Programming Paradigms Fall 2022 — Problem Sets

by Nikolai Kudasov and Khaled Ismaeel

November 9, 2023

1 Problem set №11

Consider the following knowledge base:

```
(defrel (student^o name group)
  (matche (cons name group)
          [(alisa  . 2)]
          [(bob    . 1)]
          [(chloe  . 2)]
          [(denise . 1)]
          [(edward . 2)]))

(defrel (friend^o name1 name2)
  (matche (cons name1 name2)
          [(alisa  . bob)]
          [(alisa  . denise)]
          [(bob    . chloe)]
          [(bob    . edward)]
          [(chloe  . denise)]
          [(denise . edward)]))

(defrel (parent^o parent-name child-name)
  (matche (cons parent-name child-name)
          [(marjorie . bart)]
          [(marjorie . lisa)]
          [(marjorie . maggie)]
          [(homer . bart)]
          [(homer . lisa)]
          [(homer . maggie)]
          [(abraham . homer)]
          [(mona . homer)]
          [(jacqueline . marjorie)]
          [(jacqueline . patty)]
          [(jacqueline . selma)]
          [(clancy . marjorie)]
          [(clancy . patty)]
          [(clancy . selma)]
          [(selma . ling)]))

(defrel (unary^o n)
  (conde
   [(== 'z n)]
   [(fresh (m)
           (== `(s ,m) n)
           (unary^o m))]))
```

1. Write down and explain the results of the following queries:

   (a) `(run* (y z) (friend`^{o}` alisa y) (friend`^{o}` y z))`

   (b) `(run* (friend`^{o}` x y))`

   (c) `(run* (y) (parent`^{o}` jacqueline y) (parent`^{o}` y ling))`

2. Write down relation `groupmates`^{o} that checks whether two students are from the same group.

```
(run 1 (q) (groupmateso 'alisa 'bob))       ; '()
(run 1 (q) (groupmateso 'alisa 'edward))    ; '(_.0)
```

3. Implement predicate `relative`^{o} that checks whether two people are related by blood (share a common ancestor):

```
(run 1 (q) (relativeo 'selma 'patty))       ; '(_.0)
(run 1 (q) (relativeo 'lisa 'ling))         ; '(_.0)
(run 1 (q) (relativeo 'lisa 'selma))        ; '(_.0)
(run 1 (q) (relativeo 'homer 'selma))       ; '()
```

4. Implement the following predicates for unary numbers:

   (a) Implement a predicate `double`^{o} that checks if first number is exactly two times the second:

```
(run 1 (q) (double '(s (s z)) '(s (s (s (s z))))))    ; '(_.0)
(run 1 (x) (double '(s (s z)) x))                      ; '((s (s (s (s z)))))
(run 1 (x) (double x '(s (s (s (s z))))))              ; '((s (s z)))
(run 1 (x) (double x '(s (s (s z)))))                  ; '()
```

   (b) Implement a predicate `leq`^{o} that checks if the first number is less than or equal to the second numbers:

```
(run 1 (q) (leqo '(s (s z)) '(s (s (s (z))))))         ; '(_.0)
(run 1 (q) (leqo '(s (s (s (s z)))) '(s (s (s z)))))   ; '()
```

   (c) Implement multiplication for unary numbers as a predicate `mult`^{o}:

```
(run 1 (x) (multo '(s (s z)) '(s (s (s z))) x)) ; '((s (s (s (s (s (s z)))))))
(run 1 (x) (multo x '(s (s (s z)))  '(s (s (s (s (s (s (z))))))))) ; '((s (s z)))
```

   (d) **(+0.5% extra credit)**
   Implement a predicate `power-of-2`^{o} such that `(power-of-2`^{o}` n m)` is true when $m = 2^n$:

```
(run 1 (q) (power-of-2o '(s (s z)) '(s (s (s (s z))))))   ; '(_.0)
(run 1 (x) (power-of-2o x '(s (s (s z)))))                ; '()
(run 1 (x) (power-of-2o '(s z) x))                        ; '((s (s z)))
(run 3 (x y) (power-of-2o x y))
; '((z (s z))
;   ((s z) (s (s z)))
;   ((s (s z)) (s (s (s (s z))))))
```

   Hint: for the last query to produce each result in finite time, you need to put an upper bound on the second argument, e.g. using `leq`^{o}.