Tentamen Formele Methoden voor Software Engineering (213520)

14 april 2011, 8:45–12:15 uur.

- Vermeld je studierichting op het tentamen
- Geef aan of je de huiswerkopgaven gemaakt hebt, en in welke groep/bij welke werkcollegeleider.
- Naast de sheets van de colleges mag je de FSP Quick Reference Card en de JML Cheat Sheet gebruiken.
- Het cijfer voor dit tentamen is gelijk aan het het behaalde aantal punten gedeeld door 10.

---

1. *(50 punten)* Beschouw de volgende Java-interface:

```
public interface Verzekering {
    /** Telt een (positief) bedrag op bij het totaal aan declaraties,
      * als dit lukt. Mislukken kan bijvoorbeeld omdat het totaal of
      * het aantal declaraties een maximum heeft bereikt.
      * Als de declaratie mislukt, gebeurt er niets.
      * @return true als de declaratie gelukt is, anders false */
    boolean declareer(int bedrag);

    /** Levert het gemiddelde bedrag van alle declaraties op, of 0
      * als er nog niets gedeclareerd is.  */
    double getGemiddeld();

    /** Levert het totaalbedrag aan declaraties op. */
    int getTotaal();

    /** Zet de ingediende declaraties terug op 0. */
    void reset();
}
```

   (a) *(15 punten)* Stel een JML-contract op voor `Verzekering`, waarin de beoogde werking zoveel mogelijk formeel gespecificeerd is.

Let onder andere op de **`model`**-variabelen en de invarianten.

```
public interface Verzekering {
    /*@ requires bedrag > 0;
      @ ensures \result  ==> aantal == \old(aantal)+1
      @                   && totaal == \old(totaal)+bedrag;
      @ ensures !\result ==> aantal == \old(aantal)
      @                   && totaal == \old(totaal);
      @*/
    boolean declareer(int bedrag);

    //@ ensures \result == (aantal == 0 ? 0 : totaal/aantal);
    //@ pure
    double getGemiddeld();

    //@ ensures \result == totaal;
    //@ pure
    int getTotaal();

    //@ ensures aantal == 0 && totaal == 0;
    void reset();
    /*@ instance model double totaal;
      @ instance model int aantal;
      @ invariant totaal >= 0;
```

```
    @ invariant aantal >= 0;
    @*/
}
```

(b) *(10 punten)* Vul de volgende klasse aan met de rest van de methode-implementaties, inclusief alle JML-specificaties die nodig zijn om te kunnen bewijzen dat de klasse correct is.

Let op: gebruik alleen de gegeven instantievariabelen `aant` en `decl`! Het totaalbedrag is de som van de eerste `aant` elementen van `decl`. JML kent hiervoor het sleutelwoord `\sum`.

```
public class Implementatie implements Verzekering {
    public int getTotaal() {
        int result = 0;
        int i = 0;
        while (i < aant) {
            result = result + decl[i];
            i = i+ 1;
        }
        return result;
    }
    ...
    // aantal declaraties
    private int aant;
    // array met alle gedeclareerde bedragen
    private final int[] decl = new int[10];
}
```

Een fout in onderdeel 1a kan natuurlijk hier doorwerken. Het belangrijkste zijn de JML-specificaties, bestaande uit de **`represents`**-clausules en de invarianten. Bedenk daarnaast dat de beschikbare 10 punten tamelijk royaal zijn.

```
public class Implementatie implements Verzekering {
    public int getTotaal() {
        ...
    }

    public boolean declareer(int bedrag) {
        boolean result = aant < decl.length;
        if (result) {
            decl[aant] = bedrag;
            aant = aant + 1;
        }
        return result;
    }

    public int getGemiddeld() {
        return aant == 0 ? 0 : (getTotaal() / aant);
    }

    public void reset() {
        aant = 0;
    }

    private int aant;
    private final int[] decl = new int[10];
    //@ invariant decl != null;
    //@ invariant aant <= decl.length;
    //@ invariant (\forall int i; 0 <= i && i < aant; decl[i] > 0);

    //@ represents aantal = aant;
```

```
        //@ represents totaal = (\sum int i; 0 <= i && i < aant; decl[i]);
    }
```

(c) *(10 punten)* Bewijs de correctheid van de (zelf geïmplementeerde) methode `declareer`.

