Mathematical logic – coursework 4, exercise 4

László Treszkai

March 26, 2017

4/4. *Give an example of a model that fulfills every axiom of the Robinson arithmetics, and which contains contains two elements that are neither greater than or equal to, nor smaller than or equal to one another; or prove that such a model doesn't exist.*

Let the universe of the $\mathcal{Q}'$ structure be $\omega \cup \{\kappa, \lambda\}$, where $\kappa, \lambda \notin \omega$ and $\kappa \neq \lambda$. $\mathcal{Q}'$ fulfills the axioms of Robinson arithmetics, if we define the $+$ and $\cdot$ operations as follows ($n \in \omega$):

$$\begin{aligned} \kappa \cdot 0 &\coloneqq 0 & \lambda \cdot 0 &\coloneqq 0 \\ n + \kappa &\coloneqq \kappa & n + \lambda &\coloneqq \lambda \\ \kappa + n &\coloneqq \kappa & \lambda + n &\coloneqq \lambda \\ \kappa + \kappa &\coloneqq \kappa & \lambda + \lambda &\coloneqq \lambda \end{aligned}$$

In case $n, m \in \omega$, let $n + m$ and $n \cdot m$ be defined as the result of addition and multiplication of the corresponding natural numbers. Furthermore, a few values can be chosen arbitrarily:

$$\begin{gathered} \kappa + \lambda \coloneqq \kappa \text{ or } \kappa + \lambda \coloneqq \lambda \\ \lambda + \kappa \coloneqq \kappa \text{ or } \lambda + \kappa \coloneqq \lambda \\ n \cdot \kappa \coloneqq \kappa \text{ or } n \cdot \kappa \coloneqq \lambda \\ n \cdot \lambda \coloneqq \kappa \text{ or } n \cdot \lambda \coloneqq \lambda \\ 0 \cdot \kappa \coloneqq 0 \text{ or } 0 \cdot \kappa \coloneqq \kappa \text{ or } 0 \cdot \kappa \coloneqq \lambda \\ 0 \cdot \lambda \coloneqq 0 \text{ or } 0 \cdot \lambda \coloneqq \kappa \text{ or } 0 \cdot \lambda \coloneqq \lambda \end{gathered}$$

With the choice of $\kappa + \lambda \coloneqq \lambda$ and $\lambda + \kappa \coloneqq \kappa$, we get $\neg\exists z(z + \kappa = \lambda)$ and $\neg\exists z(z + \lambda = \kappa)$, so neither $\kappa \leq \lambda$, nor $\lambda \leq \kappa$. ■

We can easily prove that $\mathcal{Q}'$ fulfills all of the required axioms for any two elements, using the above definitions of the operations and the properties of ω. For illustration, below is the proof of $x \cdot (y + 1) = (x \cdot y) + x$, for the case of $x = \kappa$, $y = n \in \omega$:

$$
\begin{aligned}
n \in \omega \;\; &\Rightarrow \;\; n+1 = m \in \omega \\
\kappa * (n+1) &= \kappa * m = \kappa \\
(\kappa * n) + \kappa &= \kappa + \kappa = \kappa
\end{aligned}
$$

The Haskell program listed in the appendix checks whether a finite substructure of $\mathcal{Q}'$ fulfills every axiom, for a given definition of the operations.

Appendix for coursework 4, exercise 4

`robinsonTest.hs`:

```haskell
import NonStdRobinson

main =
    mapM_ ( \(n, phi) ->
        print $ "Axiom " ++ show n ++
            if phi testNumbers
                then " fulfilled"
                else " failed"
    ) (zip [1..] axiomTestList)
        where testNumbers = [Nat 0, Nat 1, Nat 2, Kappa, Lambda]
        -- "Nulla, egy, ketto, sok." Urban J.

axiomTestList = [axiom1, axiom2, axiom3, axiom4, axiom5, axiom6, axiom7]

-- x + 1 != 0
axiom1 :: [CuteNumber] -> Bool
axiom1 xs = all phi xs
    where phi x =  x.+.Nat 1 /= Nat 0

-- (x != 0) -> (exists y: x = y + 1)
axiom2 :: [CuteNumber] -> Bool
axiom2 xs = all phi xs
    where phi x = (x == Nat 0) || any (\y -> x == y.+.Nat 1) xs

-- (x != y) -> (x + 1 != y + 1)
axiom3 :: [CuteNumber] -> Bool
axiom3 xs = all phi (pairs xs)
    where phi (x,y) =  (x == y) || (x.+.Nat 1 /= y.+.Nat 1)

-- x + 0 = x
axiom4 :: [CuteNumber] -> Bool
axiom4 xs = all phi xs
    where phi x =  x.+.Nat 0 == x
```

```
-- x * 0 = 0
axiom5 :: [CuteNumber] → Bool
axiom5 xs = all phi xs
    where phi x =  x.*.Nat 0 == Nat 0

-- x + (y + 1) = (x + y) + 1
axiom6 :: [CuteNumber] → Bool
axiom6 xs = all phi (pairs xs)
    where phi (x,y) =  x.+.(y.+.Nat 1) == (x.+.y).+.Nat 1

-- x * (y + 1) = (x * y) + x
axiom7 :: [CuteNumber] → Bool
axiom7 xs = all phi (pairs xs)
    where phi (x,y) =  x.*.(y.+.Nat 1) == (x.*.y).+.x

pairs :: [a] → [(a,a)]
pairs xs = (,) <$> xs <*> xs
```

NonStdRobinson.hs:

```
module NonStdRobinson
(

    CuteNumber(Nat, Kappa, Lambda),
    (.+.), (.*.)
) where

data CuteNumber = Nat Int | Kappa | Lambda deriving (Eq, Ord, Show)

-- + + + + + + + + + + + + + + + + + + + + + + +

(.+.) :: CuteNumber → CuteNumber → CuteNumber
Nat n1 .+. Nat n2 = Nat (n1 + n2)

Nat _  .+. Kappa  = Kappa
Kappa  .+. Nat _  = Kappa
Kappa  .+. Kappa  = Kappa
Kappa  .+. Lambda = Lambda

Nat _  .+. Lambda = Lambda
Lambda .+. Nat _  = Lambda
Lambda .+. Lambda = Lambda
Lambda .+. Kappa  = Kappa

-- * * * * * * * * * * * * * * * * * * * * * * *

(.*.) :: CuteNumber → CuteNumber → CuteNumber
Nat n1 .*. Nat n2 = Nat (n1 * n2)

Nat 0  .*. _      = Nat 0
```

```
_      .*. Nat 0  = Nat 0

Nat _  .*. Kappa  = Kappa
Kappa  .*. _      = Kappa

Nat _  .*. Lambda = Lambda
Lambda .*. _      = Lambda
```