Tentamen Formele Methoden voor Software Engineering (213520)

2 juli 2009, 13:30-17:00 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. *(6 punten)*

   (a) Een vreemde drankenmachine heeft het volgende gedrag: na inworp van een even aantal munten (maar minstens twee) kun je altijd koffie kiezen, na inworp van een oneven aantal munten kun je altijd thee kiezen, en na inworp van precies een munt kun je altijd chocolademelk kiezen. Na afgifte van een drank komt de machine weer in de begintoestand.

   Geef een FSP proces `Machine` dat deze drankenmachine specificeert, en teken het bijbehorende transitiesysteem.

   (b) Teken voor elk van de onderstaande paren van processen de bijbehorende transitiesystemen. Is een paar observatie-equivalent, geef dan een zwakke bisimulatie, zo nee, geef dan aan waarom er geen zwakke bisimulatie mogelijk is:

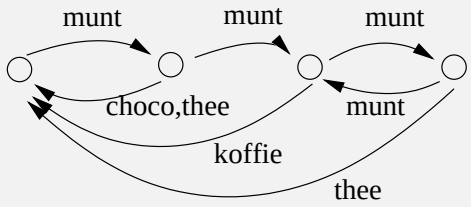
   i.
   ```
   P1 = (x -> STOP | p -> ( x -> STOP | y -> STOP ))\{p}.
   P2 = ( p -> ( x -> STOP | y -> STOP ))\{p}.
   ```

   ii.
   ```
   P1 = (p -> x -> STOP | p -> ( x -> STOP | y -> STOP ))\{p}.
   P2 = (p -> x -> STOP | p -> y -> STOP )\{p}.
   ```

   iii.
   ```
   P1 = (p -> x -> STOP | p -> ( x -> STOP | y -> STOP ))\{p}.
   P2 = (x -> STOP | p -> ( x -> STOP | y -> STOP ))\{p}.
   ```

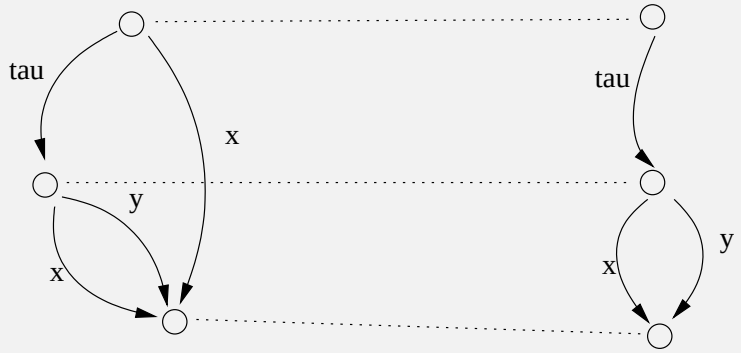
(a) *(3 punten)*

```
MACHINE = (munt -> ( {thee,choco} -> MACHINE |
                     munt -> KOFFIE
                   )
          ),
KOFFIE = (koffie -> MACHINE | munt -> THEE),
THEE = (thee -> MACHINE | munt -> KOFFIE).
```

