Tentamen Formele Methoden voor Software Engineering (192135201)

17 april 2014, 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-specificaties.

```
HOTEL1 = (kamer -> HOTEL1 | kamer -> stad -> HOTEL1).

HOTEL2 = (kamer -> UIT),
    UIT = (terug -> HOTEL2 | stad -> HOTEL2) \ {terug}.

GAST = (sleutel_G -> kamer -> UIT),
    UIT = (sleutel_B -> GAST | sleutel_B -> stad -> GAST).
BALIE = (sleutel_G -> sleutel_B -> BALIE).

||HOTEL3 = (GAST || BALIE) \ {sleutel_G, sleutel_B}.
```

   Elk van de processen `HOTELx` ($x = 1,2,3$) beschrijft het gedrag van een gast die naar zijn kamer gaat, en als hij er uit komt óf meteen weer terug naar zijn kamer, óf de stad in gaat.

   (a) *(8 punten)* Teken de transitiesystemen van `HOTEL1`, `HOTEL2`, `GAST`, `BALIE` en `HOTEL3`. Geef duidelijk aan hoe `HOTEL3` uit `GAST` en `BALIE` samengesteld is.

   (b) *(7 punten)* Welke van de drie bovenstaande `HOTELx`-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 op een `stad`-actie altijd een `kamer`-actie volgt, en teken het bijbehorende transitiesysteem. Welke van de `HOTELx`-processen voldoen *niet* aan deze eiggenschap, en waarom niet?

2. *(30 punten)* Een oud Nederlands gezegde luidt: "Eén ei is geen ei, twee ei is een half ei, drie ei is een paasei." Geleid door dit principe wil een `ETER` altijd drie eieren achter elkaar eten; daarvoor of daarna is hij bereid nieuwe eieren te kopen. Verder is er een proces `EIEREN` dat een doos met maximaal zes eieren modelleert: elk ei (`een`, `twee` of `drie`) dat gegeten wordt, vermindert het aantal eieren in de doos met 1; als de doos leeg is kan `koop` plaatsvinden, waarna de doos weer vol is. De actie `paasei` behoort niet tot het alfabet van `EIEREN`.

   Het proces `ETER` en de compositie van `ETER` en `EIEREN` zien er als volt uit:

```
ETER =
   ( een -> twee -> drie -> paasei -> ETER
   | koop -> ETER ).

||PASEN = ( ETER || EIEREN ).
```

   (a) *(5 punten.)* Schrijf het proces `EIEREN`.

   (b) *(10 punten.)* Geef een aangepast systeem `PASEN2` aan door een constante `E` te introduceren die het aantal eters weergeeft, en er voor te zorgen dat er inderdaad `E` onafhankelijke eters zijn, maar nog steeds maar één doos eieren. Welke fout treedt er op in `PASEN2`? Geef een zo gedetailleerd mogelijk antwoord.

   (c) *(5 punten.)* Geef een systeem `PASEN3` dat `PASEN2` aanpast door alle acties `twee` en `drie` een hogere prioriteit te geven; met andere woorden, een eter mag alleen aan zijn eerste ei beginnen als er niemand is die nog een tweede of derde ei blieft. Treedt dezelfde fout als in `PASEN2` nu nog steeds op? Leg je antwoord uit.

   (d) *(5 punten.)* Geef een systeem `PASEN4` dat `PASEN3` aanpast door de actie `een` van de jongste eter (d.w.z., met de laagste index) een *lagere* prioriteit te geven. Welke fout treedt er nu op? Geef een zo gedetailleerd mogelijk antwoord.

   (e) *(5 punten.)* Geef een *progress property* die uitdrukt dat er altijd wel iemand ooit de actie *paasei* kan doen. Is deze eigenschap in `PASEN4` vervuld? Leg je antwoord uit.

3. *(30 punten)* Beschouw de volgende Java-interfaces:

```
public interface Automaat {
    /** Kiest een frisdrank, als er momenteel
      * geen gekozen was, en levert de prijs op.
      */
    public int kies(Fris keus);

    /** Registreert een betaling van een bepaald bedrag, en levert
      * het nog te betalen restbedrag op, mits er een keuze is gemaakt.
      * Als het restbedrag niet-positief is, dan kan de gekozen
      * frisdrank geleverd worden.
      */
    public int betaal(int bedrag);

    /** Levert de gekozen frisdrank, mits er een keus gemaakt is
      * en hiervoor voldoende betaald is.
      */
    public Fris lever();

    /** Annuleert de bestelling, en levert het reeds betaalde bedrag op. */
    public int cancel();

}

public interface Fris {
    /*@ ensures \result > 0;
      @ pure
      @*/
    public int getPrijs();
}
```

(a) *(10 punten)* Stel een JML-contract op voor `Automaat`, waarbij de beoogde werking zoveel mogelijk formeel gespecificeerd is. Maak gebruik van modelvariabelen `keus` die het laatst gekozen (en nog niet geleverde) `Fris`-object bevat, en `rest` die het nog te betalen restbedrag bevat, als er een nog niet geleverde keus is.

(b) *(10 punten)* Schrijf een implementatie van bovenstaand interface, met voldoende JML-commentaar om de correctheid te kunnen bewijzen. Houd in de implementatie het totaal betaalde bedrag bij; het restbedrag wordt gerepresenteerd door het verschil tussen de frisprijs en het betaalde bedrag.

(c) *(10 punten)* Bewijs de correctheit van de methoden `betaal` en `lever` van je implementatie.

4. *(20 punten)* De volgende methode test of het stuk van de array `a` vanaf `aOff` tot `aOff+len` gelijk is aan het stuk van de array `b` vanaf `bOff` tot `bOff+len`.

```
boolean arrayCopy(int[] a, int aOff, int[] b, int bOff, int len) {
    boolean result = true;
    int k = 0;
    while (k < len && result) {
        result = a[aOff+k] == b[bOff+k];
        k = k+1;
    }
    return result;
}
```

(a) (*7 punten*) Voorzie deze methode van een contract dat de beoogde werking zo precies mogelijk vastlegt.

(b) (*13 punten*) Bewijs (met behulp van een lusinvariant) dat de methode aan haar contract voldoet.