Formale Grundlagen der Informatik III: Übung 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 2014/2015

Wird in der Übungsgruppe vom 19. November bis 28. November behandelt

Lösung 1 Formalisierung in Aussagenlogik

Basierend auf Raymond Smullyan's "Knights and Knaves":

In earlier times there existed an island whose inhabitants were either knights or knaves. A knight always tells the truth, while a knave always lies Hedwig and Katrin lived on that island. A historian found in the archives the following statements:
Hedwig says I am knave if and only if (iff) Katrin is a knave.
Katrin says We are of different kind.

Formalisieren Sie die obigen Aussagen in Aussagenlogik.

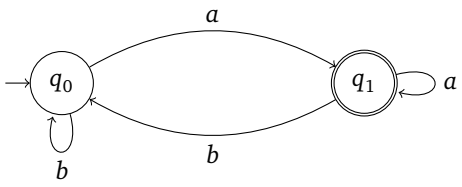
Lösung

H : Hedwig is a knight, K : Katrin is a knight
Hedwig: $H \leftrightarrow (\neg H \leftrightarrow \neg K)$
Katrin: $K \leftrightarrow \neg(H \leftrightarrow K)$

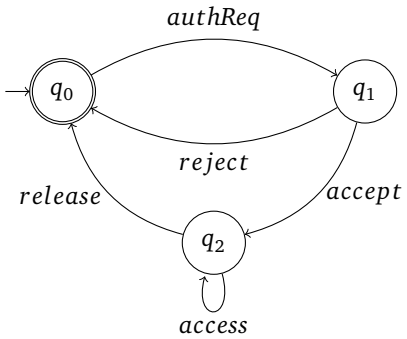
Lösung 2 Büchi Automaten

a) Welche Sprache wird von den folgenden Büchi Automaten akzeptiert?

1.



2.



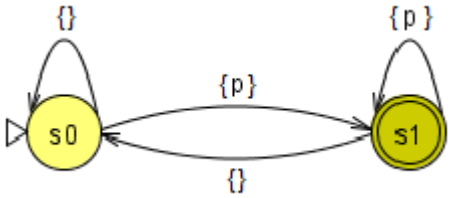
Lösung

1. $(b^*a)^\omega$
2. $(\text{authReq}\ (\text{reject} + \text{accept}\ \text{access}^*\ \text{release}))^\omega$

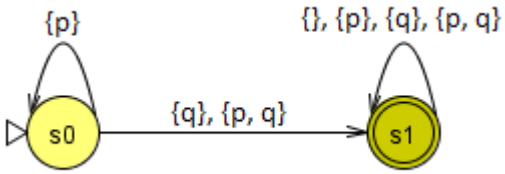
b) Wandeln Sie die folgenden LTL Formeln in Büchi Automaten um.
Verwenden Sie hierfür die Alphabete:
$\Sigma_1 := \{\emptyset, \{p\}\}$
$\Sigma_2 := \{\emptyset, \{p\}, \{q\}, \{p, q\}\}$

1. **GF**p
2. p **U** q

1.



2.



Lösung 3 Else oder True

Wir betrachten die folgenden PROMELA Programme:

```
active proctype TRUE(){
  if
    :: false -> printf("1\n")
    :: true -> printf("2\n")
  fi
}
```

und

```
active proctype ELSE(){
  if
    :: false -> printf("1\n")
    :: else -> printf("2\n")
  fi
}
```

Wenn wir diese Programme ausführen, können wir beobachten, dass Sie beide immer 2 als Ausgabe liefern. Von diesem Beispiel ausgehend: Können wir ableiten, dass man **else** durch **true** in unseren PROMELA Programmen ersetzen kann? Geben Sie Beispiele um Ihre Antwort zu unterstützen!

Lösung

```
active proctype ELSE() {
  if
    :: true -> printf("1\n")
    :: true -> printf("2\n")
  fi
}
```

Anmerkungen:

a) Die Aufgabenstellung fordert, dass die Ausgabe immer **2** zeigt. Wir können dies auch über ein Assert verifizieren, indem wir den Statements, welche **1** ausgeben jeweils ein **assert(false)** als letztes Statement hinzufügen und dann Folgendes ausführen:

```
spin -a true.pml
gcc -o pan pan.c
./pan
```

Hier erhalten wir die Ausgabe **Errors: 0**, was unsere Annahme beweist.

Etwas indirekter erhalten wir diese Information von SPIN allerdings auch schon ohne die Assertions. Wenn wir obige Kommandos ausführen erhalten wir von SPIN einen Hinweis auf **unreachable statements**. Die nicht erreichbaren Anweisungen entsprechen hierbei den Ausgabe der Zahl **1**. Beispielsweise:

```
unreached in proctype TRUE
        true.pml:3, state 2, "printf('1\n')"
        (1 of 7 states)
```

b) Nein, wir können nicht einfach **else** durch **true** ersetzen. In dem obigen Beispiel führte dies zwar zum gewünschten Ergebnis, allerdings nur, da alle anderen Fälle durch false nicht ausführbar waren. Für weitere Details sehen Sie sich bitte den entsprechenden Abschnitt in den Folien und das obige Beispiel an.

