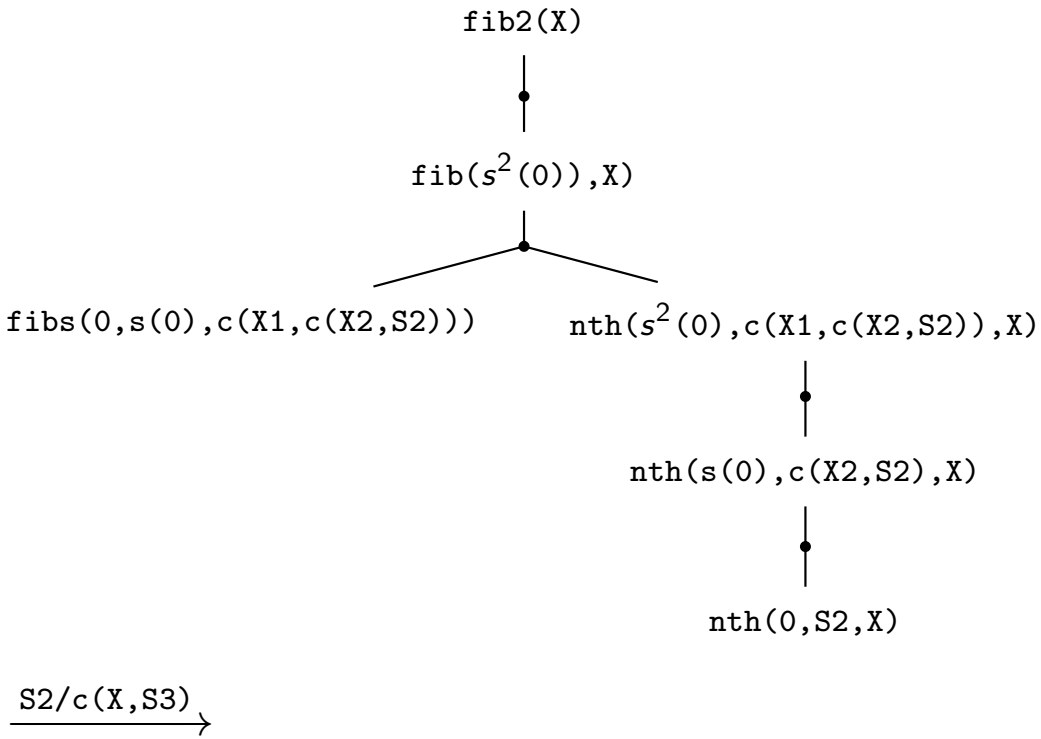


Examples of derivations with Fib: lazy step 2

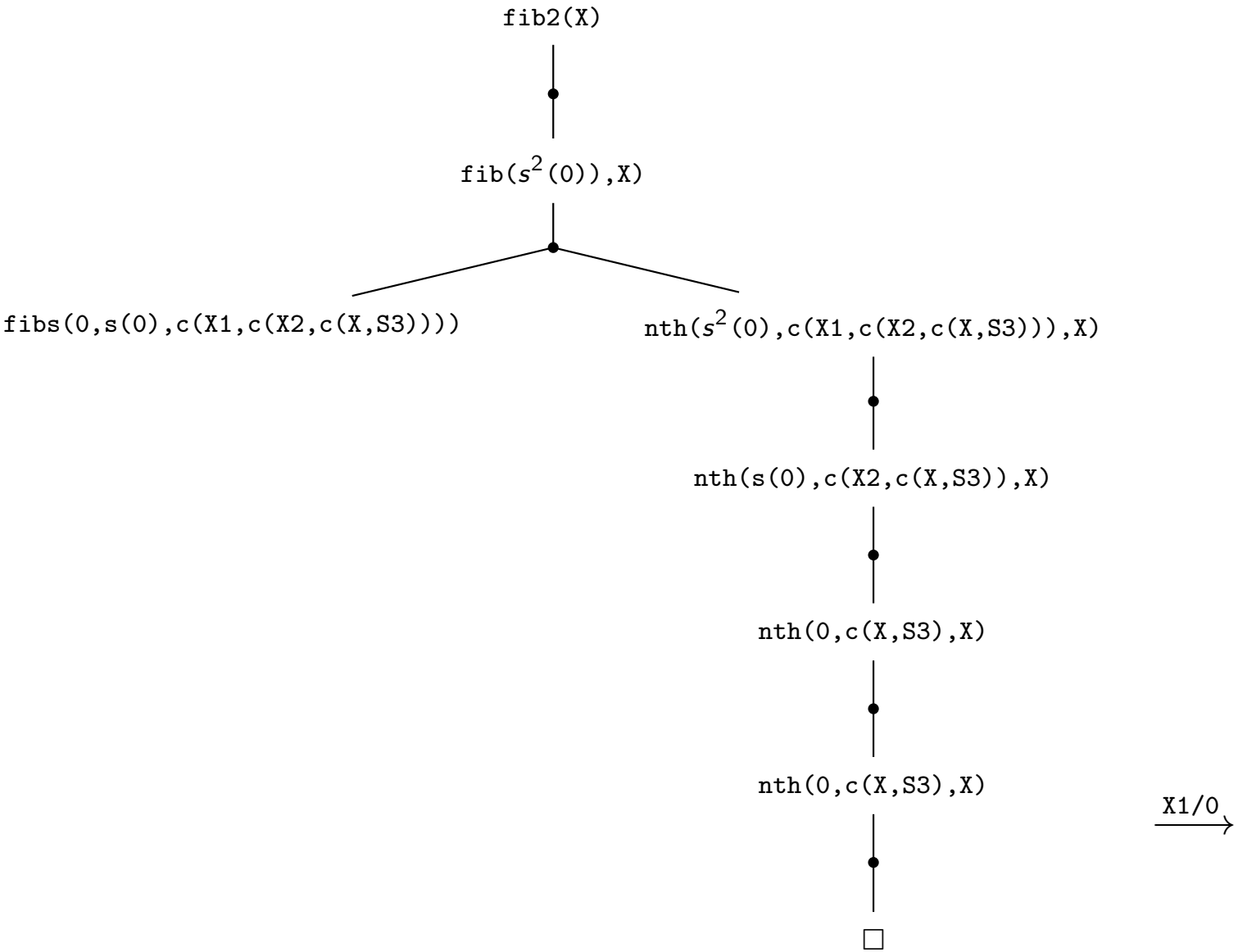



```
1.  add(0,Y,Y).
2.  add(s(X),Y,s(Z)) :-
add(X,Y,Z).
3.  fibs(X,Y,cons(X,S)) :-
add(X,Y,Z), fibs(Y,Z,S).
4.  nth(0,cons(X,S),X).
5.  nth(s(N),cons(X,S),Y) :-
nth(N,S,Y).
6.  fib(N,X) :- fibs(0,s(0),S),
nth(N,S,X).
7.  fib2(X) :- fib(s(s(0)),X).
```

