

# 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 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(double 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. */
    double 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.
- (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];
}
```

- (c) (10 punten) Bewijs de correctheid van de (zelf geïmplementeerde) methode `declareer`.
- (d) (15 punten) Bewijs de correctheid van de methode `getTotaal`.

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 transitie-systemen van DIE1, DIE2 en DIE3.
- (b) (7 punten) Zijn DIE1 en DIE2 zwak bisimilaair? 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.

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

```

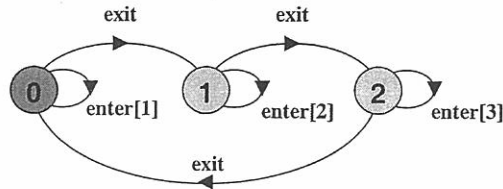
const NP = 2 range P = 1..NP // proces-indices
const NT = 3 range T = 1..NT // ticket-nummers

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 transitie-systeem van PROP1
- (b) (10 punten) Welke eigenschappen worden door PROP1 en PROP2 gespecificeerd?
- (c) (5 punten) Specificeer een FSP-proces GATE met het volgende gedrag:



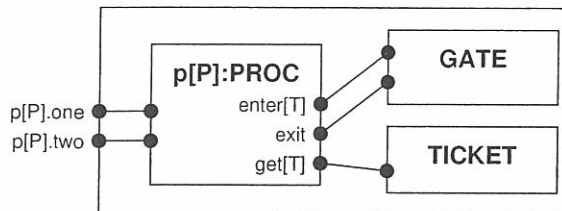
(d) (10 punten) Specificeer een systeem, bestaande uit instanties van het volgende proces PROC:

```

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.