Tentamen Automated Reasoning, 31 oktober 2003

Tijdsduur 3 uur. Voorzie alle in te leveren bladen van je naam, en nummer ze. Schrijf op het eerste blad het aantal ingeleverde bladen. Werk netjes, formuleer scherp en zorgvuldig. Schrijf duidelijk leesbaar.

Opgave 1 (25 %). Beschouw een PVS-theorie met de typeparameter `T` en de declaratie

```
x, y: VAR T
rel: VAR pred[[T, T]]
```

(a) Formuleer een PVS-lemma met de volgende inhoud: als er een `x` is zodanig dat `rel(x, y)` voor elke `y` geldt, dan is er voor elke `y` een `x` waarvoor `rel(x, y)` geldt.
(b) Schets hoe dit lemma met PVS bewezen kan worden. Geef aan hoe de bewijsboom eruit gaat zien, wat de optredende sequenten zijn, en welke commando's je gebruikt om vanuit een sequent verder te komen. Je mag standaardsymbolen (als $\forall$) gebruiken voor PVS-keywords als `FORALL`, etc. Je hoeft de typering niet mee te slepen in de sequenten.

Opgave 2 (25 %). We coderen een vorm van lineaire temporele logica in PVS, gebruik makend van een type `T` voor de toestanden, volgens

```
pad: TYPE = [nat -> T]
i, n: VAR nat
xs: VAR pad
p, q: VAR pred[T]
rel: VAR pred[[T, T]]
pp, qq: VAR pred[pad]

suffix(xs, n): pad         = LAMBDA i: xs(i+n)
sem1(p): pred[pad]         = {xs | p(xs(0))}
sem2(rel): pred[pad]       = {xs | rel(xs(0), xs(1))}
box(pp): pred[pad]         = {xs | FORALL n: pp(suffix(xs, n))}
diamond(pp): pred[pad]     = {xs | EXISTS n: pp(suffix(xs, n))} ;
IMPLIES(pp, qq): pred[pad] = {xs | pp(xs) IMPLIES qq(xs)}

exec(rel): pred[pad]
fair(rel): pred[pad]
leadsto(p, q): pred[pad]
```

Voor een pad `xs` geeft `exec(rel)` aan dat alle stappen aan de relatie `rel` voldoen, `fair(rel)` geeft aan dat er oneindig veel stappen van het pad aan de relatie `rel` voldoen, `leadsto(p,q)` geeft aan dat elke toestand van het pad die aan `p` voldoet ook aan `q` voldoet of ooit gevolgd wordt door een toestand die aan `q` voldoet.

Geef definities voor de onderste drie padpredicaten overeenkomstig deze beschrijvingen met behulp van de functies die daarboven gegeven zijn. Motiveer je antwoord door informele beschrijvingen te geven van de gebruikte subexpressies.

Z.O.Z.

Opgave 3 (50 %). Dijkstra publiceerde in 1974 het eerste zelf-stabiliserende algoritme, dat als volgt beschreven kan worden. Gegeven zijn natuurlijke getallen $N < K$. Er is een ongeïnitialiseerd array `a[0..N]` van integers. Het algoritme is nondeterministisch met stappen die voldoen aan

do
$[\!]$ $i : i < N \quad \wedge \quad \mathtt{a}[i] \neq \mathtt{a}[i+1] \quad \rightarrow \quad \mathtt{a}[i] := \mathtt{a}[i+1]$
$[\!]$ $\mathtt{a}[N] = \mathtt{a}[0] \quad \rightarrow \quad \mathtt{a}[N] := (\mathtt{a}[0] + 1)$ **mod** K
od .

Het idee is dat $N + 1$ processen in een ring staan en elk hun eigen array-element hebben, en dat telkens wijzigen op grond van de waarde van de rechter buur. De zelf-stabilisatie houdt in dat de toestand op den duur voldoet aan het predicaat

(Stab) $\quad (\exists\, j \;::\; (\forall\, i : j \leq i : \mathtt{a}[i] = \mathtt{a}[N]) \;\wedge\; (\forall\, i : i < j : (\mathtt{a}[i] + 1)$ **mod** $K = \mathtt{a}[N]))$,

waarbij i en j over $[0\,..\,N]$ lopen. Als dit predicaat geldt, is het process met het nummer $(j-1)$ **mod** K als enige in staat om een stap te doen. Dit kan bv. voor wederzijdse uitsluiting gebruikt worden.

(a) Modelleer dit algoritme in Promela met behulp van $N+1$ processen die elk verantwoordelijk zijn voor de wijzigingen van één array-element. Zorg ten behoeve van de modellering dat elk process eerst zijn eigen array-element nondeterministisch initialiseert met een waarde $< K$. Geef aan hoe je met Spin test dat altijd tenminste één van de processen een stap kan doen.

(b) Geef temporele formules die uitdrukken:
(1) als de toestand eenmaal aan (Stab) voldoet, blijft dat zo.
(2) elke berekening geraakt op den duur in een toestand die aan (Stab) voldoet.
Gebruik hiertoe temporele formules als ingevoerd op het college of in opgave 2, of zoals Spin die accepteert.

(c) Geef aan hoe de beweringen (1) en (2) met Spin getest kunnen worden. Je kunt desgewenst variabelen aan je Promelaprogramma toevoegen. Berekeningen dienen niet onnodig inefficiënt te zijn.