Pattern Recognition for Coinductive Proof Trees

Katya Komendantskaya

School of Computing, University of Dundee, UK

AI4FM'11,Edinburgh
29 April 2011

Project CLANN $\cong$ AI4FM in a **radical** form

Computational Logic in Neural Networks

Formal methods: Symbolic Logic, Theorem Provers

- Deduction in logic calculi;
- Logic programming;
- Higher-order proof assistants...

Sound symbolic methods we can trust

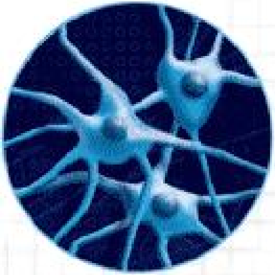
Computational Logic in Neural Networks

Formal methods: Symbolic Logic, Theorem Provers

- Deduction in logic calculi;
- Logic programming;
- Higher-order proof assistants...

Sound symbolic methods we can trust

Neural Networks



- spontaneous behavior (self-organisation);
- learning and adaptation;
- parallel computing.

Learning Heuristics while doing Formal Proofs

The main purpose of this talk:

- Share my experince in machine learning proof tactics using statistical methods (Neural Networks).

- There have been related attempts (but very often do not extend to first-order or higher-order languages with recursion). They are best-suited for languages with finite models. (Objects of these models can then be statistically classified, but dealing with Syntax is generally avoided.)
- Extending these methods to higher-order proofs and recursion may lead to solutions unnatural from machine learning perspective (also called localistic).

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

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

Why is machine learning **UN**-suitable for Formal methods:

- Many logic algorithms have a precise, rather than statistical nature.

Example

Two formulae `list(x)` and `list(nil)` are unifiable: `x/nil`. We mean exactly this, and do not want it to be substituted by some approximate such as `nol`. (Although humans would tolerate this mis-spelling had it appeared in a written text...)

Why is machine learning **UN**-suitable for Formal methods:

- Many logic algorithms have a precise, rather than statistical nature.

Example

Two formulae `list(x)` and `list(nil)` are unifiable: `x/nil`. We mean exactly this, and do not want it to be substituted by some approximate such as `nol`. (Although humans would tolerate this mis-spelling had it appeared in a written text...)

- Many important logic algorithms are sequential, e.g. unification.

Example

If I have a goal: `list(cons(x,y))` $\wedge$ `list(x)`, my proof will never succeed — `x` will get substituted by some `nat` term, e.g. `0` or `S(0)`, which will make the second formula invalid. Note that the proof would have succeed had it been concurrent.

Any Hope?

However,

- Many proofs, especially by (co-)induction, especially using constructors (like `nil` or `cons`), share some common structure (= follow some patterns in machine learning terms) that can be detected using statistical learning;
- We have concurrent algorithms for proof search to implement - e.g. coinductive proof trees [Komendantskaya,Power CALCO'2011].

What are the coinductive trees?

- They arose from coalgebraic semantics for derivations in logic programs, [Komendantskaya,Power CALCO'2011].
- They offer a proof method for recursive and corecursive logic programs.
- They also allow for concurrency.
- They offer very structured approach to automated proofs, as we will see shortly.

What are the coinductive trees?

- They arose from coalgebraic semantics for derivations in logic programs, [Komendantskaya,Power CALCO'2011].
- They offer a proof method for recursive and corecursive logic programs.
- They also allow for concurrency.
- They offer very structured approach to automated proofs, as we will see shortly.

Can we learn what they are from positive and negative examples?

Examples of a first-order coinductive trees:

Correct

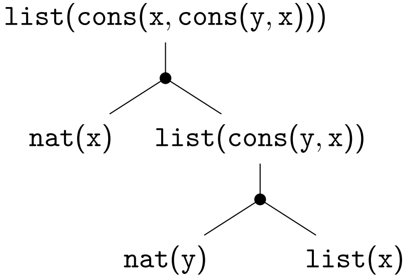
$$\mathtt{list}(\mathtt{cons}(\mathtt{x}, \mathtt{cons}(\mathtt{y}, \mathtt{x})))$$

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

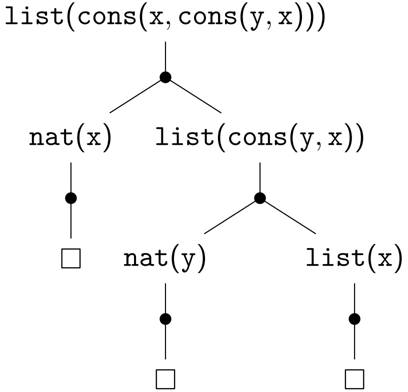
$$\mathtt{nat}(\mathtt{y}) \qquad \mathtt{list}(\mathtt{x})$$