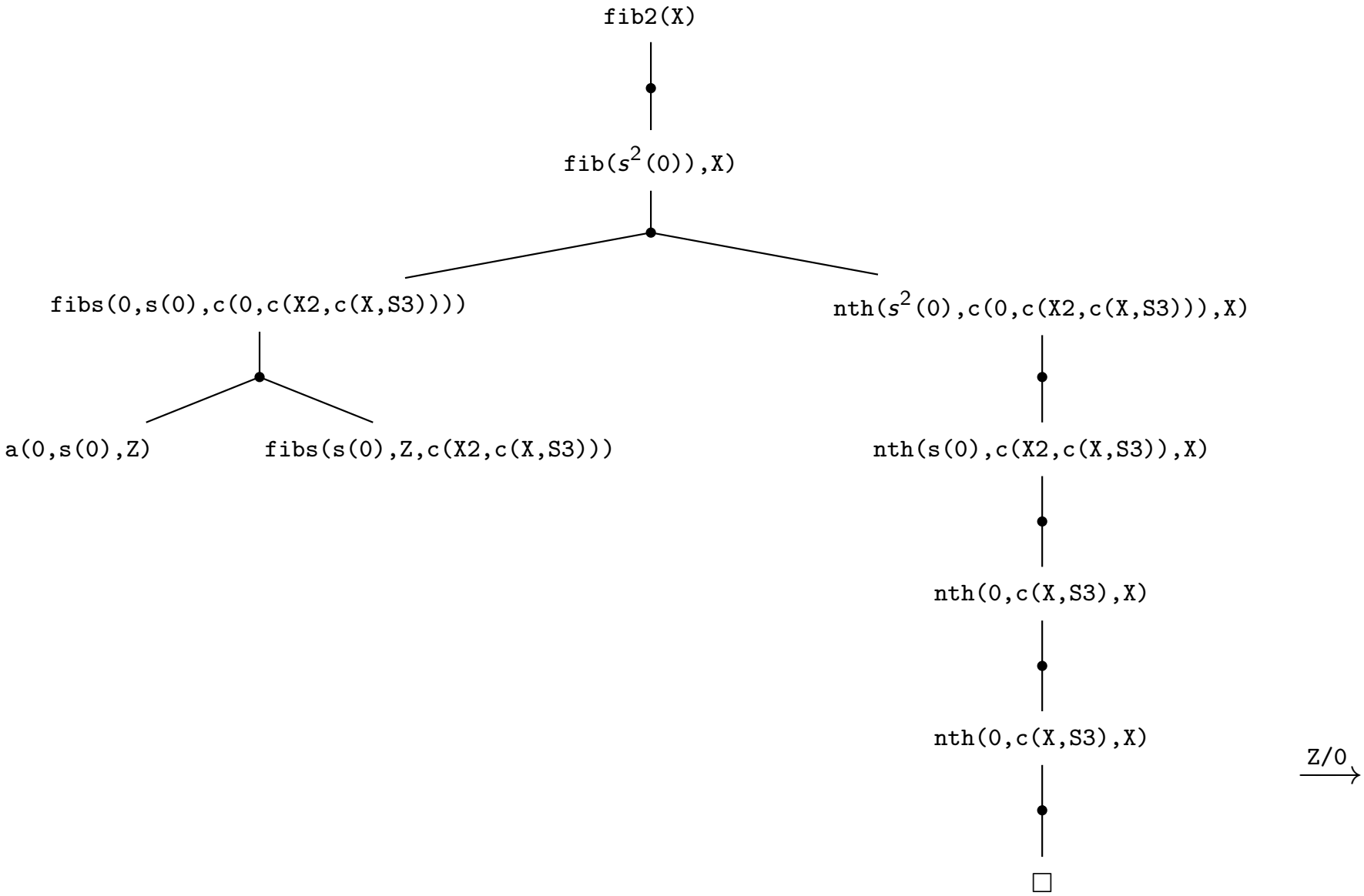
Examples of derivations with Fib: lazy step 3



