Software Engineering Hausaufgabe 5

Youran Wang (719511, RAS), Yannick Fuchs (723866, ITS)

November 2021

1 Analyse einer algebraischen Spezifikation

1.1

axioms
$false \neq true$
$not(false) = true$
$not(true) = false$

1.2

1.2.1

Ja, diese Aussage folgt aus den oberen Axiomen, da wir hier vier mal $succ()$ auf $zero$ anwenden und eines der Axiome lautet, dass wir nach dreimaliger Anwendung von $succ()$ auf $zero$, wieder $zero$ erhalten. Somit bleibt also nur noch $succ(zero)$ übrig und das steht hier ja auch so.

1.2.2

Auch diese Aussage geht aus den Axiomen hervor, denn $succ(succ(zero)) = n$ in diesem Fall, da n per Definition ein mnat sein muss und $succ(succ(zero)) \neq zero = mnat$.
Somit erhalten wir also $mult(n, zero)$, was ja per Definition $zero$ ist.

1.2.3

Nein, dies geht nicht aus den Axiomen hervor, da wir zu aller erst die $add(succ(succ(zero)), succ(zero))$ betrachten und das Ergebnis davon wäre:

$add(succ(succ(zero)), succ(zero))$
$= succ(add(succ(succ(zero)), zero))$
$= succ(succ(add(succ(zero)), zero))$
$= succ(succ(succ(add(zero, zero))))$
$= succ(succ(succ(zero)))$
$= zero$

Damit erhalten wir den Ausdruck $iszero(zero) = true$.

1.3

MNat beschreibt das mathematische Konstrukt des Körpers: $(\mathbb{N}_3, \oplus, \odot)$, da wir so gesehen die zahlen $\{0, 1, 2\}$ modellieren, da wir ja durch das Axiom $succ(succ(succ(zero))) = zero$ die Zahl 3 wieder auf die 0 abbilden. Dies gilt ebenso für die additive und die multiplikative Operation, welche wir in diesem Körper definiert haben.

1.4

Um ein Modell MNat zu spezifizieren, benötigen wir zu aller erst einmal ein Modell für Bool. Das sieht wie folgt aus:

$bool^{\mathcal{B}} = \{0, 1\}$
$false^{\mathcal{B}} = 0$
$true^{\mathcal{B}} = 1$
$not^{\mathcal{B}} = \quad 1 \to 0$
$\qquad\qquad\quad 0 \to 1$

Dies erfüllt schon mal die Spezifikation eines Bools. Kommen wir nun zu dem Modell des MNats:

$mnat^{\mathcal{M}} = \{0, 1, 2\}$
$zero^{\mathcal{M}} = 0$
$succ^{\mathcal{M}} = mnat + 1$
$add^{\mathcal{M}} = (mnat + mnat) mod 3$
$mult^{\mathcal{M}} = (mnat \cdot mnat) mod 3$
$iszero^{\mathcal{M}} = \quad 0 \to true^{\mathcal{B}}$
$\qquad\qquad\quad\ 1 \to false^{\mathcal{B}}$
$\qquad\qquad\quad\ 2 \to false^{\mathcal{B}}$

Dies ist ein Modell für die Spezifikation von MNat.

1.5

Ja, es gilt das Erzeugerprinzip, da wir theoretisch nur *zero* wirklich brauchen und jedes weitere Element von MNat durch die ein- oder zweifache Anwendung von *succ*() auf *zero* erzeugen können. Wir kommen sogar wieder zurück zu *zero*, durch die dreifache Anwendung von *succ*() auf MNat.

2

2.1

Die Funktion *intersect*() nimmt zwei Argumente, welche beide NatSets sein müssen und gibt uns ebenso ein NatSet zurück. Wir vergleichen die zwei NatSets, welche wir als Argumente eingegeben haben auf gleiche Elemente, welche in beiden NatSets enthalten sind und liefern diese "Überschneidung" der beiden in einem eigenen NatSet wieder zurück.

2.2

Die Trägermenge ist der $\mathbb{N}^{1,n}$.
$natSet^{\mathcal{A}} = \{[]\}$
$add^{\mathcal{A}} = [..., nat]$

$$
\begin{aligned}
isEmpty^{\mathcal{A}} = \quad & natSet = [] \to true^{\mathcal{B}} \\
& natSet \neq [] \to false^{\mathcal{B}} \\
contains^{\mathcal{A}} = \quad & [], n \to false^{\mathcal{B}} \\
& [n'^{\mathcal{N}}], n^{\mathcal{N}} \to false^{\mathcal{B}} \\
& [n^{\mathcal{N}}], n^{\mathcal{N}} \to true^{\mathcal{B}} \\
union^{\mathcal{A}} = \quad & [], [] \to [] \\
& [n^{\mathcal{N}}], [] \to [n^{\mathcal{N}}] \\
& [n^{\mathcal{N}}], [n'^{\mathcal{N}}] \to [n^{\mathcal{N}}, n'^{\mathcal{N}}] \\
& [n^{\mathcal{N}}], [n^{\mathcal{N}}] \to [n^{\mathcal{N}}] \\
intersect^{\mathcal{A}} = \quad & [], [] \to [] \\
& [n^{\mathcal{N}}], [] \to [] \\
& [n^{\mathcal{N}}], [n'^{\mathcal{N}}] \to [] \\
& [n^{\mathcal{N}}], [n^{\mathcal{N}}] \to [n^{\mathcal{N}}]
\end{aligned}
$$