Vanwege de vrij grote hoeveelheid invarianten is het bewijs nogal arbeidsintensief. Let voor de beoordeling daarom vooral op de volgende punten:

- *Zijn de invarianten überheupt meegenomen als pre- en postconditie?*
- *Zijn de regels voor het* **`if`***-statement en sequentiële compositie correct toegepast?*

We korten af:

- `a` staat voor `aant`
- `d` staat voor `decl`
- `r` staat voor `result`

Zowel de pre- en postcondities van de methode als de klasseninvariant moeten worden meegenomen. Vertaald naar de body van de method levert dit op:

$$
\begin{aligned}
R &\equiv a_0 = a \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0 \wedge b > 0 \\
E &\equiv \big(r \Rightarrow (a = a_0 + 1 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0 + b)\big) \wedge \big(\neg r \Rightarrow (a = a_0 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0)\big) \\
I &\equiv \textstyle\sum_{k=0}^{a-1} d[k] \geq 0 \wedge a \geq 0 \wedge d \neq \mathsf{null} \wedge a \leq |d| \wedge \forall i : 0 \leq i < a \Rightarrow d[i] > 0 \ .
\end{aligned}
$$

The proof obligation is then $\{R \wedge I\}$ `r=a<d.length; if (r) {d[a]=b; a=a+1;}` $\{E \wedge I\}$
De eerste stap in het bewijs is:

$$
\begin{aligned}
&wp(\texttt{r=a<d.length; if (r) \{d[a]=b; a=a+1;\}})\,\{E \wedge I\} \\
&\quad\equiv (r \wedge wp(\texttt{d[a]=b; a=a+1;})\,\{E \wedge I\}) \vee (\neg r \wedge E \wedge I)
\end{aligned}
$$

We korten af:

$$
\begin{aligned}
P_1 &\equiv r \wedge wp(\texttt{d[a]=b; a=a+1;})\,\{E \wedge I\} \\
P_2 &\equiv \neg r \wedge E \wedge I \\
P &\equiv P_1 \vee P_2
\end{aligned}
$$

We rekenen P_1 en P_2 uit:

$$
\begin{aligned}
&wp(\texttt{d[a]=b; a=a+1;})\,\{E\} \\
&\quad\equiv wp(\texttt{d[a]=b;})\,\big\{\big(r \Rightarrow (a = a_0 \wedge \textstyle\sum_{k=0}^{a} d[k] = t_0 + b)\big) \wedge \big(\neg r \Rightarrow (a - 1 = a_0 \wedge \textstyle\sum_{k=0}^{a} d[k] = t_0)\big)\big\} \\
&\quad\equiv \big(r \Rightarrow (a = a_0 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0)\big) \wedge \big(\neg r \Rightarrow (a - 1 = a_0 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0 - b)\big)
\end{aligned}
$$

$$
\begin{aligned}
&wp(\texttt{d[a]=b; a=a+1;})\,\{I\} \\
&\quad\equiv wp(\texttt{d[a]=b;})\,\{\textstyle\sum_{k=0}^{a} d[k] \geq 0 \wedge a + 1 \geq 0 \wedge d \neq \mathsf{null} \wedge a < |d| \wedge (\forall i : 0 \leq i \leq a \Rightarrow d[i] > 0)\} \\
&\quad\equiv b + \textstyle\sum_{k=0}^{a-1} d[k] \geq 0 \wedge a + 1 \geq 0 \wedge d \neq \mathsf{null} \wedge a < |d| \wedge (\forall i : 0 \leq i < a \Rightarrow d[i] > 0) \wedge b > 0
\end{aligned}
$$

$$
\begin{aligned}
P_1 &\equiv r \wedge wp(\texttt{d[a]=b; a=a+1;})\,\{E\} \wedge wp(\texttt{d[a]=b; a=a+1;})\,\{I\} \\
&\quad\equiv r \wedge a = a_0 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0 \\
&\qquad \wedge b + t_0 \geq 0 \wedge a + 1 \geq 0 \wedge d \neq \mathsf{null} \wedge a < |d| \wedge (\forall i : 0 \leq i < a \Rightarrow d[i] > 0) \wedge b > 0
\end{aligned}
$$

$$
\begin{aligned}
P_2 &\equiv \neg r \wedge a = a_0 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0 \\
&\qquad \wedge t_0 \geq 0 \wedge a \geq 0 \wedge d \neq \mathsf{null} \wedge a \leq |d| \wedge (\forall i : 0 \leq i < a \Rightarrow d[i] > 0)
\end{aligned}
$$