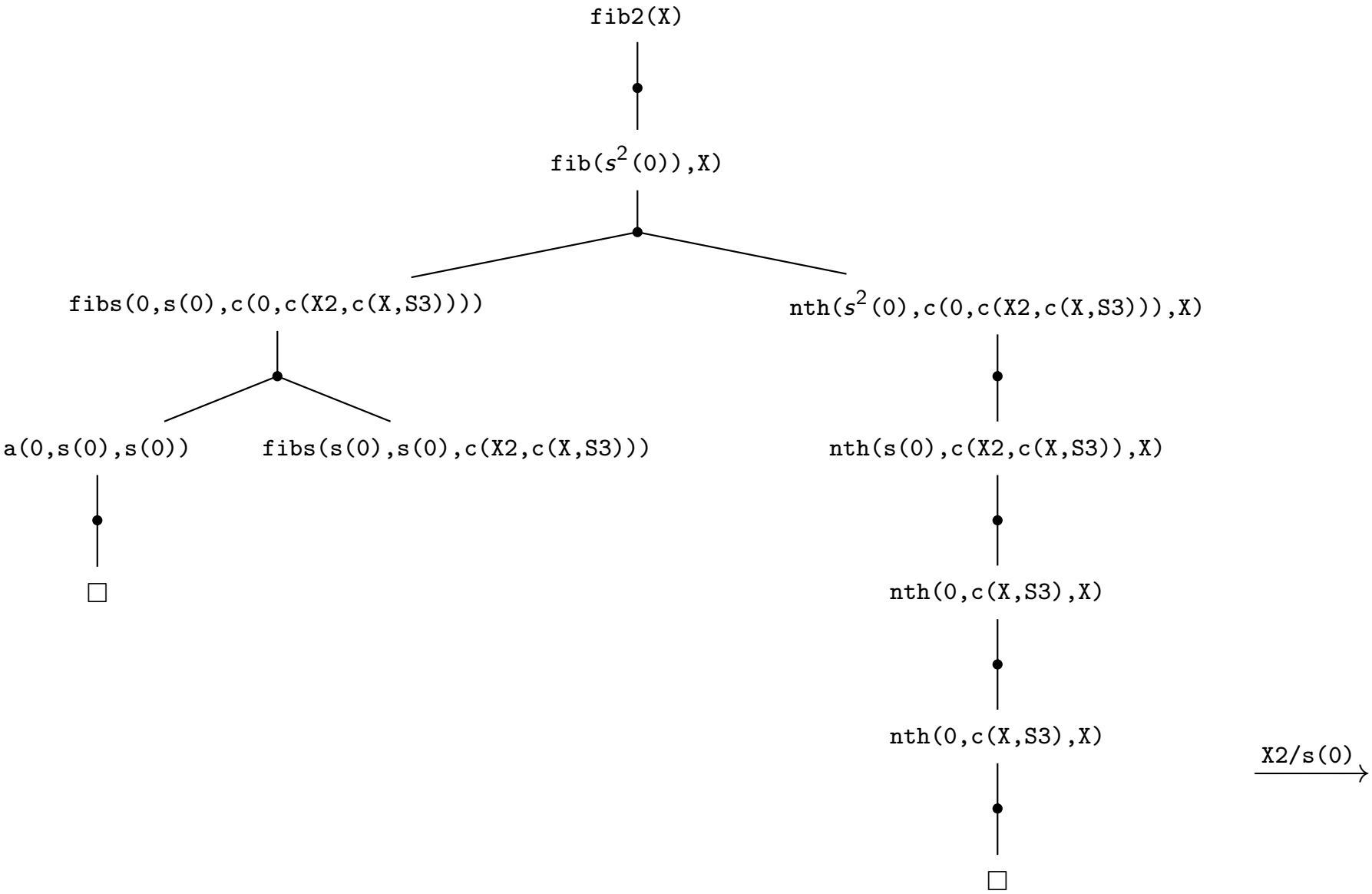
Examples of derivations with Fib: lazy step 4



