Tentamen Formele Methoden voor Software Engineering (192135201)

18 april 2013, 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. *(20 punten)* Beschouw de volgende FSP-specificatie:

```
// De gewenste gang van zaken
STUDIE1 =   ( geslaagd -> diploma -> STUDIE1 ).

// Opgedeeld in rollen
STUDENT2 = ( geslaagd -> controle -> STUDENT2 ).
BOZ =  ( controle -> diploma -> BOZ ).

||STUDIE2 = (STUDENT2 || BOZ) \ {controle}.

// Mogelijk (hoewel ongewenst) alternatief
DOCENT  = ( geslaagd -> DOCENT | gezakt -> DOCENT ).
STUDENT3 = ( geslaagd -> controle -> STUDENT3 | gezakt -> STUDENT3 ).

||STUDIE3 = (DOCENT || STUDENT3 || BOZ) \ {controle, gezakt}.
```

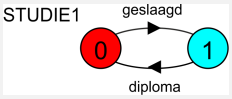
   (a) *(8 punten)* Teken de transitiesystemen van `STUDIE1`, `STUDIE2` en `STUDIE3`. Teken in de laatste twee gevallen ook de deelprocessen `STUDENT2`, `DOCENT`, `STUDENT3`, en `DOCENT`.

   (b) *(7 punten)* Welke van de drie bovenstaande processen zijn bisimilair? Geef de relatie tussen toestanden aan bij bisimilaire processen, of leg anders uit waarom dit niet kan.

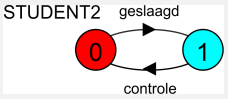
   (c) *(5 punten)* Formuleer een *safety property* die uitdrukt dat er uitsluitend afwisselend `geslaagd`- en `diploma`-acties kunnen plaatsvinden, en teken het bijbehorende transitiesysteem. Welke van de bovenstaande processen voldoen niet aan deze eigenschap, en onder welke omstandigheden?

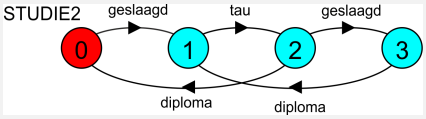
1. (a) (*8 punten*) De transitiesystemen:

   - `STUDIE1`:

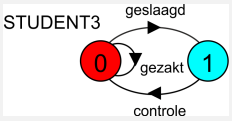


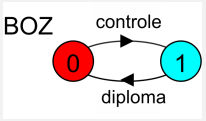
   - `STUDENT2`, `DOCENT` en `STUDIE2`:

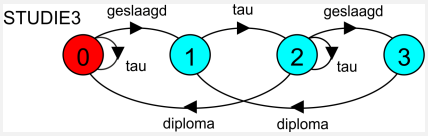




   - `STUDENT3`, `BOZ` en `STUDIE3`:







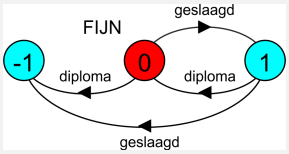
   (b) (*7 punten*) `STUDIE1` en `STUDIE2` zijn niet trace-equivalent, en dus zeker niet bisimilair: `STUDIE2` heeft een trace `pas pas` die in `STUDIE1` niet mogelijk is. Hetzelfde geldt voor `STUDIE1` t.o.v. `STUDIE3`.

   `STUDIE2` en `STUDIE3` wijn wél bisimilair: elke toestand van `STUDIE2` gedraagt zich precies zo als de overeenkomstig genummerde toestand van `STUDIE3`. De **`tau`**-lussen bij toestanden `0` en `2` van `STUDIE3` maken voor de bisimilariteit geen verschil.

   (c) (*5 punten*) De eigenschap lijkt erg op `STUDIE1`, en luidt

```
property FIJN = (geslaagd -> diploma -> FIJN).
```

   Het transitiesysteem is



   `STUDIE2` en `STUDIE3` voldoen niet aan deze eigenschap, vanwege de bestaande trace `pas pas` die in `FIJN` naar de fouttoestand `-1` leidt.

2. *(30 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) *(10 punten.)* 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 ($i = 0, 1, 2$), moet-ie daarna nog boek $(i+1)\%3$ of $(i+2)\%3$ pakken.)

   (b) *(10 punten.)* Het uitgebreide 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) *(10 punten.)* 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.
   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 in FSP?

