CS476 - Homework 5

Nishant Rodrigues

Check termination using ACRPO

list-check-termination.maude

```
--- Disable Prelude's bool so that acrpo can define it's own.
set include BOOL off .

--- Define an order on the function symbols
fmod TEST is
  sorts Element List .
  subsorts Element < List .

  sort U .
  subsorts List < U .

  op nil : -> List                          [ctor metadata "1"] .
  op a : -> Element                         [ctor metadata "2"] .
  op b : -> Element                         [ctor metadata "3"] .
  op c : -> Element                         [ctor metadata "4"] .

  op _;_  : Element List -> List            [ctor metadata "5"] .
  op _;_  : List List -> List               [     metadata "5"] .

endfm

--- Finally, check that each equation rewrites it's LHS to a "lesser" RHS.
load ../acrpo-tool-distribution/acrpo.maude
reduce (L:List ; P:List) ; Q:List >AC L:List ; (P:List ; Q:List) .
reduce L:List ; nil               >AC L:List .
reduce nil ; L:List               >AC L:List .

quit
```

Check confluence using Church-Rosser checker

list-crc.maude

```
load /home/njr/co/github/maude-team/mfe/src/mfe.maude .
(set include BOOL off .)
load list-example.maude
(select tool CRC .)
(check Church-Rosser .)
quit
```

Check sufficient completeness

list-scc.maude

```
load scc.maude
loop init-cc .

fmod LIST-EXAMPLE is
  sorts Element List .
  subsorts Element < List .
  op a : -> Element [ctor] .
  op b : -> Element [ctor] .
  op c : -> Element [ctor] .
  op nil : -> List [ctor] .

  op _;_  : List List -> List .
  op _;_  : Element List -> List [ctor] .
  eq (L:List ; P:List) ; Q:List = L:List ; (P:List ; Q:List) .
  eq L:List ; nil = L:List .
  eq nil ; L:List = L:List .
endfm

select CC-LOOP .
(scc LIST-EXAMPLE .)

quit
```

Check local confluence manually

list-local-confluence.maude

```
fmod LIST-EXAMPLE is
  sorts Element List .
  subsorts Element < List .
  op a : -> Element [ctor] .
  op b : -> Element [ctor] .
  op c : -> Element [ctor] .
  op nil : -> List [ctor] .

  op _;_  : List List -> List .
  op _;_  : Element List -> List [ctor] .
  eq (L:List ; P:List) ; Q:List = L:List ; (P:List ; Q:List) .     --- (1)
  eq L:List ; nil = L:List .                                       --- (2)
  eq nil ; L:List = L:List .                                       --- (3)
endfm

--- Unifing (1) with (1) after renaming
unify (L:List ; M:List) ; N:List =? (O:List ; P:List) ; Q:List  .

---( gives us:
Solution 1
L:List --> #1:List
M:List --> #2:List
N:List --> #3:List
O:List --> #1:List
P:List --> #2:List
Q:List --> #3:List

giving the critical pair:

    1; (2; 3) , 1; (2 ;3)

which is trivially joinable.
)
```

```
--- Unifing (1) with (2) after renaming
unify (L:List ; P:List) ; Q:List =? R:List ; nil .

---( gives us:

    Solution 1
    L:List --> #1:List
    P:List --> #2:List
    Q:List --> nil
    R:List --> #1:List ; #2:List

Giving us the critical pair:

    #1:List; (#2:List; nil) and (#1:List; #2:List)

that rewrites to:

    (#1:List ; #2:List) and (#1:List; #2:List)

and so is joinable.
)

--- Unifying (1) and (3) after renaming:
unify (L:List ; P:List) ; Q:List =? nil ; M:List                .

---( has no unifier, and generates no critical pairs )

--- Unifying (2) and (2) after renaming:
unify L:List ; nil =? M:List ; nil                              .

---( gives us:

    Solution 1
    L:List --> #1:List
    M:List --> #1:List

Giving us the critical pair:

    #1:List and #1:List

that rewrites to:

    #1:List and #1:List

and so is joinable.
)

--- Unifying (2) and (3) after renaming:
unify L:List ; nil =? nil ; M:List                              .


---( gives us:

    Solution 1
    L:List --> nil
    M:List --> nil

Giving us the critical pair:

    (nil) and (nil)
```

```
that rewrites to:

    (nil) and (nil)

and so is joinable.
)

unify nil ; L:List =? nil ; M:List                                  .

---( gives us:

    Solution 1
    L:List --> #1:List
    M:List --> #1:List

Giving us the critical pair:

    (#1) and (#1)

that rewrites to:

    (#1) and (#1)

and so is joinable.
)

quit
```