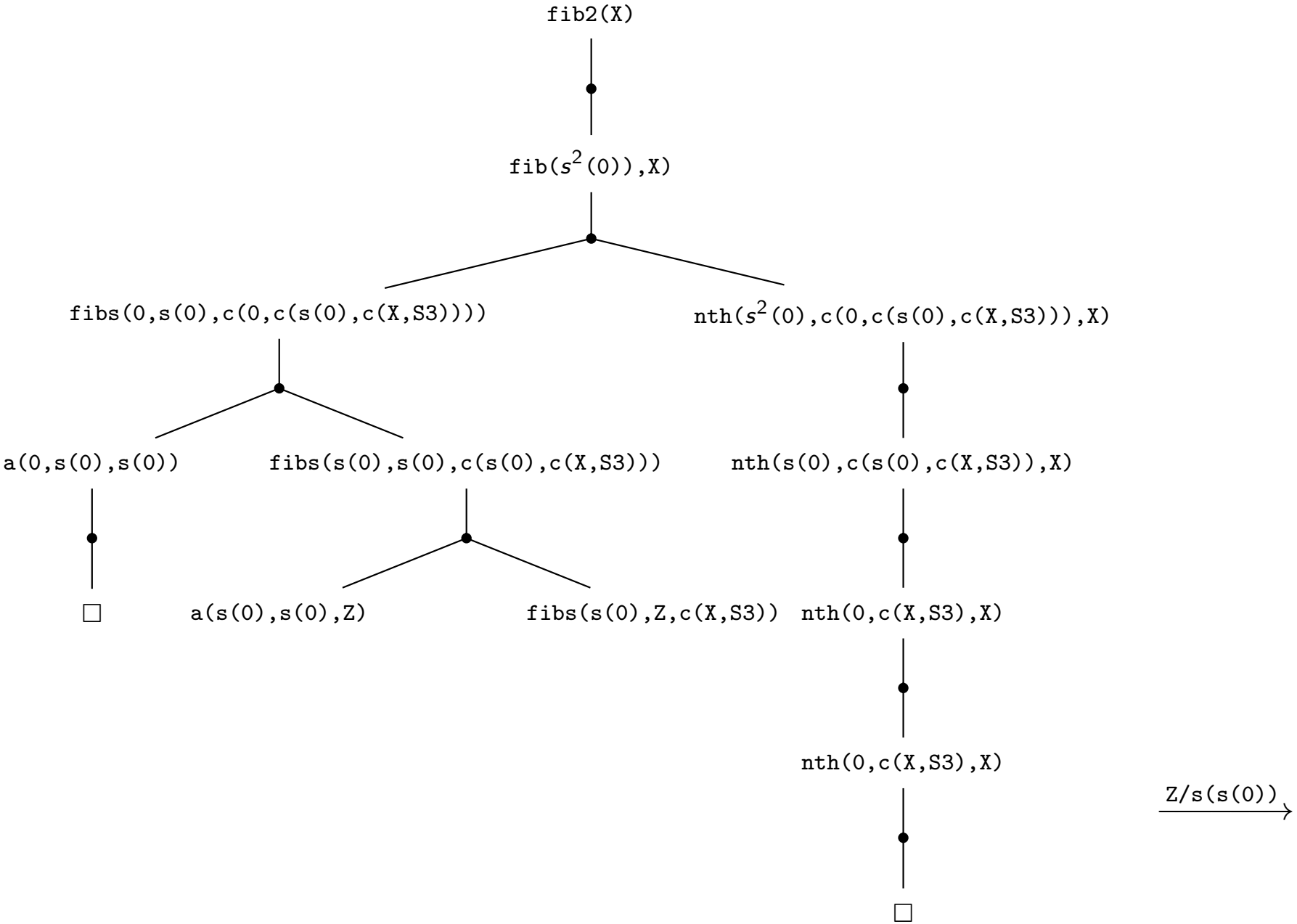
Examples of derivations with Fib: lazy step 5