2. (a) *(10 punten.)* De uitgebreide specificatie:

```
range B=0..2
range S=0..2

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

||BOOKS = (book[B]:BOOK).

STUDENT = (book[i:B].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:B][j:B] =
           ( book[i].returned -> book[j].returned -> STUDENT ).

||STUDENTS = (student[S]:STUDENT).

||LIBRARY = (STUDENTS || student[S]::BOOKS).
```

   (b) *(10 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:

```
STUDENT2 = (book[i:B].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 -> STUDENT2 )
           ),
    RETURN[i:B][j:B] =
           ( book[i].returned -> book[j].returned -> STUDENT2 ).
```

2. (c) *(10 punten.)* De gevraagde `progress`-eigenschap:

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

Scenario met dominante studenten:

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

3. *(30 punten)* Beschouw de volgende Java-interface (waarbij `Team` een voorgedefinieerde klasse is).

```
interface Wedstrijd {
    /** Levert het thuisteam van de wedstrijd op. */
    Team getThuis();

    /** Levert het uitteam van de wedstrijd op. */
    Team getUit();

    /** Registreert dat het thuis- of uitteam een punt heeft gemaakt. */
    void punt(Team team);

    /** Levert de score van het thuis- of uitteam op. */
    int getScore(Team team);

    /** Levert de winnaar van de wedstrijd op, als die er is. */
    Team getWinnaar();
}
```

(a) *(10 punten)* Stel een JML-contract op voor `Wedstrijd`, waarbij de beoogde werking zoveel mogelijk formeel gespecificeerd is. Maak waar dit nuttig is gebruik van modelvariabelen.

(b) *(8 punten)* Schrijf een implementatie `NormaleWedstrijd` van bovenstaand interface, met voldoende JML-commentaar om de correctheid te kunnen bewijzen. Representeer daarbij de modelvariabelen door middel van het totaal aantal punten en het verschil tussen thuis- en uitteam, in plaats van door de aantallen punten per team.

(c) *(7 punten)* Bewijs de correctheid van uw methode `punt`.

(d) *(5 punten)* Geef een JML-invariant voor `NormaleWedstrijd` die uitdrukt dat het thuisteam nooit op verlies kan staan. Deze invariant geldt uiteraard niet. Leg precies uit, in termen van het JML-contract van de klasse, waar de fout zit.

3. (a) *(10 punten)* Het interface met contract:

```
interface Wedstrijd {
    /*@ pure
      @ ensures \result == thuis;
      @*/
    Team getThuis();

    /*@ pure
      @ ensures \result == uit;
      @*/
    Team getUit();

    /*@ requires team == thuis || team == uit;
      @ ensures getScore(team) == \old(getScore(team))+1;
```

```
    @*/
  void punt(Team team);

  /*@ pure
    @ requires team == thuis || team == uit;
    @ ensures \result == (team == thuis ? thuisScore : uitScore);
    @*/
  int getScore(Team team);

  /*@ pure
    @ ensures \result == thuisScore > uitScore ? thuis :
    @                    thuisScore < uitScore ? uit :
    @                    null;
    @*/
  Team getWinnaar();

  /*@ instance model Team thuis, uit;
    @ instance model int thuisScore, uitScore;
    @ instance invariant thuis != null && uit != null && thuis != uit;
    @ instance invariant thuisScore >= 0 && uitScore >= 0;
    @*/
}
```

(b) *(8 punten)* De implementatie:

```
class NormaleWedstrijd {
  /*@ requires t != null && u != null && t != u;
    @ ensures thuis == t && uit == u;
    @ ensures thuisScore == 0 && uitScore == 0;
    @*/
  NormaleWedstrijd(Team t, Team u) {
      this.t = t;
      this.u = u;
  }

  Team getThuis() {
      return t;
  }

  Team getUit(){
      return u;
  }

  void punt(Team team) {
      totaal = totaal + 1;
      verschil = verschil + (team == t ? 1 : -1);
  }

  int getScore(Team team) {
      return (totaal + (team == t ? verschil : -verschil)) / 2;
  }

  Team getWinnaar() {
      return verschil > 0 ? t : verschil < 0 ? u : null;
  }

  private final Team t, u;
  private int totaal, verschil;
  //@ private represents thuis = t && uit = u;
```

```
    //@ private represents thuisScore = (totaal + verschil)/2;
    //@ private represents uitScore = (totaal - verschil)/2;
    //@ private invariant t != null && u != null && t != u;
    //@ private invariant totaal >= 0 && verschil <= totaal;
}
```

(c) *(7 punten)* Te bewijzen is het Hoare-triple

$$\{R \wedge I\}\ \texttt{tt0=tt; v0=v; tt=tt+1; v=v+(tm==t ? 1 : -1);}\ \{E \wedge I\}$$

waarbij `team` is afgekort tot `tm`, `totaal` is afgekort tot `tt`, en R, E en I zijn gedefininieerd door

$$\begin{aligned}
R &\equiv tm = t \vee tm = u \\
E &\equiv (tm = t\,?\,(tt+v)/2 : (tt-v)/2) = (tm = t\,?\,(tt_0+v_0)/2 : (tt_0-v_0)/2) + 1 \\
I &\equiv t \neq \mathsf{null} \wedge u \neq \mathsf{null} \wedge t \neq u \wedge tt \geq 0 \wedge v \leq tt
\end{aligned}$$

