Formale Grundlagen der Informatik III: Übung 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 2014/2015
Wird in der Übungsgruppe vom 03. Dezember bis 12. Dezember behandelt

Lösung 1 Algorithmus zur Umwandlung von LTL in einen Büchi Automaten

Während der letzten Übung haben Sie gelernt, wie man intuitiv eine einfache LTL Formel in einen Büchi Automaten umwandeln kann. Zur Verifikation von LTL Formeln muss diese Umwandlung allerdings algorithmisch möglich sein. Im Rahmen der Vorlesung wurde daher ein Algorithmus zur Umwandlung von LTL in einen Büchi Automaten vorgestellt. Versuchen Sie anhand des Algorithmus aus der Vorlesung die folgende Formel umzuwandeln:

- a **U** b

Lösung

Nach dem Algorithmus aus den Folien ergibt sich der folgende Ablauf. Eine größere Grafik wird in Form einer PDF über Moodle zur Verfügung gestellt!

Lösung 2 Temporale Logik

Wenn eine Anfrage auftritt, so wird diese entweder irgendwann bestätigt oder der anfragende Prozess wird niemals fortsetzen können.

Formalisieren Sie diese Eigenschaft in LTL. (Hinweis: Legen Sie erst ein Vokabular (Aussagenlogik) fest und drücken Sie dann die Eigenschaft mittels dieses Vokabulars aus)

Lösung

Festlegung der Variablen:
R $\hat{=}$ Request wurde durchgeführt, A $\hat{=}$ Request wurde bestätigt, P $\hat{=}$ der anfragende Prozess schreitet fort
$R \rightarrow (\Diamond A \vee \Box \neg P)$

Lösung 3 Formalisieren in LTL

Wir betrachten eine Kreuzung bestehend aus zwei Straßen und einer Ampel (Nr. 1) für die Nord-Süd Straße und einer Ampel (Nr. 2) für die West-Ost Straße. Nehmen Sie an, dass die folgenden aussagenlogischen Variablen vorliegen:
R1 (Ampel 1 ist rot)
G1 (Ampel 1 ist grün)
R2 (Ampel 2 ist rot)
G2 (Ampel 2 ist grün)

Geben Sie die LTL Formeln für die folgenden Eigenschaften an:

a) Beide Ampeln sind nie zur selben Zeit grün.