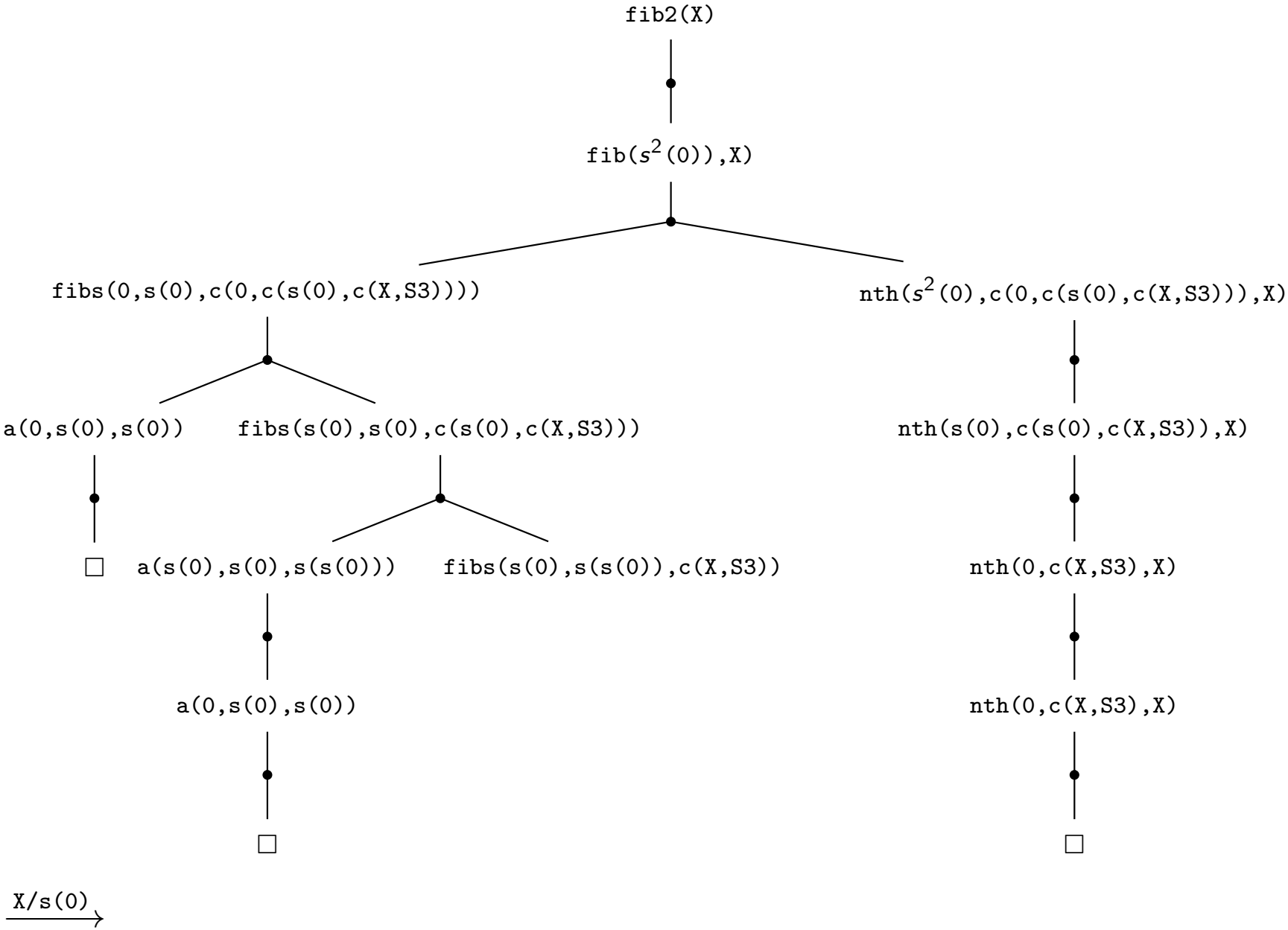
Examples of derivations with Fib: lazy step 6