We laten het eerste deel van I weg, aangezien t en u niet van waarde veranderen en de eigenschap daarom triviaal geldt. De weakest precondition-berekening wordt dan:

$$\begin{aligned}
&wp(\texttt{tt0=tt; v0=v; tt=tt+1; v=v+(tm==t ? 1 : -1);})\,\{E \wedge I\} \\
&\quad\equiv wp(\texttt{tt0=tt; v0=v; tt=tt+1;}) \left\{ \begin{array}{l} tm = t\,?\,(tt+(v+1))/2 : (tt-(v-1))/2 \\ =\ (tm = t\,?\,(tt_0+v_0)/2 : (tt_0-v_0)/2) + 1 \\ \wedge\ tt \geq 0 \wedge v + (tm = t\,?\,1 : -1) \leq tt \end{array} \right\} \\
&\quad\equiv wp(\texttt{tt0=tt; v0=v;}) \left\{ \begin{array}{l} (tm = t\,?\,(tt+2+v)/2 : (tt+2-v)/2) \\ =\ (tm = t\,?\,(tt_0+v_0)/2 : (tt_0-v_0)/2) + 1 \\ \wedge\ tt+1 \geq 0 \wedge v + (tm = t\,?\,1 : -1) \leq tt+1 \end{array} \right\} \\
&\quad\equiv (tm = t\,?\,(tt+v)/2 : (tt-v)/2) + 1 \\
&\qquad = (tm = t\,?\,(tt+v)/2 : (tt-v)/2) + 1 \\
&\qquad \wedge\ tt+1 \geq 0 \wedge v + (tm = t\,?\,1 : -1) \leq tt+1 \\
&\quad\equiv tt+1 \geq 0 \wedge v + (tm = t\,?\,1 : -1) \leq tt+1 \\
&\quad\Leftarrow tt+1 \geq 0 \wedge v+1 \leq tt+1 \\
&\quad\Leftarrow I\ .
\end{aligned}$$

Het blijkt dat we R niet eens nodig hebben!

(d) *(5 punten)* De invariant luidt:

```
private invariant verschil >= 0;
```

Deze invariant kan geschonden zijn na afloop van de methode `punt`, zelfs als bij aanroep zowel de invariant als de precondities vervuld zijn.

4. *(20 punten)* De JML-expressie `(\sum int; P(i); E(i))` levert, voor een gegeven conditie `P` en expressie `E`, de som op van alle $E(i)$ waarvoor $P(i)$ geldt. Bijvoorbeeld is `(\sum int i; 1<=i && i<=n; i)` de som van alle gehele getallen van 1 tot en met n — oftewel, in wiskundige notatie is dit hetzelfde als $\sum_{i=1}^{n} i$.

   Beschouw nu de volgende methode, die het aantal voldoendes in een gegeven array `cijfers` oplevert:

```
int tel(int[] cijfers) {
    int result = 0;
    int k = 0;
    while (k < cijfers.length) {
        if (cijfers[k] > 5) {
            result = result + 1;
        }
        k = k+1;
    }
    return result;
}
```

   (a) (*5 punten*) Voorzie deze methode van een contract dat de beoogde werking zo precies mogelijk vastlegt. Hierbij is `cijfers` een array met cijfers, d.w.z., getallen van 1 tot en met 10.

   (b) (*10 punten*) Bewijs met behulp van een lusinvariant dat de methode aan haar contract voldoet.

   (c) (*5 punten*) Beschouw nu een variant `telAlle` op de methode `tel`, met signatuur `int[] telAlle(int[] cijfers)`, die voor eenzelfde parameter `cijfers` een nieuwe array oplevert (zeg `r`) zodat `r[k]` voor elke zinnige `k` het aantal cijfers hoger dan `k` bevat. (Bijvoorbeelt bevat `r[5]` het aantal voldoendes.) Geef een zo precies mogelijk contract voor deze variant. (N.B.: je hoeft de methode `telAlle` zelf niet uit te programmeren!)

