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!

(Beoordeling: Max. 3 punten voor het contract, max. 2 voor de invariant, en max. 5 voor de verschillende ingrediënten van het bewijs.)

(a). Het contract:

```
/*@
  @ requires a != null && a.length > 0;
  @ ensures \result >= 0 && \result < a.length;
  @ ensures \result == 0 ==>
  @     (\forall int k; 0 < k && k < a.length; a[k-1] <= a[k]);
  @ ensures \result > 0 ==> a[\result] < a[\result-1] &&;
  @     (\forall int k; 0 < k && k < \result; a[k-1] <= a[k])
  @*/
```

(b). De lusinvariant:

```
/*@ loop_invariant i > 0 && i <= a.length;
  @ loop_invariant result >= 0 && result < i;
  @ loop_invariant result == 0 ==>
  @      (\forall int k; 0 < k && k < i; a[k-1] <= a[k]);
  @ loop_invariant result > 0 ==> a[result] < a[result-1] &&
  @      (\forall int k; 0 < k && k < r; a[k-1] <= a[k]);
  @*/
```

We korten af:

$$
\begin{aligned}
Ord(x) &\equiv \forall 0 < k < x . a[k{-}1] \leq a[k] \\
R &\equiv a \neq \mathsf{null} \wedge |a| > 0 \\
E &\equiv r \geq 0 \wedge r < |a| \wedge (r = 0 \Rightarrow Ord(|a|)) \wedge (r > 0 \Rightarrow a[r] < a[r{-}1] \wedge Ord(r)) \\
C &\equiv r = 0 \wedge i < |a| \\
I &\equiv i > 0 \wedge i \leq |a| \wedge r \geq 0 \wedge r < i \wedge (r = 0 \Rightarrow Ord(i)) \wedge (r > 0 \Rightarrow a[r] < a[r{-}1] \wedge Ord(r))
\end{aligned}
$$

Het bewijs valt uiteen in drie delen:

- $\{R\}$ `r=0;i=1;` $\{I\}$ (preconditie garandeert lusinvariant)

$$
\begin{aligned}
& wp(\texttt{r=0; i=1;})\,\{I\} \\
& \quad \equiv wp(\texttt{r=0;}) \left\{ \begin{array}{l} 1 \leq |a| \wedge r \geq 0 \wedge r < 1 \\ \quad \wedge (r = 0 \Rightarrow Ord(1)) \\ \quad \wedge (r > 0 \Rightarrow a[r] < a[r{-}1] \wedge Ord(r)) \end{array} \right\} \\
& \quad \equiv 1 \leq |a| \wedge Ord(1) \\
& \quad \equiv 0 < |a|
\end{aligned}
$$

  (De laatste stap maakt gebruik van het feit dat $Ord(1)$ triviaal waar is.) De implicatie $R \Rightarrow 0 < |a|$ is duidelijk.

- $\{I \wedge C\}$ **`if`** `(a[i]<a[i-1]) {r=i;} i++;` $\{I\}$ (lusinvariant is correct)

  Hievoor moeten we bewijzen dat de preconditie de zwakste preconditie impliceert. Eerst de zwakste preconditie:

$$
\begin{aligned}
& wp(\textbf{if}\ \texttt{(a[i]<a[i-1]) \{r=i;\}\ i++;})\,\{I\} \\
& \quad \equiv wp(\textbf{if}\ \texttt{(a[i]<a[i-1]) \{r=i;\}}) \left\{ \begin{array}{l} i \geq 0 \wedge i < |a| \wedge r \geq 0 \wedge r < i{+}1 \\ \quad \wedge (r = 0 \Rightarrow Ord(i{+}1)) \\ \quad \wedge (r > 0 \Rightarrow a[r] < a[r{-}1] \wedge Ord(r)) \end{array} \right\} \\
& \quad \equiv \begin{array}{l} a[i] < a[i{-}1] \wedge i \geq 0 \wedge i < |a| \wedge i \geq 0 \wedge i < i{+}1 \\ \qquad \wedge (i = 0 \Rightarrow Ord(i{+}1)) \\ \qquad \wedge (i > 0 \Rightarrow a[i] < a[i{-}1] \wedge Ord(i)) \\ \quad \vee\ a[i] \geq a[i{-}1] \wedge i \geq 0 \wedge i < |a| \wedge r \geq 0 \wedge r < i{+}1 \\ \qquad \wedge (r = 0 \Rightarrow Ord(i{+}1)) \\ \qquad \wedge (r > 0 \Rightarrow a[r] < a[r{-}1] \wedge Ord(r)) \end{array} \\
& \quad \equiv \begin{array}{l} a[i] < a[i{-}1] \wedge i > 0 \wedge i < |a| \wedge Ord(i)) \\ \quad \vee\ a[i] \geq a[i{-}1] \wedge i \geq 0 \wedge i < |a| \wedge r \geq 0 \wedge r < i{+}1 \\ \qquad \wedge (r = 0 \Rightarrow Ord(i{+}1)) \\ \qquad \wedge (r > 0 \Rightarrow a[r] < a[r{-}1] \wedge Ord(r)) \end{array}
\end{aligned}
$$