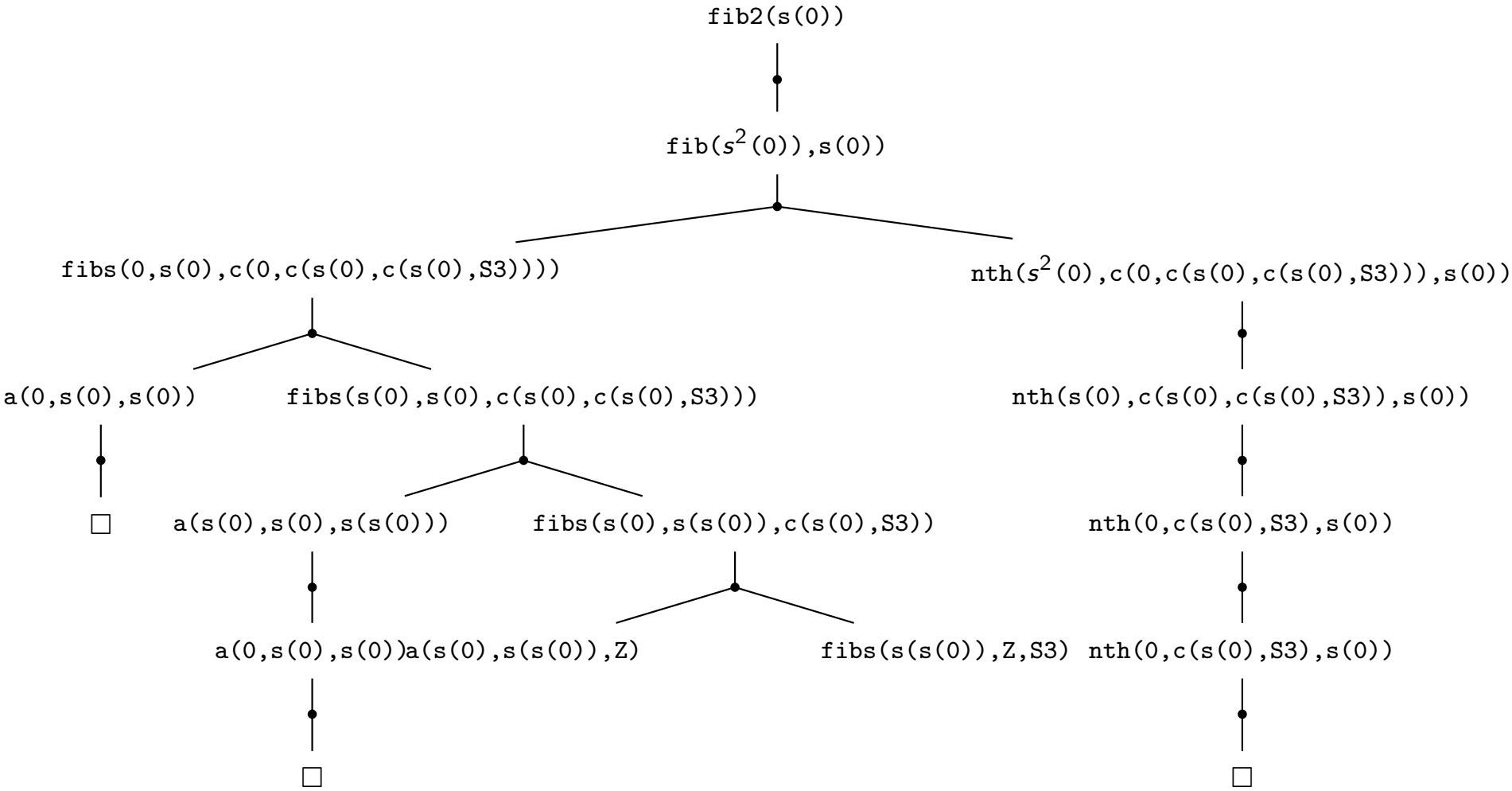
Examples of derivations with Fib: lazy step 7



Examples of derivations with Fib: lazy step 8



Examples of derivations with Fib: lazy step 9



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

Directions we are exploring

- Using CoALP in Hume: for analysis of stream-based networks and/or for type inference [thanks to Hans-Wolfgang for discussions];
- Type-inference in Haskell;
- SSReflect: overloading in canonical structures currently requires the use of back-tracking in LP-like algorithm. It could be parallel CoALP execution instead;
- CoALP for global type analysis in object-oriented languages: CoLP is already used for that.
- ...TBC...

The end

The best reference so far is

- Komendantskaya, Power, Schmidt: Coalgebraic Logic Programming: from Semantics to Implementation, Journal of Logic and Computation, 2014.
- A paper on implementing lazy guarded corecursion in CoALP using Haskell is in preparation...
- CoALP webpage has various prototype implementations to play with... http://staff.computing.dundee.ac.uk/katya/CoALP/

We will be happy to apply CoALP for TI (or other purposes) in *YOUR* language!