Tentamen Formele Methoden voor Software Engineering (213520)

15 april 2010, 8:45–12:15 uur.

BELANGRIJK: geef op je tentamen duidelijk aan:

- je studierichting
- of je beide huiswerkopgaven gemaakt hebt, en bij welke werkcollegeleider.

Bij dit tentamen mag je de de FSP Quick Reference Card en de JML Cheat Sheet gebruiken, alsmede de sheets van de colleges (geen boeken).

---

1. *(10 punten)* De volgende methode levert de index op van het eerste element in een niet-lege array dat kleiner is dan het voorgaande, of `0` als de array oploopt (dus elk element minstens zo groot is als het voorgaande):

```
int omhoog(int[] a) {
    int result = 0;
    int i = 1;
    while (result == 0 && i < a.length) {
        if (a[i] < a[i-1]) {
            result = i;
        }
        i++;
    }
    return result;
}
```

   (a) Specificeer voor deze methode een zo volledig mogelijk contract in JML.

   (b) Bewijs (met behulp van een lus-invariant) de correctheid van de methode. Geef de bewijsstappen duidelijk aan!

2. *(15 punten)* Beschouw de volgende Java-interface:

```
interface Cijfer {
    // Levert het eindcijfer op (een getal van 1 t/m 10)
    int getEind();

    // Verhoogt het cijfer met maximaal een gegeven (positief)
    // aantal punten.
    void verhoog(int punten);

    // Levert het vak op waar dit een cijfer voor is.
    Vak getVak();
}
```

   (a) Stel een JML-contract op voor `Cijfer`, waarin de beoogde werking zoveel mogelijk formeel gespecificeerd is.

   (b) Geef een (van JML-commentaar voorziene) implementatie `Simpel` van `Cijfer`, waarin het vak en het eindcijfer in instantievariabelen worden bijgehouden, en `verhoog` het aantal `punten` bij het eindcijfer optelt, met een maximum van `10`.

   (c) Geef van de methode `verhoog` van `Simpel` een correctheidsbewijs.

   (d) Geef een implementatie `Gemiddelde` van `Cijfer`, waarin een practicumcijfer en een tentamencijfer worden bijgehouden (beide instanties van `Cijfer`, voor hetzelfde vak), en het eindcijfer een gemiddelde van die twee is. Het vak is hetzelfde als dat van de twee samenstellende cijfers, en `verhoog` probeert `punten` bij beide deelcijfers op te tellen.
   (*Hint*: Vermijd zoveel mogelijk dat informatie twee keer wordt opgeslagen.)

(e) Geef voor de constructor en de methode `verhoog` van `Gemiddelde` een correctheidsbewijs.

(f) Wat gaat er fout in `Gemiddelde` als de postconditie van `verhoog` wordt versterkt, zodat het cijfer *precies* verhoogd dient te worden met `punten` (tot een maximum van `10`)?

3. *(25 punten)* De volgende eenvoudige FSP specificatie beschrijft een systeem, bestaand uit een proces en een resource. Het proces start, verkrijgt een resource, voert een taak uit, geeft de resource weer vrij en stopt.

```
RESOURCE = (obtain -> release -> RESOURCE).

PROCESS = (start -> obtain -> task -> FINISH),
FINISH = (release -> stop -> PROCESS).

||SYSTEM = (PROCESS || RESOURCE).
```

(a) *(7 punten)* Verander deze specificatie in een systeem met drie processen en drie resources. Verder moet een proces twee resources bemachtigen voor hij de taak kan uitvoeren. Hint: Als een proces resource i verkregen heeft, moet ie daarna nog resource $(i+1)\%3$ of $(i+2)\%3$ pakken.

(b) *(5 punten)* Het proces `SYSTEM` (met drie processen en drie resources) bevat een deadlock. Geef een trace die naar een deadlock leidt. We kunnen dit verhelpen door processen zo te veranderen dat ze de mogelijkheid hebben een resource vrij te geven als er nog geen tweede resource verkregen is. Pas de specificatie zo aan dat dit correct gemodelleerd wordt.

(c) *(5 punten)* We willen ook graag dat elk proces uiteindelijk twee resources kan bemachtigen en zijn taak kan uitvoeren. Druk deze eis uit m.b.v. een of meerdere `progress` eigenschappen uit. Maak aannemelijk dat deze eis niet wordt vervuld wanneer we twee dominante processen hebben die, zodra er een resource vrijkomt, deze altijd eerder te pakken hebben dan het derde proces. Hoe modelleren we dit scenario m.b.v. FSP?

(d) *(8 punten)* Om bovenstaand probleem te voorkomen, voeren we een teller in die ervoor zorgt dat er steeds ten hoogste twee processen opstarten. In dat geval hoeven de processen ook niet voortijdig resources vrij te geven. Geef een nieuwe versie van de specificatie waarin dit idee correct wordt uitgewerkt.