Online Appendix of: Tabling, Rational Terms, and Coinduction Finally Together!

THEOFRASTOS MANTADELIS, RICARDO ROCHA and PAULO MOURA

CRACS & INESC TEC, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
(*e-mail:* {theo.mantadelis,ricroc}@dcc.fc.up.pt,pmoura@inescporto.pt)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

YAP Installation and Rational Term Support

At the time of this submission, our contributions are part of the development version of YAP (`git clone git://git.code.sf.net/p/YAP/YAP-6.3`). Inside the folder `YAP-6.3/` the `ICLP2014_examples.YAP` file contains the paper examples using the appropriate tabling settings. Currently, tabling supports rational terms only when the answers are loaded by the tries. To activate that option one can use the following YAP flag:

```
yap_flag(tabling_mode, load_answers).
```

Furthermore, our coinductive transformation can be activated by the following directives:

```
:- table PREDICATE/ARITY.
:- tabling_mode(PREDICATE/ARITY, coinductive).
```

Technical Requirements: Our tabling extensions only require that the Prolog system allows the creation of rational terms and that unification (`=/2`) works between rational terms. Our `canonical_term/2` predicate also requires that the operators `==/2`, `=../2` work with rational terms.

Appendix A Coinduction Examples

The following examples are recreations of the examples presented at (Moura 2013) by using our implementation.

```
:- table(comember/2).
:- tabling_mode(comember/2, coinductive).

% Returns the infinite members of a list.
comember(H, L) :-
  drop(H, L, L1),
  comember(H, L1).

:- table(drop/3).
drop(H, [H|T], T).
drop(H, [_|T], T1) :- drop(H, T, T1).

% Queries:
```

```
?- _L=[1,2|_B], _B=[3,4,5|_B], comember(E, _L).
E = 3 ? ;
E = 4 ? ;
E = 5 ? ;
false.

?- comember(1, A).
A = [1|A] ? ;
A = [_1,1,_1|A] ? ;
A = [_1,_2,1,_1|A] ? ;
A = [_1,_2,_3,1,_1|A] ?
...

?- A=[1,2,3|A], drop(H, A, T).
A = [1,2,3|A],
H = 1,
T = [2,3,1|T] ? ;
A = [1,2,3|A],
H = 2,
T = [3,1,2|T] ? ;
A = T = [1,2,3|A],
H = 3 ? ;
false.

?- B=[1|A],A=[2,3|A], drop(H, B, T).
A = T = [2,3|A],
B = [1,2,3|B],
H = 1 ? ;
A = [2,3|A],
B = [1,2,3|B],
H = 2,
T = [3,2|T] ? ;
A = T = [2,3|A],
B = [1,2,3|B],
H = 3 ? ;
false.

?- A=[1,2,3|A], member(H, A).
A = [1,2,3|A],
H = 1 ? ;
A = [1,2,3|A],
H = 2 ? ;
A = [1,2,3|A],
H = 3 ? ;
A = [1,2,3|A],
H = 1 ? ;
A = [1,2,3|A],
H = 2 ?
...

?- B=[1|A],A=[2,3|A], member(H, B).
A = [2,3|A],
B = [1,2,3|B],
H = 1 ? ;
A = [2,3|A],
B = [1,2,3|B],
H = 2 ? ;
A = [2,3|A],
B = [1,2,3|B],
H = 3 ? ;
A = [2,3|A],
B = [1,2,3|B],
H = 2 ? ;
A = [2,3|A],
B = [1,2,3|B],
H = 3 ?
```

...

```
:- table(p/1).
:- tabling_mode(p/1, coinductive).

:- table(q/1).
:- tabling_mode(q/1, coinductive).

:- table(r/1).
:- tabling_mode(r/1, coinductive).

% Tangle example.
p([a|X]) :- q(X).
p([c|X]) :- r(X).
q([b|X]) :- p(X).
r([d|X]) :- p(X).

% Queries:
?- p(X).
X = [a,b|X] ? ;
X = [c,d|X].

?- L = [a,b,c,d|L], p(L).
L = [a,b,c,d|L].

?- L = [a,c|L], p(L).
false.
```

```
:- table(automaton/2).
:- tabling_mode(automaton/2, coinductive).

% Automaton example.
automaton(State, [Input|Inputs]) :-
  trans(State, Input, NewState),
  automaton(NewState, Inputs).

trans(s0, a, s1).
trans(s1, b, s2).
trans(s2, c, s3).
trans(s2, e, s0).
trans(s3, d, s0).

% Queries:
?- automaton(s0, X).
X = [a,b,c,d|X] ? ;
X = [a,b,e|X].

?- L = [a,b,c,d,a,b,e|L], automaton(s0, L).
L = [a,b,c,d,a,b,e|L].

?- L = [a,b,e,c,d|L], automaton(s0, L).
false.
```

```
:- table(sieve/2).
:- tabling_mode(sieve/2, coinductive).

:- table(filter/3).
:- tabling_mode(filter/3, coinductive).

% 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 patern
generate_infinite_list(N, List) :-
  sequence(2, N, List, List).

sequence(Sup, Sup, [Sup| List], List) :-
  !.
sequence(Inf, Sup, [Inf| List], Tail) :-
  Next is Inf + 1,
  sequence(Next, Sup, List, Tail).

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

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
     % as we must return a filtered coinductive list
     L = [K| T1]
  ),
  filter(H, T, T1).

% Queries:
?- primes(20, P).
P = [2,3,5,7,11,13,17,19,2,3,5,7,11,13,17,19,2,3|P].
```

