Formale Methoden im Software Entwurf
WS19/20 Übung 1

May 29, 2020

2 Nichtdeterministische Bedingte Anweisung (non-deterministic conditional statement)

2.1 Implementierung in PROMELA

a)

```
inline selectEvent(event) {
    if
    :: event = login;
    :: event = logout;
    fi
}
```

Das folgende Codesegment wählt auf pseudozufälliger Weise einer der Unterstatements des *If-Statements* aus. Auf dieser Art und Weise wird dem übergebenem Parameter 'event' ein pseudozufälliger Wert zugewiesen, in diesem Fall entweder *login* oder *logout* vom Typ *mtype*.

b)

```
proctype AuthenticationOld() {
    // counts failed authentication tries
    // replace <T1> by a suitable type
    byte fail = 0;
    // current state
    // replace <T2> by a suitable type
    byte currentState = 0;
    // received event
    mtype ev;
    // TODO: IMPLEMENT

    state0:
    selectEvent(ev);
    if
    :: fail >= 3; currentState = 2; goto state2;
    :: else;
        if
        :: ev == login;
            if
            :: fail = fail + 1;
            :: currentState = 1; goto state1;
            fi
        fi
    fi
    state1:
    selectEvent(ev);
    if
    :: ev == logout; currentState = 0; goto state0;
    fi
    state2:
    selectEvent(ev);
}
```

2.3

```
active proctype Authentication() {
    // counts failed authentication tries
    // replace <T1> by a suitable type
    byte fail = 0;
    // current state
    // replace <T2> by a suitable type
    byte currentState = 0;
    // received event
    mtype ev;
    // TODO: IMPLEMENT

    state0:
    if
    :: fail >= 3; currentState = 2; goto state2;
    :: else;
        if
        :: ev = login;
            if
            :: fail = fail + 1;
            :: currentState = 1; goto state1;
            fi
        :: true; false;
        fi
    fi
    state1:
    if
    :: ev = logout; currentState = 0; goto state0;
    :: true; false;
    fi
    state2:
}
```

3 Knapp vor der Deadline

```
if
  :: result = result + 1;
  :: result = result + 2;
  :: result = result + 3;
  :: result = result + 4;
  :: result = result + 5;
  :: else -> result = result + 6;
fi;
```

In diesem Codeabschnitt, muss das *else* durch ein *true* ersetzt werden(alternativ kann einfach der ganze Guard entfernt werden. Das liegt daran, dass Ausdrücke, die keine Vergleiche oder Variablen sind, immer als *true / 1* ausgewertet werden. Folglich wird dieser Codebereich nie erreicht und der Spieler könnte nie ein 6 würfeln.

```
do
  :: dice == 12 -> printf("Won\n"); goto gameEnd
  :: dice != 12 && times < 1 -> rollDiceTwice(dice); times = times + 1
  :: true -> printf("Lost\n"); goto gameEnd
od;
```

In diesem Code-Abschnitt muss das *true* mit einem *else* ersetzt werden, da sonst die Situation modelliert werden kann, dass der Spieler ehe er überhaupt mal würfeln durfte schon verloren hat.

5 Interleaving

a)

We have following interleaving possibilities:

$$P \to P \to Q \to R$$
$$P \to P \to R \to Q$$
$$P \to Q \to P \to R$$
$$P \to Q \to R \to P$$
$$P \to R \to P \to Q$$
$$P \to R \to Q \to P$$
$$Q \to P \to P \to R$$
$$Q \to P \to R \to P$$
$$Q \to R \to P \to P$$
$$R \to P \to P \to Q$$
$$R \to P \to Q \to P$$
$$R \to Q \to P \to P$$

b)

$$P \to P \to Q \to R$$

$$x = 4, y = 1, z = 5$$

a

$$P \to P \to R \to Q$$

$$x = 4, y = 2, z = 5$$

b

$$P \to Q \to P \to R$$

$$x = 4, y = 1, z = 5$$

a

$$P \to Q \to R \to P$$

$$x = 4, y = 1, z = 1$$

c

$$P \to R \to P \to Q$$

$$x = 4, y = 2, z = 2$$

d

$$P \to R \to Q \to P$$

$$x = 4, y = 2, z = 2$$

d

$$Q \to P \to P \to R$$

$$x = 4, y = 1, z = 5$$

a

$$Q \to P \to R \to P$$

$$x = 4, y = 1, z = 1$$

c

$$Q \to R \to P \to P$$

$$x = 4, y = 1, z = 4$$

e

$$R \to P \to P \to Q$$
$$x = 4, y = 2, z = 3$$

f

$$R \to P \to Q \to P$$
$$x = 4, y = 2, z = 4$$

g

$$R \to Q \to P \to P$$
$$x = 4, y = 2, z = 4$$

g

All possible states:

$$x = 4, y = 1, z = 5$$
$$x = 4, y = 2, z = 5$$
$$x = 4, y = 1, z = 1$$
$$x = 4, y = 2, z = 2$$
$$x = 4, y = 1, z = 4$$
$$x = 4, y = 2, z = 3$$
$$x = 4, y = 2, z = 4$$