Lösung

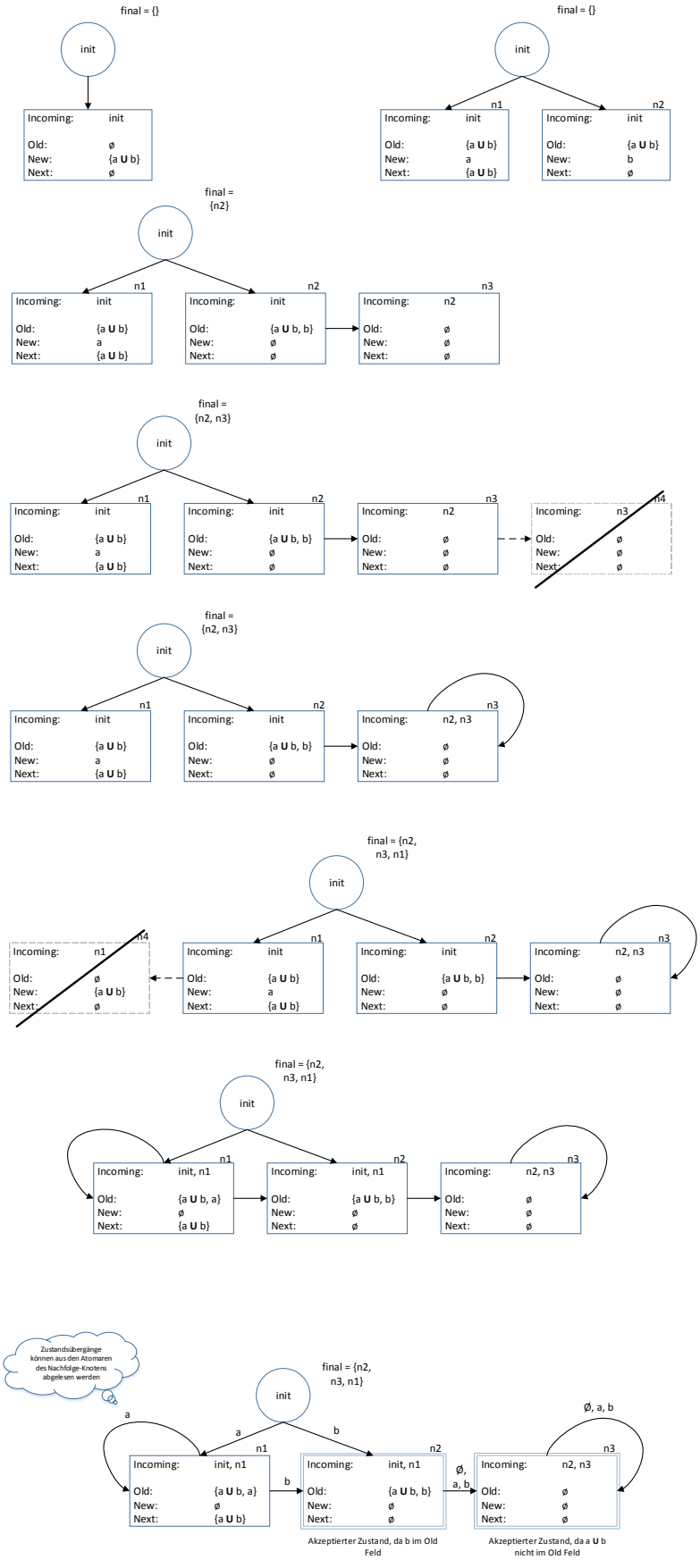
$\Box(\neg G1 \vee \neg G2)$

b) Aus jedem Zeitpunkt kann jede Ampel irgendwann grün werden.

Lösung

$\Box(\Diamond G1 \wedge \Diamond G2)$

Abbildung 1: Ablauf zu Aufgabe 1



c) Ampel 1 bleibt rot, bis Ampel 2 rot geworden ist.

Lösung

$R1 \cup R2$

Lösung 4 Nebenläufigkeit: Shared Variables

Definieren Sie einen Prozess P, der eine globale Variable b wie folgt erhöht:

```
byte b = 0;

proctype P(){
  b++
}
```

Starten Sie zwei Instanzen von P. Warten sie anschließend, bis diese fertig sind und führen Sie anschließend die Assertion (b == 2) durch. Wird diese true ergeben?

Verändern Sie P wie folgt:

```
byte b = 0;

proctype P(){
  byte tmp = b;
  tmp++;
  b = tmp
}
```

Wird die Assertion in diesem Fall true sein? Wenn nicht, können Sie den obigen Block so modifizieren, dass er true ergibt.

Lösung

```
proctype p2() {
  byte tmp;
  atomic {
    tmp  = b;
    tmp ++;
    b = tmp
  }
}


init {
     atomic{
       run p2();
       run p2();
     }
     _nr_pr == 1 -> assert(b == 2)
}

/* _nr_pr==1 ensures all other processes have terminated */
```

Lösung 5 Rendez-Vous Channels

Betrachten Sie das folgende PROMELA Model:

```
chan com = [0] of {byte}
byte value = 0;

proctype p() {
  byte i = 0;
  do
    :: if
       :: i >= 5 -> break
       :: else   -> printf("Doing something else\n");
                    i++
       fi
    :: com ? value ;
    printf("p received: %d\n",value)
  od
}

init {
  atomic { run p() }
end:
  com!100
}
```

a) Formulieren Sie eine LTL Eigenschaft, die besagt, dass **value** = 100 ist, wenn Prozess **p** terminiert. Benutzen Sie SPIN um zu zeigen, dass Ihre Eigenschaft gültig ist. Sie dürfen den Code modifizieren und Ghost variables, labels oder ähnliches einfügen, falls Sie dies zur Verifikation benötigen.