Examples of a first-order coinductive trees:

Correct

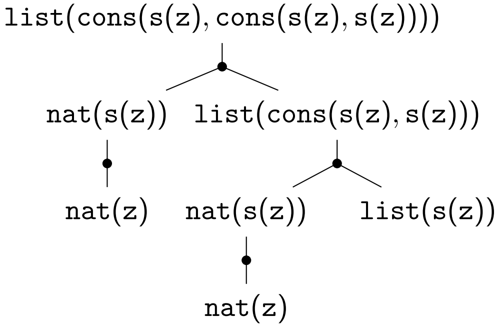


Incorrect



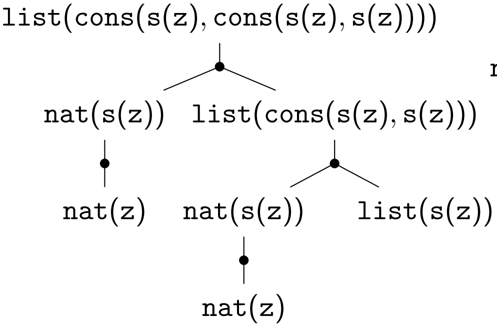
Examples of first-order coinductive trees:

Correct

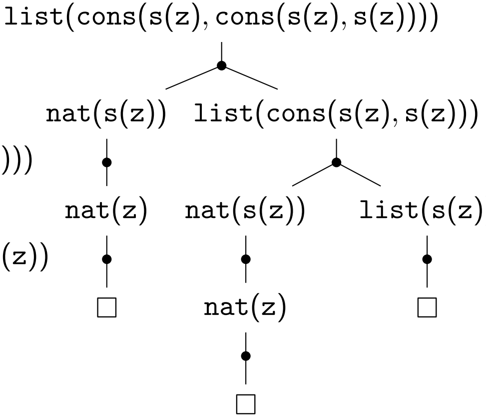


Examples of first-order coinductive trees:

Correct

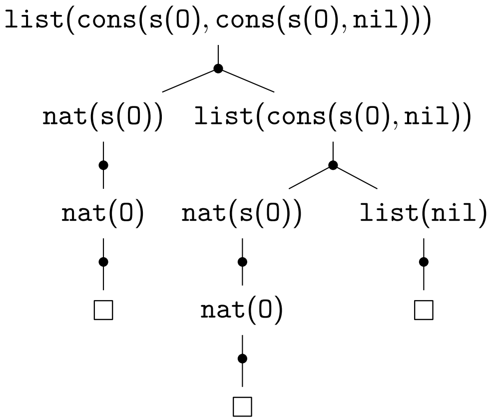


Incorrect



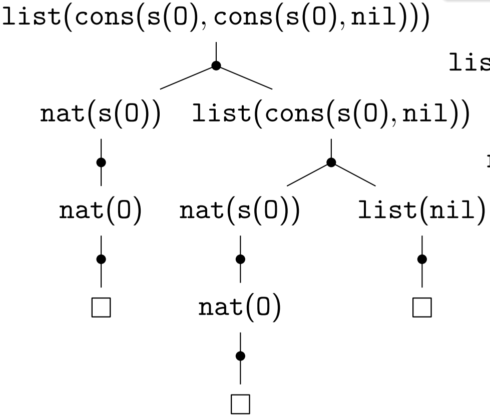
Examples of first-order coinductive trees:

Correct

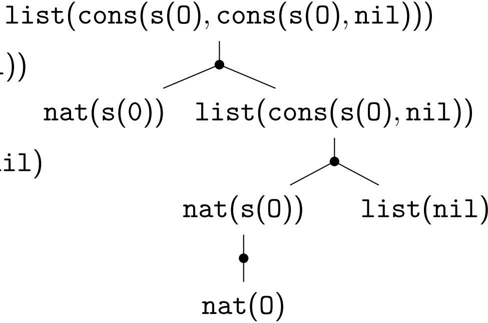


Examples of first-order coinductive trees:

Correct



Incorrect



Test examples: Is this a correct CPT?

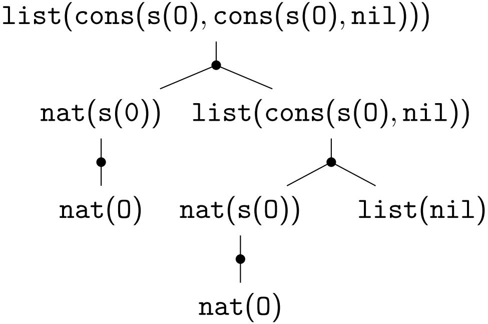
$\mathtt{nat(cons(s(0), cons(s(0), nil)))}$

Test examples: Is this a correct CPT?