Lösung 4 Interleaving

Betrachten Sie das folgende PROMELA Modell. Wie viele Möglichkeiten für Interleaving existieren? Was sind die Endergebnisse für x, y und z bei jeder Möglichkeit?

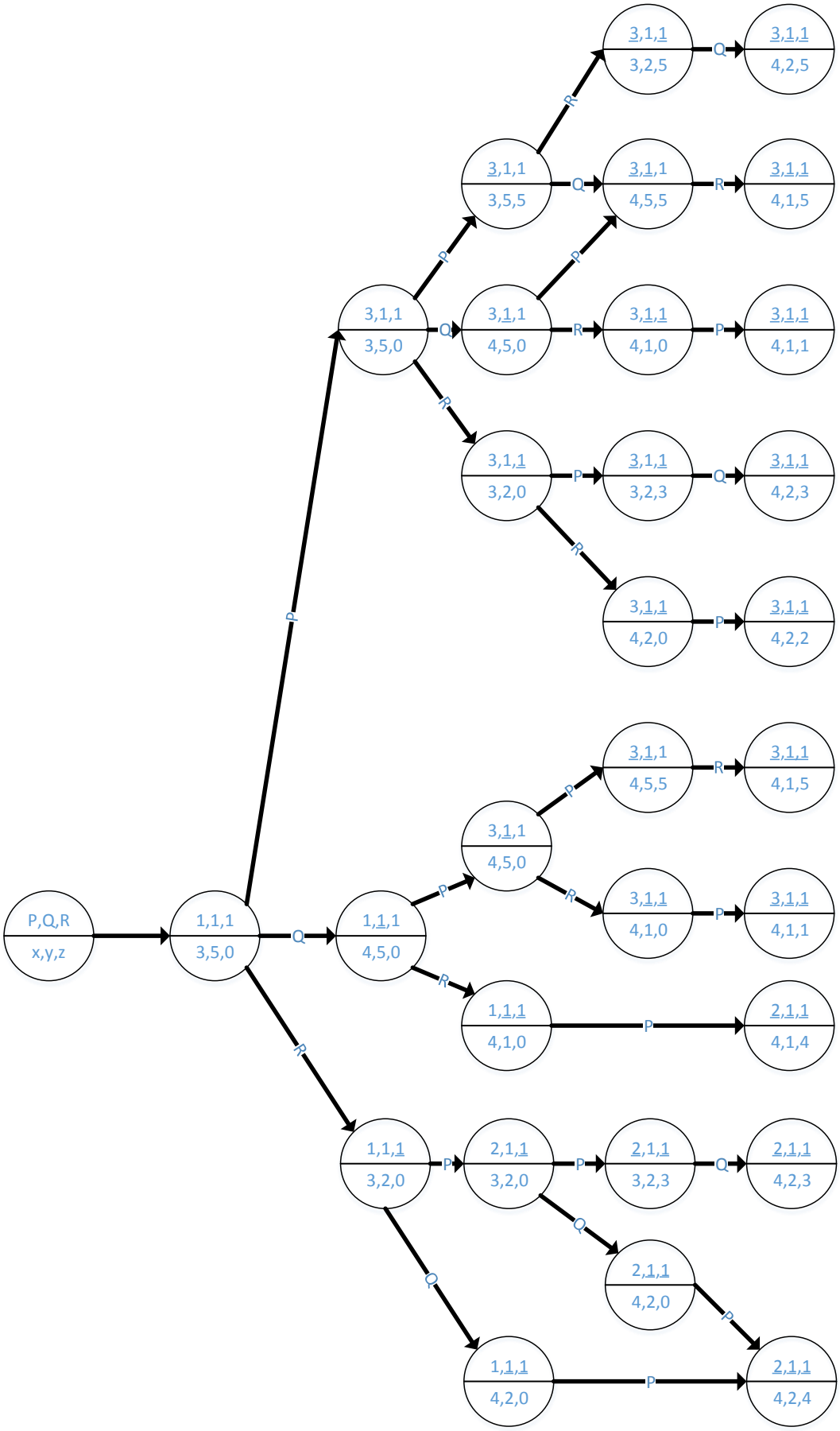
```
int x = 3;
int y = 5;
int z = 0;

active proctype P(){
  if
    :: x > y -> z = x
    :: else -> z = y
  fi
}

active proctype Q(){
  x++
}

active proctype R(){
  y = y - x
}
```

Lösung


P,Q,R
x,y,z
1,1,1
3,5,0
P
Q
R
3,1,1
3,5,0
3,1,1
3,5,5
3,1,1
4,5,5
3,1,1
3,2,5
3,1,1
4,2,5
3,1,1
4,5,5
3,1,1
4,1,5
3,1,1
4,5,0
3,1,1
4,1,0
3,1,1
4,1,1
3,1,1
3,2,0
3,1,1
3,2,3
3,1,1
4,2,3
3,1,1
4,2,0
3,1,1
4,2,2
1,1,1
4,5,0
3,1,1
4,5,5
3,1,1
4,1,5
3,1,1
4,5,0
3,1,1
4,1,0
3,1,1
4,1,1
1,1,1
4,1,0
2,1,1
4,1,4
1,1,1
3,2,0
2,1,1
3,2,0
2,1,1
3,2,3
2,1,1
4,2,3
2,1,1
4,2,0
1,1,1
4,2,0
2,1,1
4,2,4