Lösung

```
#define prop_value (value == 100)

chan com = [0] of { byte };

byte value = 0;

proctype p() {
  byte i = 0;
  do
  :: if
     :: i >= 5 -> break
     :: else -> printf("Doing something else\n");
                i++
     fi
  :: com ? value;
     printf("p received: %d\n",value)
  od;
terminated:
  skip
}

init {
     atomic {
        run p();
     }
     end:
        com ! 100;
```

```
}

ltl valueRead {
  [](p@terminated -> prop_value)
}
```

b) **Zusatzaufgabe:** Schreiben Sie ein Modell, in dem **drei** Prozesse über **einen** Rendezvous-Channel den Zugriff auf eine critical section verwalten. Der Prozess, der Zugriff auf die critical section hat gibt nach dem Verlassen der critical section den Zugriff gezielt an einen der beiden anderen Prozesse (über den Channel) weiter. Sie können die Weitergabe in eine Richtung oder auch nicht-deterministisch realisieren. Es darf jedoch nur ein einziger Channel zur Autorisierung verwendet werden.

Lösung

```
#define M 3

mtype = {tokenPush} /* msg type for the channel message */

/* msg type, rcvr, count of tokens */
chan tokenBus = [0] of {mtype, byte, byte}

active [M] proctype Process(){
  byte id = _pid;
  byte left =  (id - 1 + M) % M;
  byte right = (id + 1) % M;
  byte tokens = 0;
  byte outgoing = 0;

processWaits:
  if
    :: tokenBus ? tokenPush(eval(id), tokens)
  fi;
processWorks:
  printf("Process␣%d␣fired␣-␣%d\n", id, tokens);
  outgoing = tokens;
  tokens = tokens - outgoing;
  if
    :: tokenBus ! tokenPush(left, outgoing)
    :: tokenBus ! tokenPush(right, outgoing)
  fi;
  goto processWaits
}


active proctype Initializer(){
  tokenBus ! tokenPush(1, 1)
}
```

Lösung 6 CBMC Primer: Programmieren mit C

Zur Vorbereitung auf die Vorlesung und das Lab zu CBMC werden wir in dieser Übung einige erforderliche Grundlagen zur C Programmierung schaffen. Sollten Sie bereits Erfahrungen in der Programmierung mit C haben, so können Sie direkt zu Teilaufgabe c)/d) übergehen.

a) Lesen Sie sich das folgende Dokument zur Programmierung in C durch:
http://www.cs.cornell.edu/courses/cs414/2005sp/cforjava.pdf

b) Schreiben Sie zum Einstieg ein einfaches Programm, welches den Text **Hello World!** ausgibt

Lösung

```
#include<stdio.h>

main()
{
    printf("Hello World");
}
```

c) Schreiben Sie eine Funktion, welche zwei Integer übergeben bekommt und zu diesen den größten gemeinsamen Teiler bestimmt und zurückgibt.

Lösung

```
#include <stdio.h>

/* Iterative Variante */
int gcd (int a, int b){
  int c;
  while (a != 0){
     c = a;
     a = b % a;
     b = c;
  }
  return b;
}

/* Rekursive Variante */
int gcd_rec (int a, int b){
  if (a == 0) return b;
  return gcd_rec (b % a, a);
}
```

d) Testen Sie die korrekte Arbeitsweise Ihrer Funktion händisch.

Lösung

```
int main(void)
{
  int a = 299792458;
  int b = 6447287;
  int c = 256964964;

  printf("a = %d, b = %d, c = %d\n", a,b,c);

  printf("gcd(a,b) = gcd(%d, %d) = %d\n", a, b, gcd(a,b));

  printf("gcd(a,c) = gcd(%d, %d) = %d\n", a, c, gcd(a,c));

  printf("gcd(c,b) = gcd(%d, %d) = %d\n", c, b, gcd(c,b));

  return 0;
}
```

e) *Optional:* Zur Vertiefung Ihrer C Kenntnisse können Sie das folgende Dokument durcharbeiten:
https://www.site.uottawa.ca/~ivan/CSI3131/files/c-introMaassen.pdf