

Formale objektorientierte Software-Entwicklung

Priv.-Doz. Dr. R. Hennicker, Dr. P. Kosiuczenko

Aufgabe 1 b) Blatt 4

Vorgehensweise bei der Erstellung von Vor- und Nachbedingungen für delete-Operationen von Systemklassen

Es seien die Klassen und Multiplizitäten wie in Abbildung gegeben. Statt einer Operation *addA* betrachten wir hier eine Operation *delA*. Im Gegensatz zu *add*-Operationen sind für *delete*-Operationen die Multiplizitäten *k* und *j* relevant, da

bei $k \geq 1$ kein B-Objekt existieren darf, das zu dem A-Objekt gehören muss,

bei $j \geq 1$ kein C-Objekt existieren darf, das das A-Objekt haben muss.

Der Fall $j, k \in \{0, *\}$ ist *trivial*. Es seien $j, k \in \{1, 1 \dots *, *\}$.¹ Dann müssen Vor- und Nachbedingung folgende Forderungen enthalten:

context Sys :: delA(kA:T)

Vorbedingung enthält:

- (1) Es existiert ein *a* in *as* mit $a.keyA = kA$
- (2) Falls $k = 1$ oder $k = 1 \dots *$: $a.myBs \rightarrow isEmpty()$
- (3) Falls $j = 1$: Es existiert kein *c* in *cs* mit $c.myAs = a$
- (4) Falls $j = 1 \dots *$: Es existiert kein $c \in cs$ mit
 $c.myAs \rightarrow includes(a)$ and $c.myAs \rightarrow size() = 1$

Nachbedingung enthält:

$A.allInstances() = A@pre.allInstances() \rightarrow excluding(a)$

was jedoch in Java schwer zu implementieren sein kann. Man kann das aber hier genauer beschreiben:

- (i) $as = as@pre \rightarrow excluding(a)$
- (ii) Falls $j = 1$ oder $j = 1 \dots *$: Für alle *c* in *cs* gilt $c.myAs \rightarrow excludes(a)$

1. Wir betrachten nicht den Fall in dem *j* oder *k* nach oben beschränkt sind.

Aufgabe 2 b) Blatt 4

Vor- und Nachbedingungen für die del-Operationen der Klasse Flugmanager

```
context Flugmanager ::
  delfluglinie(flname : String):Boolean
pre preDelFluglinieOk:
  let fl = findFluglinie(flname)
  in
  existsFluglinie(flname) and
    -- kein Pilot gehoert zur Fluglinie, d.h. Piloten wuerden fruher entlassen
    -- fl.personal -> isEmpty() and -- ist nicht noetig
    -- kein Flug gehoert zur Fluglinie (noetig wegen fuehrt durch Assoziation)
    fl.fluege -> isEmpty()
    -- dann hat auch kein Flug diese Fluglinie
post postDelFluglinieOk:
  let fl = fluglinien@pre -> any(name = flname)
  in
  result = true and
    -- Fluglinie entfernt
    fluglinien = fluglinien@pre -> excluding(fl) and
    -- Fluglinie als Partner entfernt
    fluglinien -> forAll(partner = partner@pre -> excluding(fl))
    -- oder
    -- Fluglinie.allInstances() = Fluglinie@pre.allInstances() -> excluding(fl)

pre: not(preDelFluglinieOk)
post: result = false
```

```
context Flugmanager :: delpilot(pname:String):Boolean
pre preDelPilotOk:
  let p = findPilot(pname)
  in
  existsPilot(pname) and
    -- kein Flug gehoert zum Piloten
    p.einsatz -> isEmpty() -- ein Pilot ist durch hoechstens 1 Fluglinie beschaefligt
post postDelPilotOk:
  let p = piloten@pre-> any(name = pname)
  in
  result = true and
    -- Pilot entfernt
    piloten = piloten@pre -> excluding(p) and
    -- Pilot vom Personal entfernt
    fluglinien -> forAll(personal -> excludes(p)) and
    -- Fluglinien unveraendert
    fluglinien = fluglinien@pre
    -- oder Pilot.allInstances() = Pilot@pre.allInstances() -> excluding(p)

pre: not(preDelPilotOk)
post: result =false
```

```

context Flugmanager :: delflug(dat: String, fnr: Integer):Boolean
pre preDelFlugOk:
    existsFlug(dat, fnr)
post postDelFlugOk:
    let f = fluege@pre -> any(datum = dat and fnr = flugnr)
    in
    result = true and
    -- Flug entfernt
    fluege = fluege@pre -> excluding(f) and
    -- Flug von Fluglinie entfernt
    fluglinien -> forAll(fluege -> excludes(f)) and
    -- Flug von Piloten entfernt
    piloten -> forAll(einsatz -> excludes(f)) and
    -- Fluglinien und Piloten unveraendert
    fluglinien = fluglinien@pre and piloten = piloten@pre
    -- oder: Flug.allInstances() = Flug@pre.allInstances() -> excluding(f)
pre: not(preDelFlugOk)
post: result =false

```