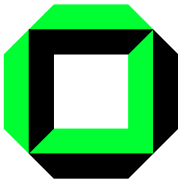

**Universität Karlsruhe**
Intitut für Theoretische Informatik
*Prof. Dr. P. H. Schmitt*
*C. Engel, F. Werner*

28. November 2006

# Praktikum
# Formale Entwicklung objektorientierter Software
Übungsblatt 4

## Aufgabe 11

Verifizieren Sie in KeY, ob die in Aufgabe 5 implementierten Operationen der Chipkarte bzw. des Kartenlesers, die zum Eingeben und Überprüfen der PIN dienen, die Spezifikationen erfüllen. Sollten Sie dabei Fehler in der Spezifikation/Implementierung finden, korrigieren Sie diese.

## Aufgabe 12: Installation Uppaal

Sie können Uppaal von Ihren Accounts im Rechnerraum ausführen oder lokal bei sich Zuhause installieren. Eine aktuelle Installation erhalten sie unter `http://www.uppaal.com`. Um Ihnen die Arbeit zu erleichtern - und da wir wissen, daß sie die Arbeit mit Uppaal kaum erwarten können - haben wir bereits die aktuelle Version für Sie bereitgestellt unter `/home/werner/uppaal/uppaal`.

## Aufgabe 13: Einführung und Umgang mit Uppaal

Starten Sie im folgenden Uppaal und machen Sie sich mit der Oberfläche vertraut. Sie finden im Unterverzeichnis `demo` einige Beispiele, von denen Ihnen das *TrainCrossing* Modell aus der Vorlesung bekannt vorkommen sollte. Öffnen Sie *TrainCrossing* und schauen sie sich im **Editor** die Modell Definition an. Starten Sie den **Simulator** und machen sie sich mit der Interaktion und verschiedenen Simulationsdurchläufen vertraut.
Versuchen sie im **Verifier** die vordefinierten Properties zu verifizieren; definieren Sie eigene Queries, die Sie verifizieren können, oder auch nicht. Hilfestellung finden Sie sie in der Uppaal Anleitung[1].

## Aufgabe 14: Das Druzba MUTEX Problem OHNE Fairness

In der Vorlesung wurde Ihnen das *Druzba MUTEX Problem* vorgestellt. Modellieren sie das Problem neu für $n = 5$ Hotelgäste. Jeder Gast, der duschen möchte, geht dabei auf den Flur. Auf dem Flur stehen, ist dabei jedoch keine Garantie (*ohne Safety*), daß die Dusche benutzt werden kann. Es kann auch passieren, daß andere Gäste später kommen und doch vorher duschen.
Besonders interessant sind dabei die folgenden temporalen Operatoren `A[]` invariantly, `E<>` possibly, `A<>` always eventually, `E[]` potentially always). Formulieren und verifizieren Sie nun die folgenden Eigenschaften in CTL* als richtig oder falsch.

[1] `http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf`

Als Beispiel nehmen wir die Eigenschaft beschrieben durch `E<> Hotelguest(1).Corridor and Hotelguest(2).Shower`, die sich wie folgt verbalisieren läßt: *Existiert ein Zustand, in dem Hotelgast 1 sich auf dem Flur befindet, während Gast 2 duscht?*. Sie können auch für große Anfragen All- und Existenz-Quantoren benutzen um Ihnen die Arbeit zu erleichtern: `E<> forall (i: id_t) exists (j:id_t) Hotelguest(i).Shower imply Hotelguest(j).Shower`.

(a) Safety:
*Es gibt im System keine Deadlocks*

(b) Safety:
*Ein Hotelgast geht nach 5 Zeiteinheiten entweder in die Dusche oder zurück in sein Zimmer.*

(c) Liveness:
*Es ist möglich, daß ein Hotelgast sein Zimmer nie verläßt.*

(d) Mutex Requirement:
*Es kann immer nur ein Hotelgast duschen (duschen Hotelgäste i und j gleichzeitig, so muß i=j sein).*

(e) Bounded Liveness:
*Hotelgast 1 kann höchstens bis zur Zeit 15 in der Dusche bleiben.*

(f) Fairness:
*Ist das System fair, d.h. es soll nicht möglich sein, dass ein späterer Gast einen früheren beim Duschen überholen kann?*

## Aufgabe 15: Das Druzba MUTEX Problem MIT Fairness

Da einige Gäste erbost über das System von Hotel *Druzba* waren, überlegte sich die Hotelleitung nun folgendes: Jeder Gast, der die Dusche benutzen möchte stellt sich an der Schlange der bereits wartenden Gäste auf - ähnlich wie es von der Wursttheke im Supermarkt Ihres Vertrauens bekannt sein sollte - und wartet in der Schlage im Korridor bis er an der Reihe ist. Es gelten nachwievor die gleichen Bedingungen, d.h. wartet ein Gast länger als 5 Zeiteinheiten, ohne an die Reihe zu kommen, kehrt er in sein Zimmer zurück und versucht es später nocheinmal.
Überprüfen Sie im folgenden die Properties (a) bis (f) aus der vorherigen Aufgabe. Was stellen sie fest? Wie sieht es mit der Komplexität Ihres Modells aus. Läßt sich Eigenschaft (f) noch für $n = 7$ verifizieren? Für welches $n$ können sie Eigenschaft (f) noch beweisen?