  Dan de implicatie:

$$
\begin{aligned}
I \wedge C &\equiv i > 0 \wedge i < |a| \wedge r = 0 \wedge Ord(i) \\
&\equiv i > 0 \wedge i < |a| \wedge r = 0 \wedge Ord(i) \\
&\equiv i > 0 \wedge i < |a| \wedge r = 0 \wedge (Ord(i) \wedge a[i] < a[i{-}1] \vee Ord(i{+}1)) \\
&\Rightarrow \begin{array}{l} a[i] < a[i{-}1] \wedge i > 0 \wedge i < |a| \wedge Ord(i)) \\ \quad \vee\ a[i] \geq a[i{-}1] \wedge i \geq 0 \wedge i < |a| \wedge r \geq 0 \wedge r < i{+}1 \\ \qquad \wedge (r = 0 \Rightarrow Ord(i{+}1)) \\ \qquad \wedge (r > 0 \Rightarrow a[r] < a[r{-}1] \wedge Ord(r)) \end{array}
\end{aligned}
$$

- $I \wedge \neg C \Rightarrow E$ (lusinvariant garandeert postconditie)

$$I \wedge \neg C \equiv (I \wedge r \neq 0) \vee (I \wedge i \geq |a|)$$

We behandelen deze twee gevallen apart:

$$
\begin{aligned}
I \wedge r \neq 0 \equiv\ & i > 0 \wedge i \leq |a| \wedge r > 0 \wedge r < i \wedge a[r] < a[r{-}1] \wedge \mathit{Ord}(r) \\
\Rightarrow\ & r > 0 \wedge r < |a| \wedge a[r] < a[r{-}1] \wedge \mathit{Ord}(r) \\
\Rightarrow\ & E \\
I \wedge i \geq |a| \equiv\ & i > 0 \wedge i = |a| \wedge r \geq 0 \wedge r < i \\
& \wedge (r = 0 \Rightarrow \mathit{Ord}(i)) \\
& \wedge (r > 0 \Rightarrow a[r] < a[r-1] \wedge \mathit{Ord}(r)) \\
\Rightarrow\ & E
\end{aligned}
$$

Voor de beoordeling is het vooral van belang dat de verschillende stappen herkenbaar zijn. De details van het bewijs zijn complex en hoeven niet 100% correct uitgevoerd te zijn.

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.

```
interface Cijfer {
    //@ ensures \result == eind;
    //@ pure
    int getEind();

    //@ requires punten > 0;
    //@ ensures eind >= \old(eind);
    //@ ensures eind <= \old(eind)+punten;
    void verhoog(int punten);

    //@ ensures \result == vak;
    //@ pure
    Vak getVak();

    //@ instance model int eind;
    //@ instance model Vak vak;
    //@ invariant vak != null;
    //@ invariant eind >= 1 && eind <= 10;
}
```

   (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`.

```
class Simpel implements Cijfer {
    //@ requires vak != null;
    Simpel(Vak vak) {
        this._vak = vak;
        this._eind = 1;
    }

    public Vak getVak() {
        return _vak;
    }

    public int getEind() {
        return _eind;
    }

    public void verhoog(int punten) {
        _eind = _eind + punten;
        if (_eind > 10) {
            _eind = 10;
        }
    }

    private Vak _vak;
    private int _eind;
    //@ private represents vak = _vak;
    //@ private represents eind = _eind;
}
```

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

We korten `eind` tot `e` en `punten` tot `p` af. Om het bewijs te leveren moet de modelvariabele `e` in zijn implementatie `_e` worden omgezet, en moet er een *ghost*-variabele `e0` worden ingevoerd. Voor de leesbaarheid schrijven we toch `e` in plaats van `_e`. De bewijsverplichting kan dan worden geformuleerd doos het Hoare triple

$$\{I \wedge R\}\ \texttt{e0=\_e; \_e=\_e+p; if(\_e>10) \_e=10;}\ \{I \wedge E\}$$

waarbij

$$\begin{aligned} R &\equiv p > 0 \\ E &\equiv e \leq e_0 + p \\ I &\equiv e \geq 1 \wedge e \leq 10 \wedge v \neq \mathsf{null}. \end{aligned}$$

