Tentamen Concurrency, 7 februari 2007

Tijdsduur 3 uur. Gesloten boek tentamen.

NB Als een implementatie gevraagd wordt met (gesplitste binaire) semaforen of (simpele of compound) await statements, dan is busy waiting niet toegestaan.

Opgave 1 (40 %). Gegeven zijn gehele getallen M en N met $1 \leq M \leq N$. We beschouwen een systeem met N processen en twee gedeelde variabelen volgens

var n : *int* := 0 , b : *bool* := *false* ;

process *ThreadC* (*self* := 0 **to** $N-1$)
  do *true* →
10: *TNS1*
11: ⟨ **await** ¬b **then** n ++ ; b := (n = M) ⟩
12: ⟨ **await** b ⟩
13: *TNS2*
14: ⟨ n -- ; b := (n > 0) ⟩
  od
end *ThreadC* .

De commando's *TNS1* en *TNS2* zijn gegeven, eindigen gegarandeerd en modificeren de gedeelde variabelen b en n niet.
(a: 10 %) Formuleer en bewijs zo sterk mogelijke zinvolle invarianten van de vorm:

(J0) n = #{q | ...} ,
(J1) ¬b ⇒ "*iets over* n",
(J2) b ⇒ "*iets over* n",
(J3) q **in** { ... } ⇒ b .

(b: 10 %) Er gelden de voorgangscondities:

(V0) ¬b $o\!\!\rightarrow$ b ,
(V1) b $o\!\!\rightarrow$ ¬b .

Wat betekent dit (geef de definities)? Bewijs deze voortgangscondities.
(c: 20 %) Implementeer de atomaire commando's 11, 12, 14 van dit systeem met één of meer gesplitste binaire semaforen. Het moet hierbij mogelijk zijn dat meerdere processen tegelijk *TNS1* of *TNS2* (of beide) uitvoeren.

Uitwerking. (a)

(J0) n = #{q | q **in** {12, 13, 14}} ,
(J1) ¬b ⇒ n < M ,
(J2) b ⇒ $1 \leq$ n $\leq M$,
(J3) q **in** {13, 14} ⇒ b .

Een eventuele toevoeging $0 \leq$ n in (J1) is overbodig omdat dat uit (J0) volgt.

Bewijs van (J0): het geldt initieel; n wordt alleen opgehoogd bij passage van 11 en alleen verlaagd bij passage van 14 en dus blijft (J0) geldig.

Bewijs van (J1): Het geldt initieel en als b van true false wordt. (J1) blijft geldig bij verhoging van n omdat b zo nodig true wordt. (J1) blijft geldig bij het false maken van b in 14 omdat n dan 0 wordt.

Bewijs van (J2): Het geldt initieel omdat b dan false is. Het blijft geldig als b true wordt omdat dan n = M geldt. Zolang b geldt, kan n alleen verlaagd worden, maar niet tot 0, omdat b dan false wordt.

Bewijs van (J3): Het geldt initieel omdat alle processen dan bij 10 zijn. Als q in 13 komt, komt hij uit 12 en geldt b. Als een process in 13, 14 is en b geldt, kan

dit alleen false worden doordat `b` false wordt in 14, maar dan wordt `n` $= 0$ zodat er volgens (J0) geen processen meer in 13, 14 over blijven.

(b) (V0) betekent dat als ¬ `b` op zeker moment geldt, binnen een begrensd aantal ronden `b` geldt. (V1) betekent dat als `b` op zeker moment geldt, binnen een begrensd aantal ronden ¬ `b` geldt.

De geldigheid hiervan blijkt als volgt. (V0) Als ¬`b` geldt, dan is `n` $< M$ volgens (J1). Volgens (J0) zijn er dan $N -$ `n` processen in 10, 11; en volgens (J3) zitten de andere bij 12. Zolang `b` niet geldt, blijven de processen bij 12 wachten, terwijl de processen vanuit 10 en 11 binnen twee ronden bij 12 zijn. Dus moet intussen `b` true geworden zijn. NB. Let wel, het argument dat `n` regelmatig verhoogd wordt, is niet genoeg. Je moet ook aangeven, dat en waarom `n` niet ondertussen verlaagd weer wordt (terwijl ¬`b` geldt).