Appendix B Experiments

We used the following example program in order to run experiments.

```
:- table(path/2).
:- tabling_mode(path/2, coinductive).

% Finds infinite paths starting from node F.
path(F, [F|P]) :-
  edge(F, N),
  path(N, P).

edge(1, 2).
edge(1, 3).
edge(2, 4).
edge(2, 3).
edge(3, 2).

% Queries:
?- path(1, P).
P = [1,2,3|P] ? ;
P = [1,3,2|P].

?- path(2, P).
P = [2,3|P].

?- path(3, P).
P = [3,2|P].

?- path(4, P).
```

```
false.
```

Instead of the small graph shown above, we used a fully connected graph of different sizes, as presented next:

```
full_edge_size(8).

edge(X, Y) :-
  posint(X),
  posint(Y),
  X \== Y.

posint(N) :-
  posint(N, 0).
posint(_, I) :-
  full_edge_size(N),
  I > N, !,
  fail.
posint(I, I).
posint(X, I) :-
  NI is I + 1,
  posint(X, NI).
```

And implemented the same program in Logtalk:

```
:- object(path).
  :- public(path/2).
  :- coinductive(path/2).
%  :- table(path/2). % used for tabled co-SLD

  path(F, [F|P]) :-
    edge(F, N),
    path(N, P).

  full_edge_size(8).

  edge(X, Y) :-
    posint(X),
    posint(Y),
    X \== Y.

  posint(N) :-
    posint(N, 0).
  posint(_, I) :-
    full_edge_size(N),
    I > N, !,
    fail.
  posint(I, I).
  posint(X, I) :-
    NI is I + 1,
    posint(X, NI).
:- end_object.
```

We executed the query: `time((path(1, _P), fail))` on graphs with size 8x8 up to 19x19. Table B 1 presents our results; all times are in seconds. For co-SLD we used the coinductive transformation of Logtalk. For co-SLG we used our transformation. Furthermore, we also experimented with the `path/2` coinductive predicate in Logtalk being tabled with our rational term support. This can be achieved by just tabling the co-SLD transformed predicate of Logtalk as shown at the comment instruction of the Logtalk code.

Table B 1. *Experimental results for the query* `(path(1, _P), fail)`

Graph Size	co-SLD	co-SLG	Tabled co-SLD
8x8	1	0.005	4
9x9	11	0.014	*
10x10	126	0.035	
11x11	1,724	0.053	
12x12	> 324,000	0.126	
13x13		0.287	
14x14		0.674	
15x15		1.5	
16x16		3.5	
17x17		8	
18x18		17	
19x19		39	
20x20		*	

As expected the difference for the path example for co-SLD and co-SLG is significant. This is easy to explain as tabling in this case decreases the complexity of the problem. co-SLD is able to solve up to an 11x11 graph in approximately 1,700 seconds, and it takes more than 1.5 hours which was used as our timeout to solve the 12x12 problem. On the other hand, enumerating all the cyclic paths with co-SLG is speed efficient but very memory consuming. Our system exhausted the 10GB memory that was available to it trying to calculate the 20x20 graph. We also executed experiments by tabling the co-SLD approach. These experiments, which obtained the worst results, are important in order to point out that co-SLG is not simply achieved by tabling a co-SLD transformed predicate. The tabled co-SLD predicate soon went out of memory as it had the task to table both the resulting paths and the coinductive hypothesis. Unfortunately, we could not ignore the coinductive hypothesis from being tabled due to a limitation of YAP's mode-directed tabling (Santos and Rocha 2012).

It is well known that using tabling naively can result on a poor performance; for example tabling `append/2` predicate as shown in (Swift and Warren 2012). The same applies for using co-SLG over co-SLD naively. There are examples that co-SLD will perform better than co-SLG like the Eratosthenes sieve example.

References

Moura, P. 2013. A Portable and Efficient Implementation of Coinductive Logic Programming. In *International Symposium on Practical Aspects of Declarative Languages*. LNCS, vol. 7752. Springer-Verlag, 77–92.

Santos, J. and Rocha, R. 2012. Mode-Directed Tabling and Applications in the YapTab System. In *Symposium on Languages, Applications and Technologies*. 25–40.

Swift, T. and Warren, D. S. 2012. XSB: Extending Prolog with Tabled Logic Programming. *Theory and Practice of Logic Programming 12,* 1 & 2, 157–187.