Voor de tweede stap gebruiken we de volgende implicatie:

$$
wp(\texttt{r=a<d.length;})\,\{P\} \Leftarrow wp(\texttt{r=a<d.length;})\,\{P_1\} \vee wp(\texttt{r=a<d.length;})\,\{P_2\} \qquad (1)
$$

We berekenen nu de twee separate weakest preconditions:

$$
\begin{aligned}
&wp(\texttt{r=a<d.length;})\,\{P_1\}\\
&\quad\equiv a < |d| \wedge a = a_0 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0\\
&\qquad \wedge b + t_0 \geq 0 \wedge a + 1 \geq 0 \wedge d \neq \mathsf{null} \wedge (\forall i : 0 \leq i < a \Rightarrow d[i] > 0) \wedge b > 0\\
&wp(\texttt{r=a<d.length;})\,\{P_2\}\\
&\quad\equiv a = |d| \wedge a = a_0 \wedge \textstyle\sum_{k=0}^{a-1} d[k] = t_0\\
&\qquad \wedge t_0 \geq 0 \wedge a \geq 0 \wedge d \neq \mathsf{null} \wedge (\forall i : 0 \leq i < a \Rightarrow d[i] > 0)
\end{aligned}
$$

Het is nu direct in te zien dat

$$
\begin{aligned}
R \wedge I &\Rightarrow wp(\texttt{r=a<d.length;})\,\{P_1\}\\
R \wedge I &\Rightarrow wp(\texttt{r=a<d.length;})\,\{P_2\}
\end{aligned}
$$

Samen met de implicatie 1 bewijst dit de gewenste eigenschap.

(d) *(15 punten)* Bewijs de correctheid van de methode `getTotaal`

Voor de beoordeling is het vooral van belang dat de verschillende stappen herkenbaar zijn.
Aangezien de methode **`pure`** is hoeft de (klassen)invariant niet correct te worden bewezen. We hoeven dus alleen naar de postconditie te kijken. Volgens het stappenplan:

i. Transformeer het methode-contract naar pre- en postcondities P en Q voor de methode-body. De pre-conditie (**`requires`**) van de method is true, maar we hebben ook een stukje van de klasseninvariant nodig.

$$
\begin{aligned}
P &\equiv a \geq 0\\
Q &\equiv r = \textstyle\sum_{k=0}^{a-1} d[k] \ .
\end{aligned}
$$

ii. Stel een lusinvariant op.

$$
I \equiv i \leq a \wedge r = \textstyle\sum_{k=0}^{i-1} d[k] \ .
$$

iii. Bewijs de preprocessing correct. Te bewijzen: $\{P\}$ `r=0; i=0;` $\{I\}$

$$
\begin{aligned}
&wp(\texttt{r=0; i=0;})\left\{i \leq a \wedge r = \textstyle\sum_{k=0}^{i-1} d[k]\right\}\\
&\quad\equiv wp(\texttt{r=0;})\left\{0 \leq a \wedge r = \textstyle\sum_{k=0}^{-1} d[k]\right\}\\
&\quad\equiv 0 \leq a \wedge 0 = 0\\
&\quad\equiv P
\end{aligned}
$$

iv. Bewijs de lus correct. Te bewijzen: $\{I \wedge C\}$ `r=r+d[i]; i=i+1;` $\{I\}$

$$
\begin{aligned}
&wp(\texttt{r=r+d[i]; i=i+1;})\left\{i \leq a \wedge r = \textstyle\sum_{k=0}^{i-1} d[k]\right\}\\
&\quad\equiv wp(\texttt{r=r+d[i];})\left\{i < a \wedge r = \textstyle\sum_{k=0}^{i} d[k]\right\}\\
&\quad\equiv i < a \wedge r + d[i] = \textstyle\sum_{k=0}^{i} d[k]\\
&\quad\equiv C \wedge I
\end{aligned}
$$

v. Bewijs de postprocessing correct. Te bewijzen: $I \wedge \neg C \Rightarrow Q$.