(V1) Als `b` geldt, zitten er `n` processen bij 12, 13, 14. De andere processen zitten bij 10, 11 en kunnen daar niet verder zolang `b` geldt. Binnen 3 ronden verlaten alle processen dus 12, 13, 14, wordt `n` $= 0$ volgens (J0) en wordt `b` false.

(c) We hebben drie binaire semaforen nodig: een voor de atomiciteitshaakjes, en één voor elke **await**. We hebben twee tellers nodig voor de wachtrijen bij de **await** locaties.

```
var cnt0 := 0 , cnt1 := 0 ;
sem gs := 1 , ba0 := 0 , ba1 := 0 ;
```

De body van de lus wordt:

```
TNS1 ;
P(gs) ; cnt0 ++ ; VA ;
P(ba0) ; cnt0 -- ; n ++ ; b := (n = M) ; cnt1 ++ ; VA ;
P(ba1) ; cnt1 -- ; VA ;
TNS2 ;
P(gs) ; n -- ; b := (n > 0) ; VA

VA = if cnt0 > 0 and not b  ->  V(ba0)
     [] cnt1 > 0 and b      ->  V(Ba1)
     [] else                ->  V(gs)
     fi
```

Opgave 2 (30 %). We beschouwen een systeem waarin $N > 1$ processen alle van tijd tot tijd een gedeelde variabele `x` moeten wijzigen met behulp van privé gegevens en een gegeven functie f volgens de specificatie:

var `x` : *Item* := `x0` ;

process *Thread*(*self* := 0 **to** $N - 1$)
  var *priv* : *Data*
  do *true* →
    `NCS` {kan *priv* wijzigen}
    ⟨ `x` := f(*priv*, `x`) ⟩
  od end .

Voorgesteld wordt dit te implementeren volgens:

var `list` : **set of** *Process* := ∅ ;

process *Thread*(*self* := 0 **to** $N - 1$)
  var *priv* : *Data* ; *it* : *Item* ; *bb* : *bool*
  do *true* →

```
10              NCS   {kan priv wijzigen}
11              bb := false
12              do ¬ bb  →
13                 ⟨ it := x ; list := list ∪ {self} ⟩ ;
14                 it := f(priv, it)
15                 ⟨ if  self ∈ list  →
                       x := it ; list := ∅ ; bb := true  fi ⟩
                od
            od end Thread .
```

Commando 13 heet LL (*load-linked*); commando 15 heet SC (*store-conditional*).
(a) Bewijs, dat een process p de binnenlus van 12 tot 15 dan en alleen dan verlaat als hij een toekenning aan x doet. Welke invariant heb je hier nodig?
(b) Bewijs, dat als process p een toekenning aan x doet, deze variabele de waarde $f(priv.p, \mathtt{x})$ krijgt. Formuleer en bewijs hiertoe geschikte invarianten.

Uitwerking. (a) Een proces p in de binnenlus kan die alleen verlaten (in 12, niet in 15) als $bb.p$ geldt. Bij het ingaan van de binnenlus geldt echter $\neg\, bb.p$. Gezien de guard van 12 en de toekenningen aan de privévariabele bb hebben we de invariant:

(K0) $\quad q$ **in** $\{13, 14, 15\} \;\Rightarrow\; \neg\, bb.q$.

Hieruit volgt dat process p de binnenlus dan en alleen dan verlaat als hij $bb.p$ waar maakt in 15. Dit is precies dan als zijn guard bij 15 geldt, en dat is precies dan als hij een toekenning aan x doet.

(b) We postuleren de invarianten:

(K1) $\quad q$ **at** $14 \;\wedge\; q \in \mathtt{list} \;\Rightarrow\; it.q = \mathtt{x}$,
(K2) $\quad q$ **at** $15 \;\wedge\; q \in \mathtt{list} \;\Rightarrow\; it.q = f(priv.q, \mathtt{x})$.

Als een process p een toekenning aan x doet, dan doet p dat in 15; (K2) zegt nu dat x dan de waarde $f(priv.p, \mathtt{x})$ krijgt, als gevraagd.

