Coinductive Logic Programming in Logtalk

Paulo Moura
Dep. of Computer Science, Univ. of Beira Interior, Portugal
Center for Research in Advanced Computing Systems
INESC TEC, Portugal
pmoura@logtalk.org

http://logtalk.org/
Logtalk is...

• ... an object-oriented logic programming language
• ... an highly portable Prolog extension
• ... focused on code encapsulation and code reuse mechanisms
  • ... the best thing since sliced bread (subliminal text for your subconscious pleasure)
Why support coinduction?

• Enhance Logtalk as a strong framework for solving complex problems where coinduction is one of the solution components

• Promote a portable and thus widely available coinduction implementation

• Help to push for improved Prolog support for rational terms, tabling, constraint libraries

  • It’s fun! Finite terms and automata are so boring ... plus, you don’t have to think about base cases!
The Logtalk implementation of coinduction ...

- ... supports ECLiPSe, SICStus Prolog, SWI-Prolog, and YAP as back-end Prolog compilers
- ... supports debugging of coinductive predicates
- ... includes a set of examples and corresponding unit tests
  - ... boldly goes where no implementation of coinduction have gone before
Beauty is skin deep

- What the user writes:

```prolog
:- object(binary).

:- public(p/1).
:- coinductive(p/1).

p([0| T]) :- p(T).
p([1| T]) :- p(T).

:- end_object.
```
Beauty is skin deep

- What the Logtalk compiler generates:

```prolog
p_1_coinduction_preflight(A, Stack) :-
  ( member(p(A), Stack) *->
    true
  ;  p(A, [p(A)| Stack])
  ).

p([0| A], Stack) :-
  p_1_coinduction_preflight(A, Stack).
p([1| A], Stack) :-
  p_1_coinduction_preflight(A, Stack).
```

Segunda-feira, 10 de Setembro de 12
Limitations ...

• ... we can only *generate* *basic cycles*
• ... but we can *recognize* any valid solution
• ... only SWI-Prolog prints bindings containing rational terms unambiguously
• ... no support yet for tabling of rational terms
• .... lack of standards for constraint libraries
• Rats! This was going so well ...
Basic cycles?!?

?- binary::p(X).
X = [0|X] ;
X = [1|X] ;
false.

?- L = [0,1,0| L], binary::p(L).
L = [0, 1, 0|L] ;
false.

• Why is theoretical people moving unsettlingly in their chairs?
Is the alternative worthy?

?- binary::p(X).
X = [0|X] ;
X = [0|_S1], % where
    _S1 = [0|_S1] ;
X = [0, 0|X] ;
X = [0, 0|_S1], % where
    _S1 = [0|_S1] ;
X = [0|_S1], % where
    _S1 = [0, 0|_S1] ;
X = [0, 0, 0|X] ;
...

•  There! Satisfied? Hope so; you will be pressing ";" for a long time ...
# Printing bindings

<table>
<thead>
<tr>
<th>Prolog compiler</th>
<th>First solution</th>
<th>Second solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECLiPSe 6.1.115</td>
<td>( X = [0, 0, 0, 0, \ldots] )</td>
<td>( X = [1, 1, 1, 1, \ldots] )</td>
</tr>
<tr>
<td>SICStus Prolog 4.0.4</td>
<td>( X = [0, 0, 0, 0, \ldots] )</td>
<td>( X = [1, 1, 1, 1, \ldots] )</td>
</tr>
<tr>
<td>SWI-Prolog 6.1.11</td>
<td>( X = [0</td>
<td>X] )</td>
</tr>
<tr>
<td>YAP 6.3.2</td>
<td>( X = [0</td>
<td>**] )</td>
</tr>
</tbody>
</table>

* Thank the gods for the print depth limit write option!
Tabling of rational terms where art thou?

\[\text{- coinductive}(\text{comember}/2).\]

\[\text{comember}(X, L) :-\]
\[\text{drop}(X, L, L1),\]
\[\text{comember}(X, L1).\]

\[\text{- table}(\text{drop}/3).\]

\[\text{drop}(H, [H| T], T).\]
\[\text{drop}(H, [ _| T], T1) :-\]
\[\text{drop}(H, T, T1).\]

- But... if simple things like this don’t work ... and there’s no money back guarantee ... damn!
Constraints babel tower

- No syntax standards for constraint libraries
- Do we even have the same exact semantics?
- Conditional compilation directives as a workaround
  - Copy–paste–modify programming!
My master piece... err... my first coinductive example

:- object(sieve).

  :- public(primes/2).
  % computes a coinductive list with all the primes in the 2..N interval
  primes(N, Primes) :-
      generate_infinite_list(N, List),
      sieve(List, Primes).

  % generate a coinductive list with a 2..N repeating pattern
  generate_infinite_list(N, List) :-
      sequence(2, N, List, List).

  • As the Sieve of Eratosthenes coroutining example in the ICLP 2007 paper in coinduction coconfuses my mind!
sequence(Sup, Sup, [Sup | List], List) :- !.
sequence(Inf, Sup, [Inf | List], Tail) :-
    Next is Inf + 1,
    sequence(Next, Sup, List, Tail).

:- coinductive(sieve/2).
sieve([H | T], [H | R]) :-
    filter(H, T, F),
    sieve(F, R).

:- coinductive(filter/3).
filter(H, [K | T], L) :-
    (    K > H, K mod H =:= 0 ->
        % throw away the multiple we found
        L = T1
    ;
        % we must not throw away the integer used for
        % filtering in order to return a filtered
        % coinductive list
        L = [K | T1]
    ),
    filter(H, T, T1).

:- end_object.
Conclusions

• Widely available and portable implementation
• Basic debugging support
• Several examples complemented by unit tests
• Useful for demoing the basic ideas of coinductive logic programming in the classroom
• But also useful for solving actual problems
Future work

• Prove the intuition behind the idea of basic cycles

• Find how to derive an expression that represents the combination of basic cycles and that can be used for checking or generating any solution?

• Keep pace with the progress on the theoretical aspects of coinduction

• Arm–twist Prolog implementers for better support for rational terms and tabling