$$
\begin{aligned}
i \leq q \wedge r = \textstyle\sum_{k=0}^{i-1} d[k] \wedge \neg(i < a) \\
&\equiv i = a \wedge r = \textstyle\sum_{k=0}^{i-1} d[k] \\
&\Rightarrow r = \textstyle\sum_{k=0}^{a-1} d[k] \\
&\equiv Q
\end{aligned}
$$

2. *(20 punten)* Beschouw de volgende FSP-specificatie van een (4-zijdige) dobbelsteen:

```
range Y = 1..4

DIE1 = (roll -> eyes[Y] -> DIE1)@{eyes[Y]}.

DIE2 = (roll[i:Y] -> eyes[i] -> DIE2)@{eyes[Y]}.

TRICK = ( roll[i:Y] -> eyes[i] -> TRICK | subst -> FALSE ),
    FALSE = (roll[1] -> eyes[1] -> FALSE).

||DIE3 = TRICK@{eyes[Y]}.
```

   (a) *(5 punten)* Teken de transitiesystemen van `DIE1`, `DIE2` en `DIE3`.

   (b) *(7 punten)* Zijn `DIE1` en `DIE2` zwak bisimilair? Waarom, of waarom niet?

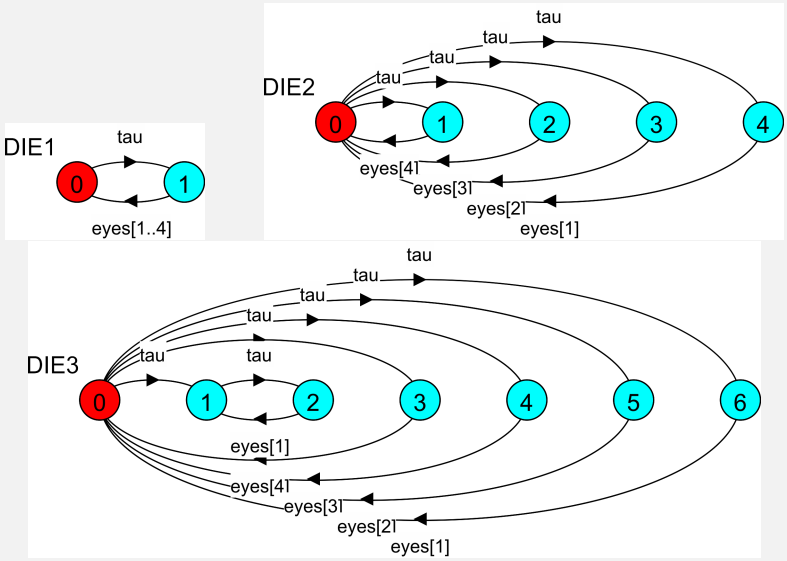
   (c) *(8 punten)* Beschouw de volgende *progress*-eigenschappen:

      i. De standaard-*progress*-eigenschap

      ii. `progress EYES = {eyes[Y]}`

   Welke van de processen `DIE1`, `DIE2` en `DIE3` voldoet aan welke van deze eigenschappen? Licht het antwoord toe.

2. (a) *(5 punten)* Als volgt:



   (b) *(7 punten)* Nee, `DIE1` en `DIE2` zijn niet bisimilair.

   - In toestandspaar $(0, 0)$: `DIE2` kan (met `tau`) naar toestand 1 gaan; `DIE1` kan alleen reageren door in 0 te blijven of ook naar 1 te gaan. Dus moeten $(0, 1)$ of $(1, 1)$ ook bisimilair zijn.

- In toestandspaar $(0,1)$: `DIE1` kan (met `tau`) vanuit 0 naar 1 gaan; `DIE2` kan alleen in 1 blijven. Dus moet $(1,1)$ bisimilair zijn.
- In toestandspaar $(1,1)$: `DIE1` kan `eyes[2]` doen, `DIE2` kan dat niet nadoen. Dus is $(1,1)$ niet bisimilair.