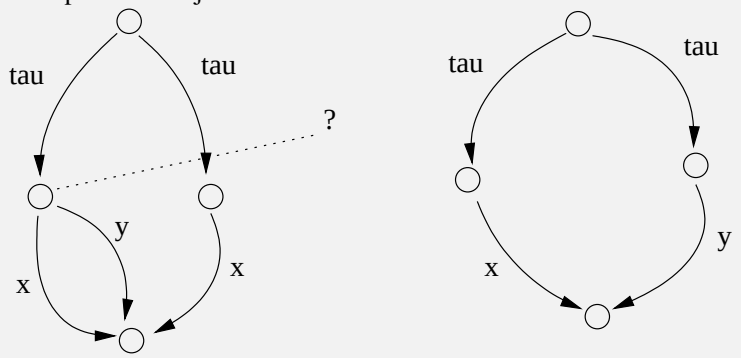


(b) *(3 punten)*

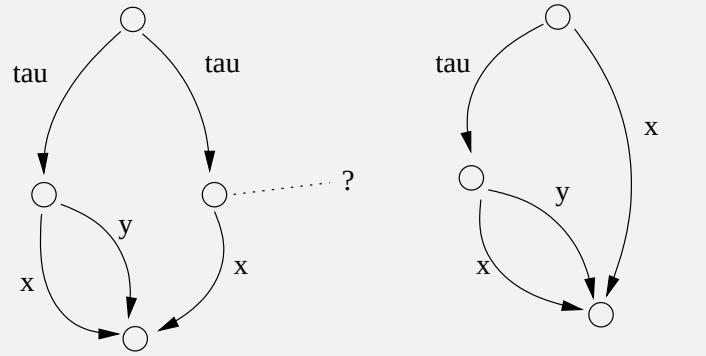
i. De twee processen zijn zwak bisimilair:



ii. De twee processen zijn niet zwak bisimilair:



iii. De twee processen zijn niet zwak bisimilair:



2. *(19 punten)*

De volgende eenvoudige FSP specificatie beschrijft een bibliotheek met een boek en een student.

```
BOOK = (taken -> returned -> BOOK).

STUDENT = (taken -> study -> RETURN),
RETURN = (returned -> STUDENT).

||LIBRARY = (STUDENT || BOOK).
```

(a) Verander deze specificatie in een bibliotheek met drie studenten en drie boeken. Verder moet een student twee boeken bemachtigen voor hij kan gaan studeren. Hint: Als een student boek i gepakt heeft, moet ie daarna nog boek $(i+1)\%3$ of $(i+2)\%3$ pakken.

(b) Het proces `LIBRARY` (met drie studenten en drie boeken) bevat een deadlock. Geef een trace die naar een deadlock leidt. We kunnen dit verhelpen door studenten de mogelijkheid te geven boeken te laten terugleggen als er geen tweede boek beschikbaar is. Pas de specificatie zo aan dat dit correct gemodelleerd wordt.

(c) We willen ook graag dat elke student uiteindelijk twee boeken kan bemachtigen en kan gaan studeren. 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 studenten hebben die, zodra er een boek vrijkomt, dat boek altijd eerder te pakken hebben dan de derde student. Hoe modelleren we dit scenario m.b.v. FSP?

(d) Om bovenstaand probleem te voorkomen, voeren we een teller in die ervoor zorgt dat er steeds ten hoogste twee studenten in de bibliotheek zijn. In dat geval hoeven de studenten ook niet voortijdig boeken terug te leggen. Geef een nieuwe versie van de specificatie waarin dit idee correct wordt uitgewerkt.

(a) *(5 punten)* De aangepaste specificatie:

```
range I=0..2

BOOK = (taken -> returned -> BOOK).

||BOOKS = ({book[I]}:BOOK).

STUDENT = (book[i:I].taken ->
                (book[(i+1)%3].taken -> study -> RETURN[i][(i+1)%3]
                |book[(i+2)%3].taken -> study -> RETURN[i][(i+2)%3])
           ),
RETURN[i:I][j:I] = (book[i].returned -> book[j].returned -> STUDENT).

||STUDENTS = ({student[I]}:STUDENT).

||LIBRARY = (STUDENTS || { student[I]}::BOOKS).
```

(b) *(4 punten)* De deadlock trace:

```
Trace to DEADLOCK:
        student.0.book.0.taken
        student.1.book.1.taken
        student.2.book.2.taken
```

De aangepaste specificatie:

```
...
STUDENT = (book[i:I].taken ->
                (book[(i+1)%3].taken -> study -> RETURN[i][(i+1)%3]
                |book[(i+2)%3].taken -> study -> RETURN[i][(i+2)%3]
                |book[i].returned -> STUDENT)
           ),
...
```

(c) *(4 punten)* De gevraagde `progress`-eigenschap:

```
...
||LIBRARY = (STUDENTS || { student[I]}::BOOKS) <<
{student[0..1].book[I].taken}.

progress STUDY[i:I] = {student[i].study}
```

(d) *(6 punten)*

```
...
STUDENT = (enter -> book[i:I].taken ->
                (book[(i+1)%3].taken -> study -> RETURN[i][(i+1)%3]
                |book[(i+2)%3].taken -> study -> RETURN[i][(i+2)%3])
           ),
RETURN[i:I][j:I] = (book[i].returned -> book[j].returned -> leave ->
```