Invariantie van (K1): het geldt initieel. Als p bij 14 komt, heeft hij zojuist $it.p$ de waarde x gegeven. Als een proces p in 15 een nieuwe waarde aan x geeft, maakt hij $q \in \mathtt{list}$ onwaar, zodat (K1) behouden blijft.

Invariantie van (K2): het geldt initieel. Als p naar 15 gaat terwijl $p \in \mathtt{list}$ geldt, geeft hij aan $it.p$ de waarde $f(priv.p, it.p)$ terwijl $it.p = \mathtt{x}$ geldt volgens (K1). Dan wordt de consequent van (K2) dus geldig. Als een proces p in 15 een nieuwe waarde aan x geeft, maakt hij $q \in \mathtt{list}$ onwaar, zodat (K2) behouden blijft.

Opgave 3 (30 %). Beschouw een broadcast-systeem met één zendend proces en N ontvangende processen, die communiceren middels gedeelde variabelen volgens

```
var boodschap: Boodschap := null

process Zender
   var b: Boodschap
   do true ->
      b := verzin()
      boodschap := b
   od end Zender

process Ontvanger (self := 0 to N-1)
   var b: Boodschap
   do true ->
      b := boodschap
      verwerk(b)
   od end Ontvanger
```

Dit systeem dient met gedeelde variabelen zo gesynchroniseerd te worden, dat elke boodschap van de zender door *elke* ontvanger *precies één* keer ontvangen wordt.

Gebruik hiertoe simpele **await** statements en gedeelde en/of privé variabelen, zodanig dat geen van de processen onnodig hoeft te wachten. Je mag desgewenst onbegrensde integers gebruiken. Een gedeelde integer variabele mag atomair met 1 verhoogd of verlaagd worden. Maak de correctheid aannemelijk met behulp van invarianten.

Uitwerking. Er wordt hier een implementatie met gedeelde variabelen gevraagd, niet met boodschappen. Compound await statements zijn niet toegestaan (zie sectie 3.7 voor het verschil).

We gebruiken twee gedeelde variabelen: `cnt` telt het aantal ontvangers dat de huidige boodschap nog moet lezen en `sqn` is het volgnummer van de huidige of de eerstvolgende boodschap. Elke ontvanger heeft een privévariabele *own* met het volgnummer van de huidige of laatst gelezen boodschap.

```
     var boodschap: Boodschap := null
     var  cnt: int := 0
     var  sqn: int := 0

     process Zender
        var b: Boodschap
        do true ->
10         b := verzin()
11         < await cnt = 0 >
12         boodschap := b
13         < cnt := N >
14         < sqn ++ >
        od end Zender

     process Ontvanger (self := 0 to N-1)
        var b: Boodschap
        var own := 0
        do true ->
20         < await own != sqn >
21         b := boodschap
22         < cnt -- ; own ++ >
23         verwerk(b)
        od end Ontvanger
```

De twee toekenningen in 22 mogen in één atomair commando genomen worden, omdat *own* een privévariabele is. We doen dat omdat dat beter uitkomt in de invarianten hierna. De rollen van `cnt`, `sqn` en *own* worden gepreciseerd in de invarianten (we laten q lopen over de ontvangers):

(K0) zender **in** $\{10, 11\}$ $\Rightarrow$ $\texttt{cnt} = \#\{q \mid own.q \neq \texttt{sqn}\}$,
(K1) $\texttt{sqn} - 1 \leq own.q \leq \texttt{sqn}$.

Om (K0) en (K1) te behouden in 14 en 22 eisen we de invarianten

(K2) zender **in** $\{12, 13, 14\}$ $\Rightarrow$ $own.q = \texttt{sqn}$,
(K3) zender **at** 14 $\Rightarrow$ $\texttt{cnt} = N$,
(K4) q **in** $\{21, 22\}$ $\Rightarrow$ $own.q = \texttt{sqn} - 1$.

Uit (K0) volgt, dat zender in 11 wacht tot alle ontvanger gelezen hebben alvorens een nieuwe boodschap te zenden. De ontvangers wachten in 20 op een nieuwe boodschap; ze lezen elke boodschap dus maar één keer.

De integers `sqn` en *own* zijn onbegrensd. Je kunt het ophogen van `sqn` en *own* echter vervangen door ophogen modulo een getal $R \geq 2$. Het bewijs daarvan wordt niet gevraagd.