Aangezien er met `v` niets gebeurt, laten we de bewering $v \neg \mathsf{null}$ weg. We berekenen eerst de zwakste preconditie:

$$\begin{aligned} & wp(\texttt{e0=\_e; \_e=\_e+p; if(\_e>10) \_e=10;})\{e \geq 1 \wedge e \leq 10 \wedge e \geq e_0 \wedge e \leq e_0 + p\} \\ & \quad \equiv wp(\texttt{e0=\_e; \_e=\_e+p;}) \left\{ \begin{array}{l} e > 10 \wedge 10 \geq 1 \wedge 10 \leq 10 \wedge 10 \geq e_0 \wedge 10 \leq e_0 + p \\ \vee\, e \leq 10 \wedge e \geq 1 \wedge e \leq 10 \wedge e \geq e_0 \wedge e \leq e_0 + p \end{array} \right\} \\ & \quad \equiv wp(\texttt{e0=\_e;}) \left\{ \begin{array}{l} e + p > 10 \wedge 10 \geq e_0 \wedge 10 \leq e_0 + p \\ \vee\, e{+}p \leq 10 \wedge e{+}p \geq 1 \wedge e{+}p \geq e_0 \wedge e{+}p \leq e_0 + p \end{array} \right\} \\ & \quad \equiv e + p > 10 \wedge 10 \geq e \wedge 10 \leq e + p \\ & \qquad \vee\, e{+}p \leq 10 \wedge e{+}p \geq 1 \wedge e{+}p \geq e \wedge e{+}p \leq e + p \\ & \quad \equiv e + p > 10 \wedge 10 \geq e \\ & \qquad \vee\, e{+}p \leq 10 \wedge e{+}p \geq 1 \wedge e{+}p \geq e \end{aligned}$$

De implicatie:

$$\begin{aligned} I \wedge R &\equiv e \geq 1 \wedge e \leq 10 \wedge p > 0 \\ &\Rightarrow e{+}p \geq 1 \wedge e \leq 10 \wedge e{+}p \geq e \wedge (e{+}p > 10 \vee e{+}p \leq 10) \\ &\Rightarrow e + p > 10 \wedge 10 \geq e \\ &\quad \vee e{+}p \leq 10 \wedge e{+}p \geq 1 \wedge e{+}p \geq e \end{aligned}$$

(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.)

```
class Gemiddelde implements Cijfer {
    //@ requires prac != null;
    //@ requires tent != null;
    //@ requires prac.vak == tent.vak;
    Gemiddelde(Cijfer prac, Cijfer tent) {
        this.prac = prac;
        this.tent = tent;
    }

    public Vak getVak() {
        return prac.getVak();
    }

    public int getEind() {
        return (prac.getEind() + tent.getEind())/2;
    }

    public void verhoog(int punten) {
        prac.verhoog(punten);
        tent.verhoog(punten);
    }

    private Cijfer prac, tent;
    //@ private invariant prac != null;
    //@ private invariant tent != null;
    //@ private represents vak = prac.vak;
    //@ private represents eind = (prac.eind + tent.eind)/2;
}
```

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

De invariant is nu:

$$I \equiv (pr.e + te.e)/2 \geq 1 \wedge (pr.e + te.e)/2 \leq 10 \wedge pr.v \neq \text{null} \wedge pr \neq \text{null} \wedge te \neq \text{null}$$

Bovendien kunnen we in de bewijzen gebruik maken van de invariant van pr en te, en van de contracten van die objecten.
De bewijsverplichting voor de constructor (we gebruiken ppr en pte voor de parameters om het onderscheid met de instantievariabelen duidelijk te houden):

$$\{I \wedge ppr \neq \text{null} \wedge pte \neq \text{null} \wedge ppr.v = pte.v\}\ \texttt{pr=ppr; te=pte;}\ \{I\}$$