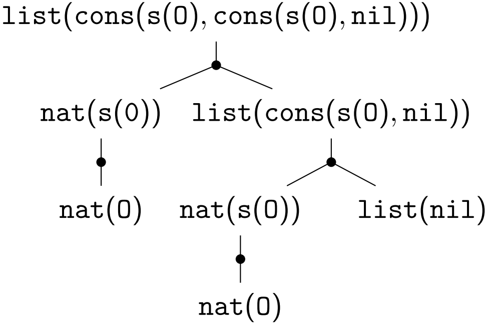
$\texttt{nat}(\texttt{cons}(\texttt{s}(\texttt{0}), \texttt{cons}(\texttt{s}(\texttt{0}), \texttt{nil})))$

Yes.

Test examples: Is this a correct CPT?

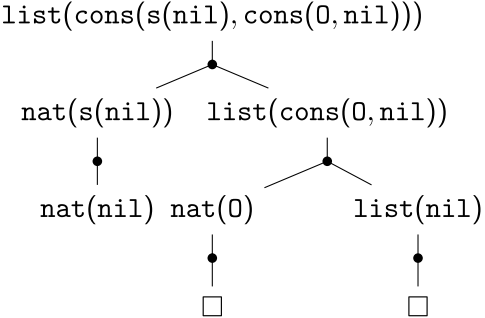


Test examples: Is this a correct CPT?

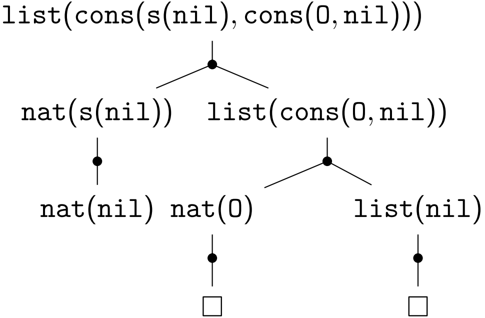


No.

Test examples: Is this a correct CPT?

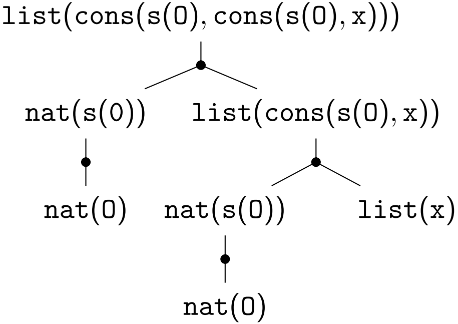


Test examples: Is this a correct CPT?

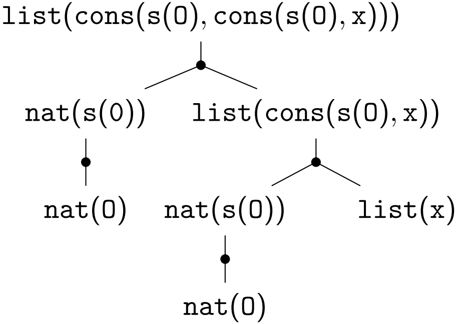


Yes.

Test examples: Is this a correct CPT?



Test examples: Is this a correct CPT?



No.

MATLAB Demo

Our Result Versus Neural Nets result

Our results

Training set:
Testing examples: %.

Our Result Versus Neural Nets result

Our results

Training set:
Testing examples: %.

Neural Network

Training set: - roughly 82%.
(in the demo - 94%)
Testing examples: - 100%.

Different properties can be separated

We can ask questions about coinductive trees proving `nat` and `list`, and can tune neural networks to classify the results into four classes:

- correct-for-list
- incorrect-for-list
- correct-for-nat
- incorrect-for-nat

Future work

- Find an interesting application
- Note on Finding "Why?s": my experience with Coq in INRIA was that the experts often justify those "Why's" statistically rather than conceptually.
- Note on combinations of tactics, like `intro`, `induction`, `simpl`, and so on in Higher-order interactive provers. Even very complicated proofs use about 50-100 tactics only; BUT their combinations can be very clever. Again, is there room for statistical analysis?
- We have learned *both* from positive and negative examples, as discussed yesterday.

Future work

- Find an interesting application
- Note on Finding "Why?s": my experience with Coq in INRIA was that the experts often justify those "Why's" statistically rather than conceptually.
- Note on combinations of tactics, like `intro`, `induction`, `simpl`, and so on in Higher-order interactive provers. Even very complicated proofs use about 50-100 tactics only; BUT their combinations can be very clever. Again, is there room for statistical analysis?
- We have learned *both* from positive and negative examples, as discussed yesterday.

Really GOOD news

The approach is 100% natural from machine learning perspective: uses standard nets and not localistic.

Questions?

(Please contact me katya@computing.dundee.ac.uk if they arise later!)