(c) *(8 punten)*

i. De standaard-*progress*-eigenschap is in `DIE3` niet vervuld: bijvoorbeeld is $\{1,2\}$ een terminal set, waarin `eyes[2]`, `eyes[3]` en `eyes[4]` niet meer mogelijk zijn.

ii. Deze speciale *progress*-eigenschap is wél vervuld: ook in de terminal set $\{1,2\}$ blijft één van de `eyes[Y]`-actie wel mogelijk, namelijk `eyes[1]`.

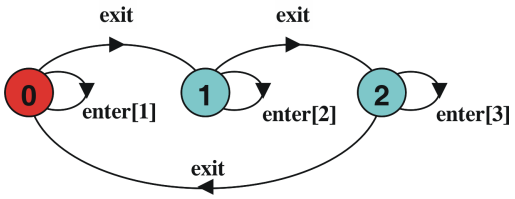
3. *(30 punten)* Beschouw de volgende FSP-specificatie:

```
const NP = 2 range P = 1..NP // proces-indices
const NT = 3 range T = 1..NT // ticket-nummers

property PROP1 = (p[pi:P].one -> p[pi].two -> PROP1).

property PROP2 = (p[pi:P].one -> PROP2[pi%NP + 1]),
  PROP2[pi:P] = (p[pi].one -> PROP2[pi%NP + 1]).
```

(a) *(5 punten)* Teken het transitiesysteem van `PROP1`

(b) *(10 punten)* Welke eigenschappen worden door `PROP1` en `PROP2` gespecificeerd (in je eigen woorden)?

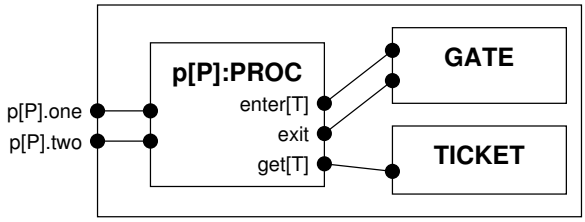
(c) *(5 punten)* Specificeer een FSP-proces `GATE` met het volgende gedrag:



(d) *(10 punten)* Specificeer een systeem, bestaande uit instanties van het volgende proces `PROC`:

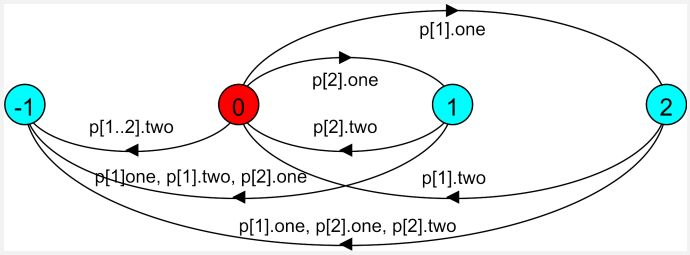
```
PROC    = (get[t:T] -> enter[t] -> one -> two -> exit -> PROC).
```

Het systeem moet aan `PROP1` voldoen, door te synchroniseren met `GATE` (zie hierboven) en een derde proces `TICKET`, zoals schematisch in de volgende figuur weergegeven:



Geef FSP-definities voor `TICKET` en het samengestelde systeem, en beargumenteer dat `PROP1` vervuld is.

3. (a) *(5 punten)* Als volgt:



(b) *(10 punten)*

- `PROP1` beschrijft een *mutual-exclusion*-eigenschap: na de actie `one` van `p1` kan alleen `two` van `p1` gebeuren, en geen van de acties `one` of `two` van `p2`; en vice versa.
- `PROP2` beschrijft dat de `one`-acties van `p1` en `p2` alleen maar om en om kunnen gebeuren, dus eerst `p[1].one`, dan `p[2].one`, dan weer `p[1].one`, enzovoort.

(c) *(5 punten)* Als volgt:

```
GATE            = GATE[1],
    GATE[t:T]   = ( enter[t] -> GATE[t]
                  | exit -> GATE[t%NT+1] ).
```

(d) *(10 punten)* Als volgt:

```
TICKET          = TICKET[1],
    TICKET[t:T] = ( get[t] -> TICKET[t%NT+1] ).

||SAFE = (p[P]:PROC || p[P]::TICKET || p[P]::GATE || PROP1)
         @{p[P].{one,two}}.
```