Het bewijs is rechttoe-rechtaan: de eisen $pr \neq \text{null}$ en $te \neq \text{null}$ zijn gegarandeerd door de precondities $ppr \neq \text{null}$ en $pte \neq \text{null}$; $pr.v \neq \text{null}$ volgt uit de invariant van ppr; en de volgende implicaties garanderen

de rest van de invariant

$$pr.e \geq 1 \wedge te.e \geq 1 \Rightarrow (pr.e + te.e)/2 \geq 1$$
$$pr.e \leq 10 \wedge te.e \leq 10 \Rightarrow (pr.e + te.e)/2 \leq 10$$

De bewijsverplichting voor `verhoog`:

$$\{I \wedge R\}\ \texttt{e0=(pr.e+te.e)/2; pr.verhoog(p); te.verhoog(p);}\ \{I \wedge E\}$$

waarin

$$R \equiv p > 0$$
$$E \equiv (pr.e + te.e)/2 \geq e_0 \wedge (pr.e + te.e)/2 \leq e_0 + p$$

Hier moeten we gebruik maken van het contract van de methode `verhoog` van pr en te. In de colleges is dit niet in detail behandeld; het exacte bewijs hoeft hier dan ook niet geleverd te worden. Voor de geïnteresseerden: het komt erop neer dat `verhoog` zich gedraagt als een toewijzing `e=e+p0;` voor een onbekende waarde `p0` met $0 \leq p_0 \leq p$. Zo krijgen we dan, voor één conjunct van de bewijsverplichting:

$$\begin{aligned}
&wp(\texttt{e0=(pr.e+te.e)/2; pr.verhoog(p); te.verhoog(p);})\,\{(pr.e + te.e)/2 \leq e_0 + p\} \\
&\quad \equiv wp(\texttt{e0=(pr.e+te.e)/2; pr.verhoog(p);})\,\{\exists 0 \leq p_0 \leq p.(pr.e + te.e + p_0)/2 \leq e_0 + p\} \\
&\quad \equiv wp(\texttt{e0=(pr.e+te.e)/2;})\,\{\exists 0 \leq p_0, p_1 \leq p.(pr.e + te.e + p_0 + p_1)/2 \leq e_0 + p\} \\
&\quad \equiv \exists p_0, p_2.0 \leq p_0 + p1 \leq 2p \wedge (pr.e + te.e + p_0 + p_1)/2 \leq (pr.e + te.e)/2 + p
\end{aligned}$$

Dit wordt geïmpliceerd door R. Voor de andere onderdelen van $I \wedge E$ verloopt het bewijs analoog.

(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`)?

Deze postconditie wordt niet gegarandeerd door de implementatie: bijvoorbeeld, als het practicumcijfer een 7 is en het tentamencijfer een 9, is het eindcijfer een 8, maar na de aanroep van `verhoog(2)` is het practicumcijfer een 9 en het tentamencijfer een 10, dus het eindcijfer een 9 (in plaats van een 10, zoals het volgens het contract zou moeten zijn).

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.

```
range I=0..2

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

||RESOURCES = ({res[I]}:RESOURCE).
```

```
PROCESS = ( start -> res[i:I].obtain ->
                (res[(i+1)%3].obtain -> task -> RETURN[i][(i+1)%3]
                |res[(i+2)%3].obtain -> task -> RETURN[i][(i+2)%3])
          ),
RETURN[i:I][j:I] = (res[i].release -> res[j].release -> stop ->
  PROCESS).

||PROCESSES = ({proc[I]}:PROCESS).

||SYSTEM = (PROCESSES || { proc[I]}::RESOURCES).
```

(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.

De deadlock trace:

```
Trace to DEADLOCK:
proc.0.res.0.obtain
proc.1.res.1.obtain
proc.2.res.2.obtain
```

De aangepaste specificatie:

```
...
PROCESS = (res[i:I].obtain ->
                (res[(i+1)%3].obtain -> task -> RETURN[i][(i+1)%3]
                |res[(i+2)%3].obtain -> task -> RETURN[i][(i+2)%3]
                |res[i].release -> PROCESS)
          ),
...
```

(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?

De `progress`-eigenschap:

```
progress STUDY[i:I] = {proc[i].task}
```

Dat twee processen dominant zijn kan in FSP met behulp van de prioriteitsoperator worden uitgedrukt:

```
||SYSTEM = (PROCESSES || { proc[I]}::RESOURCES)
          << {proc[0..1].res[I].obtain}.
```

(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.

```
...
COUNT = COUNT[0],
COUNT[i:I] = ( when (i>0) stop -> COUNT[i-1]
             | when (i<2) start -> COUNT[i+1]
             ).

||SYSTEM = (PROCESSES || {proc[I]}::RESOURCES || {proc[I]}::COUNT).
```