2.3

Für diesen Beweis möchte ich nochmals darauf hinweisen, dass $n^{\mathcal{N}} \neq n'^{\mathcal{N}}$ gilt.
Unser Modell erfüllt das Erzeugungsprinzip, da wir jede einzelne Variation eines NatSets mit dieser Anzahl an Objekten erzeugen können. In unserem jetzigen Fall gibt es natürlich nur fünf Variationen, dies würde sich aber natürlich mit einer steigenden Zahl an Objekten auch erhöhen.

Wir können nun also entweder $[], [n^{\mathcal{N}}], [n'^{\mathcal{N}}], [n^{\mathcal{N}}, n'^{\mathcal{N}}]$ und $[n'^{\mathcal{N}}, n^{\mathcal{N}}]$ erzeugen.

Diese NatSets können wir alle mit $add()$ und $[]$ erzeugen, in dem wir $n^{\mathcal{N}}$ und $n'^{\mathcal{N}}$ in der richtigen Reihenfolge verwenden. Z.B.: $add(add([], n^{\mathcal{N}}), n'^{\mathcal{N}}) = [n^{\mathcal{N}}, n'^{\mathcal{N}}]$ und $add(add([], n'^{\mathcal{N}}), n^{\mathcal{N}}) = [n'^{\mathcal{N}}, n^{\mathcal{N}}]$.

Dies ist allerdings nicht unbedingt so wichtig, da Sets an sich ja nicht geordnet sind. Uns interessiert also nur welche Elemente enthalten sind, jedoch können wir auch dort jede erdenkliche Kombination auf die oben beschriebene Weise erzeugen. Bei den einzelnen Elementen müssen wir die $add()$-Funktion sogar nur einmal anwenden um die NatSets zu erzeugen und das leere NatSet benötigt nicht mal den Einsatz von $add()$. Somit können wir also jede Kombination aller unserer Elemente erzeugen und das bedeutet, dass unser Modell dem Erzeugungsprinzip folgt.

2.4

Seien hier $s = [n'^{\mathcal{N}}], s' = [n''^{\mathcal{N}}]$. Laut unserem Modell würden wir die Gleichung wie folgt lösen:

$$
\begin{aligned}
union(add(s, n), s') &= add(union(s, s'), n) \\
union(add([n'^{\mathcal{N}}], n^{\mathcal{N}}), [n''^{\mathcal{N}}]) &= add(union([n'^{\mathcal{N}}], [n''^{\mathcal{N}}]), n^{\mathcal{N}}) \\
union([n'^{\mathcal{N}}, n^{\mathcal{N}}], [n''^{\mathcal{N}}]) &= add([n'^{\mathcal{N}}, n''^{\mathcal{N}}], n^{\mathcal{N}}) \\
[n'^{\mathcal{N}}, n^{\mathcal{N}}, n''^{\mathcal{N}}] &= [n'^{\mathcal{N}}, n''^{\mathcal{N}}, n^{\mathcal{N}}]
\end{aligned}
$$

$$[n^{\mathcal{N}}, n'^{\mathcal{N}}, n''^{\mathcal{N}}] = [n^{\mathcal{N}}, n'^{\mathcal{N}}, n''^{\mathcal{N}}]$$

Wir haben die Elemente hier, der besseren Übersicht zu Gunsten, am Ende nochmal geordnet. Dies ist allerdings wie in dem vorherigen Aufgabenteil bereits beschrieben keine Eigenschaft des NatSets, hier zählt nur, dass die selben Elemente enthalten sind. Ihre Reihenfolge ist also egal. Wie wir sehen geht die Gleichung dann auf und dieses Axiom gilt in unserem Modell.

3

spec NatArray = Bool and Nat then
    sort
        $natArray = emptySet \mid set(natArray, nat, nat)$
    ops
        defined: (natArray, nat) bool
        get: (natArray, nat) nat
        remove: (natArray, nat)
        merge: (natArray, natArray) natArray
        sum: (natArray) nat
    vars
        s,s',s": natArray
        n,n',n": nat
    axioms
        defined(emptySet,n) = false
        defined(s,n) = false
        defined(set(s,n,n'),n) = true

        get(set(s,n,n'),n) = n'
        get(s,n) = null
        get(emptySet,n) = null
        get(remove(set(s,n,n'),n),n) = null

        remove(emptySet,n) = emptySet
        remove(s,n) = s'
        remove(set(s,n,n'),n) = s
        merge(emptySet,emptySet) = emptySet
        merge(s,emptySet) = s
        merge(emptySet,s) = s
        merge(s,s') = s"
        merge(set(s,n,n'),set(s,n,n")) = set(s',n,n')

        sum(emptySet) = 0
        sum(s) = n