Lösung 5 Verifikation mit SPIN

Gegeben sei folgendes Modell:

```
byte mode = 1;
byte count = 0;

active proctype m(){
endLoop:
  if
    :: mode = 1
    :: mode = 2
  fi;
  do
    :: mode == 1 && count < 30 -> count++
    :: mode == 2 -> count = 0; goto endLoop
    :: mode == 3 -> break
    :: else -> endLoop;
  od;
  count = 0
}

active proctype n(){
  do
    :: mode = 3
  od
}
```

Formalisieren Sie die folgenden Eigenschaften in LTL. Zeigen Sie mit SPIN ob diese im obigen Modell mit und ohne weak-fairness gültig sind. Erklären Sie Ihre Beobachtungen.

a) **count** ist niemals größer-gleich 30.

b) wenn **count** in einem Zustand größer als 0 wird, so bleibt es positiv, solange bis **mode** größer als 1 wird.

c) wenn **count** in einem Zustand größer 0 ist, wird es zu einem späteren Zeitpunkt irgendwann auf 0 zurückgesetzt.

d) **mode** wird irgendwann 3 werden.

Lösung

```
/*
For an action A,
Weak Fairness: [] enabled A -> <> taken A
(Strong Fairness: []<> enabled A -> <> taken A)

Satisfiability of LTL properties:
a: not satisfied
b: satisfied with weak fairness
c: not satisfied
d: satisfied with weak fairness
*/

byte mode  = 1;
byte count = 0;

active proctype m() {

endLoop:
  if
    :: mode = 1
    :: mode = 2
  fi;
  do
    :: mode == 1 && count < 30 -> count++
    :: mode == 2 -> count = 0; goto endLoop
    :: mode == 3 -> break
    :: else -> goto endLoop
  od;
  count = 0
}

active proctype n() {
  do
   :: mode = 3
  od
}

ltl a { !<>(count >= 30) }
ltl b { [](count>0 -> ((count>0)U(mode > 1)))}
ltl c { [](count>0 -> <>(count == 0)) }
ltl d { <>(mode == 3) }
```

Lösung 6 Drucker Zugriff

Modellieren Sie ein Büro in dem zwei Computer mit einem einzigen Drucker verbunden sind. Realisieren Sie hierbei einen mutual exclusion zwischen den beiden Computern durch einen rendezvous channel.

a) Verifizieren Sie, dass niemals beide Systeme zur gleichen Zeit den Drucker verwenden können.

Lösung

```
/** establishes mutual exclusion via a rendezvous channel
    One using ghost variables, the other labels.
    This solution does not prevent starvation.
*/

mtype = {request, release}

chan chanPrinter = [0] of { mtype };

/* ghost fields for verification purposes */
byte printing = 0;

active proctype printer() {
  do
    :: chanPrinter ? request;
       chanPrinter ? release
  od
}

active [2] proctype user() {
  do
    :: chanPrinter ! request ->
          printing++;
          printf("user␣%d␣is␣printing\n", _pid);
          printing--;
          chanPrinter ! release
  od
}

active proctype verifier() {
  assert(printing <= 1)
}
```

b) Erweitern Sie Ihr Modell so, dass es niemals zu Starvation kommt (ein Prozess, der Drucken möchte muss dies auch irgendwann dürfen).

c) Formulieren Sie eine Korrektheitseigenschaft gegen Starvation in LTL und verifizieren Sie diese mit SPIN.

Lösung

```
mtype = {request, access, release}

chan chanRequest = [2] of { mtype, byte };
chan chanReply = [0] of { mtype, byte };

/* ghost fields for verification purposes */
byte printing = 0;
bool waiting[2];

 proctype printer() {
```

```
  byte proc_nr;
endPrinter:
  do
    :: chanRequest ? request(proc_nr);
       chanReply ! access(proc_nr);
       chanReply ? release(eval(proc_nr));
  od
}

 proctype user() {
endUser:
  do
    :: atomic {
       waiting[_pid-1] = true;
       chanRequest ! request(_pid);
       } ->
userWaits:
       chanReply ? access(eval(_pid));
       printing++;
userPrints:
       printf("user␣%d␣is␣printing\n", _pid);
       printing--;
       waiting[_pid-1] = false;
       chanReply ! release(_pid);
  od
}

init {
  atomic {
    run user();
    run user();
    run printer()
  }
}

ltl mutualGhost {
  [] (printing <= 1)
}

ltl mutualWithoutGhost {
  [] (!user[1]@userPrints || !user[2]@userPrints)
}

ltl noStarvation{
  [](( user[1]@userWaits -> <>user[1]@userPrints) &&
(user[2]@userWaits -> <>user[2]@userPrints))
}

ltl noStarvation2{
  [](( waiting[0] -> <>!waiting[0]) && (waiting[1] -> <>!waiting[1]))
}
```