4. (a) (*5 punten*) De geannoteerde methode:

```
/*@ requires cijfers != null;
  @ requires (\forall int i;
  @           0<=i && i < cijfers.length;
  @           cijfers[i] >= 1 && cijfers[i] <= 10);
  @ ensures \result ==
  @             ( \sum int i;
  @               0 <= i && i < cijfers.length && cijfers[i] > 5;
  @               1 );
  @*/
int tel(int[] cijfers) {
    int result = 0;
    int k = 0;
    /*@ loop_invariant k <= cijfers.length;
      @ loop_invariant
      @     result ==
      @        ( \sum int i;
      @          0 <= i && i < k && cijfers[i] > 5;
      @          1 );
      @*/
    while (k < cijfers.length) {
        if (cijfers[k] > 5) {
            result = result + 1;
        }
        k = k+1;
    }
    return result;
}
```

4. (b) (*10 punten*) We korten `cijfers` af tot c en `result` tot r. We definiëren een hulppredicaat $Tel(k)$ dat uitdrukt dat de resultaten in c tot en met positie $k-1$ geteld zijn:

$$Tel(k) \equiv r = |\{i \mid 0 \leq i \wedge i < k \wedge c[i] > 5\}|$$

De lusinvariant en pre-en postcondities (hierboven in JML syntax) zijn:

$$\begin{aligned} I &\equiv k \leq |c| \wedge Tel(k) \\ P &\equiv c \neq \mathsf{null} \wedge \forall i : 0 \leq i \wedge i < |c| \Rightarrow c[i] \geq 0 \wedge c[i] \leq 10 \\ Q &\equiv Tel(|c|) \ . \end{aligned}$$

De bewijsverplichting bestaat uit de volgende onderdelen:

- *Pre-processing:* $\{P\}$ `r=0; k=0;` $\{I\}$.
  Dit kan worden bewezen met een weakest precondition:

$$\begin{aligned} & wp(\texttt{r=0; k=0;})\left\{k \leq |c| \wedge Tel(k)\right\} \\ & \quad \equiv 0 \leq |c| \wedge |\{i \mid 0 \leq i \wedge i < 0 \wedge c[i] > 5\}| = 0 \\ & \quad \equiv 0 \leq |c| \wedge \nexists i : 0 \leq i \wedge i < 0 \wedge c[i] > 5 \\ & \quad \equiv 0 \leq |c| \end{aligned}$$

  Aangezien dit wordt geïmpliceerd door P (namelijk door $c \neq \mathsf{null}$) zijn we klaar.
- *Invariantie:* $\{I \wedge k < |c|\}$ `if (c[k]>5) {r=r+1;} k=k+1;` $\{I\}$
  Opnieuw een weakest precondition-berekening:

$$\begin{aligned} & wp(\texttt{if (c[k]>5) \{r=r+1;\} k=k+1;})\left\{k \leq |c| \wedge Tel(k)\right\} \\ & \quad \equiv wp(\texttt{if (c[k]>5) \{r=r+1;\}})\left\{k < |c| \wedge Tel(k+1)\right\} \\ & \quad \equiv k < |c| \wedge \begin{pmatrix} (c[k] > 5 \wedge r + 1 = |\{i \mid 0 \leq i \wedge i \leq k \wedge c[i] > 5\}|) \\ \vee\ (c[k] \leq 5 \wedge r = |\{i \mid 0 \leq i \wedge i \leq k \wedge c[i] > 5\}|) \end{pmatrix} \\ & \quad \equiv k < |c| \wedge \begin{pmatrix} (c[k] > 5 \wedge r + 1 = |\{i \mid 0 \leq i \wedge i < k \wedge c[i] > 5\}| + 1) \\ \vee\ (c[k] \leq 5 \wedge r = |\{i \mid 0 \leq i \wedge i < k \wedge c[i] > 5\}|) \end{pmatrix} \\ & \quad \equiv k < |c| \wedge (c[k] > 5 \vee c[k] \leq 5) \wedge r = |\{i \mid 0 \leq i \wedge i < k \wedge c[i] > 5\}| \\ & \quad \equiv k < |c| \wedge Tel(k) \ . \end{aligned}$$

- *Post-processing:* $I \wedge k \geq |c| \Rightarrow Q$.
  Uit $I \wedge k \geq |c|$ volgt $k = |c|$, en dus is $I \Rightarrow Tel(|c|) \equiv Q$.

(c) (*5 punten*) Het gevraagde contract:

```
/*@ requires cijfers != null;
  @ requires ( \forall int i;
  @             0 <= i && i < cijfers.length;
  @             cijfers[i] >= 1 && cijfers[i] <= 10 );
  @ ensures ( \forall int c;
  @           1 <= c && c <= 10;
  @           \result[c] ==
  @              ( \sum int i;
  @                0 <= 1 && i < cijfers.length && cijfers[i] == c;
  @                1 ) );
  @*/
int[] telAlle(int[] cijfers) {
}
```