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

||LIBRARY = (STUDENTS || { student[I]}::BOOKS || {student[I]}:: COUNT).
```

3. *(10 punten)* De volgende methode creëert een array waarin op elke positie de som van de elementen van de invoer-array tot en met die positie wordt gezet:

```
int[] somOp(int[] a) {
    int[] result = new int[a.length];
    int i = 0;
    int total = 0;
    while (i < a.length) {
        total += a[i];
        result[i] = total;
        i++;
    }
    return result;
}
```

   (a) Specificeer voor deze methode een zo volledig mogelijk contract in JML. (Maak hierbij gebruik van het JML-sleutelwoord `\sum`.)

   (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.)

1. Het contract:

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

2. De lusinvariant:

```
/*@ loop_invariant result.length == a.length && i <= a.length;
  @ loop_invariant total == (\sum int n; 0 <= n && n < i; a[n]);
  @ loop_invariant (\forall int k; 0 <= k && k < i;
  @    result[k] == (\sum int n; 0 <= n && n <= k; a[n]));
  @*/
while (i < a.length) {
    total += a[i];
    result[i] = total;
    i++;
}
```

We korten af:

$$
\begin{aligned}
R &= a \neq \mathsf{null} \\
E &= r \neq \mathsf{null} \wedge |r| = |a| \wedge \forall 0 \leq k < |a| : r[k] = \textstyle\sum_{n=0}^{k} a[n] \\
C &= i < |a| \\
I &= |r| = |a| \wedge i \leq |a| \wedge t = \textstyle\sum_{n=0}^{i-1} a[n] \wedge \forall 0 \leq k < i : r[k] = \textstyle\sum_{n=0}^{k} a[n]
\end{aligned}
$$

Het bewijs valt uiteen in drie delen:

- $\{R\}$ `r=new int[a.length]; i=0; t=0;` $\{I\}$ (preconditie garandeert lusinvariant)

$$
\begin{aligned}
& wp(\texttt{r=new int[a.length]; i=0; t=0;}) \\
& \quad \left\{ |r| = |a| \wedge i \leq |a| \wedge t = \textstyle\sum_{n=0}^{i-1} a[n] \wedge \forall 0 \leq k < i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \right\} \\
= \;& wp(\texttt{r=new int[a.length]; i=0;}) \\
& \quad \left\{ |r| = |a| \wedge i \leq |a| \wedge \textstyle\sum_{n=0}^{i-1} a[n] = 0 \wedge \forall 0 \leq k < i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \right\} \\
= \;& wp(\texttt{r=new int[a.length];}) \\
& \quad \{ |r| = |a| \wedge 0 \leq |a| \wedge 0 = 0 \wedge \mathsf{true} \} \\
= \;& |a| = |a| \\
= \;& a \neq \mathsf{null}
\end{aligned}
$$

  Note that the introduction of the condition $a \neq \mathsf{null}$ in the last step is necessary due to the fact that any condition with $|a|$ implicitly implies $a \neq \mathsf{null}$, but this implicit condition disappears when we replace $|a| = |a|$ with true.

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

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

$$
\begin{aligned}
& wp(\texttt{t+=a[i]; r[i]=t; i++;}) \\
& \quad \left\{ |r| = |a| \wedge i \leq |a| \wedge t = \textstyle\sum_{n=0}^{i-1} a[n] \wedge \forall 0 \leq k < i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \right\} \\
= \;& wp(\texttt{t+=a[i]; r[i]=t;}) \\
& \quad \left\{ |r| = |a| \wedge i < |a| \wedge t = \textstyle\sum_{n=0}^{i} a[n] \wedge \forall 0 \leq k \leq i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \right\} \\
= \;& wp(\texttt{t+=a[i];}) \\
& \quad \left\{ |r| = |a| \wedge i < |a| \wedge t = \textstyle\sum_{n=0}^{i} a[n] \wedge \forall 0 \leq k \leq i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \right\} \\
= \;& |r| = |a| \wedge i < |a| \wedge t + a[i] = \textstyle\sum_{n=0}^{i} a[n] \wedge \forall 0 \leq k \leq i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \\
= \;& |r| = |a| \wedge i < |a| \wedge t = \textstyle\sum_{n=0}^{i-1} a[n] \wedge \forall 0 \leq k \leq i : r[k] = \textstyle\sum_{n=0}^{k} a[n]
\end{aligned}
$$

  Aangezien duidelijk is dat de laatste conditie equivalent is met $C \wedge I$ zijn we klaar.

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

$$
\begin{aligned}
& |r| = |a| \wedge i \leq |a| \wedge t = \textstyle\sum_{n=0}^{i-1} a[n] \wedge \forall 0 \leq k < i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \wedge i \geq |a| \\
& \quad \Rightarrow \quad |r| = |a| \wedge i = |a| \wedge \forall 0 \leq k < i : r[k] = \textstyle\sum_{n=0}^{k} a[n] \\
& \quad \Rightarrow \quad r \neq \mathsf{null} \wedge |r| = |a| \wedge \forall 0 \leq k < |a| : r[k] = \textstyle\sum_{n=0}^{k} a[n]
\end{aligned}
$$

4. *(15 punten)* Beschouw de volgende Java-interface:

```
interface Ding {
    // Levert de prijs van dit ding op
    double getPrijs();

    // Vermindert de prijs met een bepaald percentage
    // strict tussen de 0 en de 100
    void korting(double perc);

    // Levert het wagentje op waar dit ding in is gelegd.
    // Als het resultaat null is, ligt het ding niet in een wagentje.
    Wagentje getWagentje();

    // Legt dit ding in een gegeven wagentje.
    // Alleen aan te roepen als het ding niet al in een wagentje ligt.
    void laadIn(Wagentje wagentje);

    // Haalt dit ding uit het wagentje.
    // Alleen aan te roepen als het ding inderdaad in een wagentje ligt.
    void laadUit();
}
```

(a) Stel een contract in JML op voor `Ding`, waarin het JavaDoc-commentaar zoveel mogelijk formeel gespecificeerd is.

(b) `Ding` wordt geïmplementeerd in een klasse `Boek`. Geef van `Boek` de volgende onderdelen, met (waar van toepassing) bijbehorende JML-specificaties zodat de klasse correct te bewijzen is:

- De constructor (waarin de `prijs` wordt gezet)
- De methode `korting`
- De instantievariabelen

(c) Geef van de methode `korting` van `Boek` het correctheidsbewijs.

Beschouw nu de volgende (onvolledige) klasse:

```
class Wagentje {
    Wagentje() {
        inhoud = new Ding[10];
        aantal = 0;
    }

    //@ requires ding != null;
    //@ ensures ding.getWagentje() == this;
    public boolean voegToe(Ding ding) {
        if (aantal < inhoud.length) {
            inhoud[aantal] = ding;
            aantal++;
            ding.laadIn(this);
            return true;
        } else {
            return false;
        }
    }

    //@ ensures aantal > 0 ==> \result > 0;
    public double totaal() {
        double result = 0;
        for (int i = 0; i < aantal; i++) {
            result += inhoud[i].getPrijs();
        }
```

```
        return result;
    }

    // inhoude van het wagentje
    private Ding[] inhoud;
    // aantal elementen in inhoud die werkelijk een Ding bevatten
    // (voor de rest is inhoud null)
    private int aantal;
}
```

(d) Geef zo compleet mogelijke invarianten voor de instantievariabelen van `Wagentje`. Betrek daarin ook de specificatie van `Ding`.

(e) Welke JML-gerelateerde fout(en) bevat de methode `voegToe`? Geef aan hoe deze verbeterd moeten worden.

(f) Geef het bewijsidee voor de correctheid van `totaal` (in `Wagentje`), inclusief de uit te voeren bewijsstappen. *5 bonuspunten:* Geef het formele correctheidsbewijs.

(a). *(3 punten)* Het contract van `Ding`:

```
interface Ding {
    //@ ensures \result == prijs;
    //@ pure
    double getPrijs();

    //@ requires perc > 0 && perc < 100;
    //@ ensures prijs == \old(prijs) * (100-perc)/100;
    void korting(double perc);

    //@ ensures \result == wagentje;
    //@ pure
    Wagentje getWagentje();

    //@ requires wagentje != null && this.wagentje == null;
    //@ ensures this.wagentje == wagentje;
    void laadIn(Wagentje wagentje);

    //@ requires this.wagentje != null;
    //@ ensures this.wagentje == null;
    void laadUit();

    //@ instance model double prijs;
    //@ instance model Wagentje wagentje;
    //@ invariant prijs > 0;
}
```

(b). *(2 punten)* De relevante onderdelen van `Boek`:

```
class Boek implements Ding {
    //@ requires prijs > 0;
    Boek(double prijs) {
        _prijs = prijs;
    }

    public void korting(double perc) {
        _prijs *= (100-perc)/100;
    }
    ...
    //@ public represents prijs = _prijs;
```

```
    //@ public represents wagentje = _wagentje;
}
```

(c). *(3 punten)* Zoals gewoonlijk moeten we bewijzen:

$$\{R \wedge I\}\ \texttt{body}\ \{E \wedge I\}$$

waarbij

$$\begin{aligned} R &= 0 < pe < 100 \\ E &= (pr = pr_0 * (100 - pe)/100) \\ I &= pr > 0 \end{aligned}$$

We bewijzen (opnieuw zoals gewoonlijk) $R \wedge I \Rightarrow wp(\texttt{body})\,\{E \wedge I\}$

$$\begin{aligned} & wp(\texttt{pr0=pr; pr=pr*(100-pe)/100;})\,\{pr = pr_0 * (100 - pe)/100 \wedge pr > 0\} \\ &= wp(\texttt{pr0=pr;})\,\{pr * (100 - pe)/100 = pr_0 * (100 - pe)/100 \wedge pr * (100 - pe)/100 > 0\} \\ &= pr * (100 - pe)/100 = pr * (100 - pe)/100 \wedge pr * (100 - pe) > 0 \\ &= (pr > 0 \wedge (100 - pe) > 0) \vee (pr < 0 \wedge (100 - pe) < 0) \\ &= (pr > 0 \wedge pe < 100) \vee (pr < 0 \wedge pe > 100) \end{aligned}$$

Since clearly $R \wedge I \Rightarrow (pr > 0 \wedge pe < 100) \vee (pr < 0 \wedge pe > 100)$, we are done.

(d). *(2 punten)* De invarianten voor `Wagentje`:

```
/*@ private invariant inhoud != null;
  @ private invariant 0 <= aantal && aantal <= inhoud.length;
  @ private invariant (\forall int i; 0<=i && i<aantal;
  @           inhoud[i] != null && inhoud[i].getWagentje() == this);
  @ private invariant (\forall int i; aantal<=i && i<inhoud.length;
  @           inhoud[i] == null);
  @*/
```

(e). *(3 punten)* Twee fouten in het contract:

- De preconditie van `Ding.laadIn` is niet vervuld, aangezien `voegToe` niet eist dat `ding.wagentje == null`. De preconditie moet dus worden aangescherpt.
- De postconditie `ding.getWagentje() == this` van `voegToe` is niet vervuld als de **else**-tak wordt gekozen. De postconditie moet dus worden verzwakt.

De gerepareerde specificatie ziet er als volgt uit:

```
//@ requires ding != null && ding.wagentje == null;
//@ ensures \result ==> ding.getWagentje() == this;
```

(f). *(2 punten)* Uiteraard meten we wee bewijzen dat $\{R \wedge I\}\ \texttt{body}\ \{E \wedge I\}$ geldt, waarbij

- $R = \mathsf{true}$
- $E = a > 0 \Rightarrow r > 0$
- $I = 0 \leq a \wedge a \leq |in|$

Aangezien `totaal` een lus bevat kan dit niet met de weakest precondition-semantiek: in plaats daarvan moeten de volgende bewijsstappen worden uitgevoerd, waarbij $C = (i < a)$ (de conditie van de lus) en $J = (a > 0 \Rightarrow r > 0) \wedge 0 \leq i \leq a$ (de lusinvariant):

- $\{R \wedge I\}$ `r=0; i=0;` $\{J\}$
- $\{J \wedge C\}$ `r = r+in[i].getPrijs(); i=i+1;` $\{J\}$
- $J \wedge \neg C \Rightarrow E \wedge I$