**Hinweis:** Benutzen sie ein Array der Größe $n$ um die Schlange zu modellieren und 2 Hilfsfunktionen zum Hinzufügen und Entfernen der Teilnehmer aus der Schlange.

**Abgabe bis Montag, den 18.12.2006 - 12:00 MEZ**
Es braucht pro Gruppe nur *eine* Lösung abgegeben werden.
Ihre Uppaal Modelle (`xml`-Dateien) legen sie bitte mit kommentierten Queries (`.q`-Dateien) in einem Ordner in Ihrem Abgabeverzeichnis ab, z.B. `Aufgabe_<Nr>.xml`. Einige Aufgaben verlangen eine schriftliche Bearbeitung, diese ist dann je nach Komplexität als ASCII, html, ps- oder pdf-Dokument abzugeben. Auf *keinen* Fall im MS Word `doc`-Format.

# Praktikum
# Formale Entwicklung objektorientierter Software
Übungsblatt 11 – Lösungen

## Aufgabe 14: Das Druzba MUTEX Problem OHNE Fairness

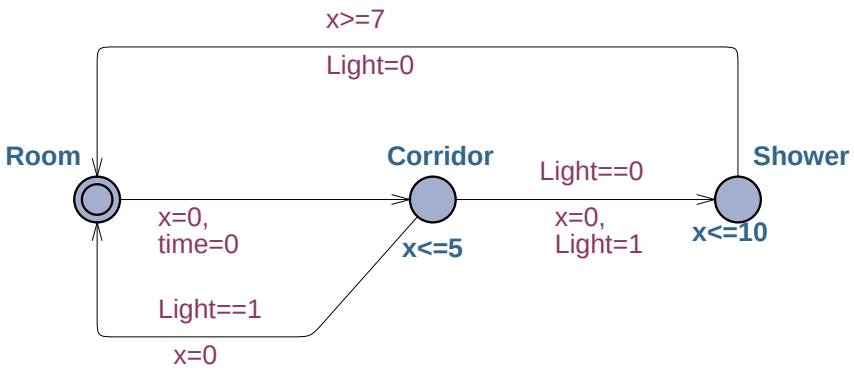


Abbildung 1: Druzba Hotel Modell ohne Fairness.

**Eigenschaften** Folgende Eigenschaften sind erfüllt ($\surd$) oder nicht erfüllt ($\neg\surd$):

(a) Safety: ($\surd$)
```
A[] not deadlock
```

(b) Safety: ($\surd$)
```
Hotelguest(0).Corridor and Hotelguest(0).time>5 -> Hotelguest(0).Shower
or Hotelguest(0).Room
```

(c) Liveness: ($\neg\surd$)
```
E[] exists (i:id_t) not Hotelguest(i).Room
```

(d) Mutex Requirement: ($\surd$)
```
A[] forall (i:id_t) forall(j:id_t)
Hotelguest(i).Shower && Hotelguest(j).Shower imply i==j
```

(e) Bounded Liveness: ($\surd$)
```
A[] (Hotelguest(1).Shower imply Hotelguest(1).time<=15)
```

(f) Fairness: ($\surd$)
```
E<> exists (i:id_t) exists (j:id_t)
Hotelguest(i).Corridor and Hotelguest(i).time>4 and Hotelguest(j).Shower
and Hotelguest(j).time<2
```

## Aufgabe 15: Das Druzba MUTEX Problem MIT Fairness

**Uppaal Modell** Das Modell aus Aufgabe 13 wird nun um die beiden Funktionen `GetQueued()` und `RemoveFromQueue()` erweitert. Die Implementierung der Hildsfunktionen sieht wie folgt aus:

```
//
// *** Variable Declaration ***
//
int [0,1] Light;
clock global;
const int N=15;

typedef meta int[0,N-1] id_t;

int [-1,N] queue[N]={-1, -1, -1, -1, -1};

//
// *** Function Declaration ***
//
void GetQueued(id_t id){
    int i=0;
    while ((i&amp;lt;N) &amp;amp;&amp;amp; (queue[i]!=-1)){
        i=i+1;
    }
    queue[i]=id;
}

void RemoveFromQueue(id_t id){
    int i=0;
    while (id!=queue[i]){
        i++;
    }
    while ((i&amp;lt;N-1) &amp;amp; (queue[i]&amp;gt;-1)){
        queue[i]=queue[i+1];
        i++;
    }
    queue[N-1]=-1;
}

//
// *** System Declaration ***
//

//With Fairness
Hotelguest(const id_t id) = WithLightNSafe(id);

system Hotelguest;
```

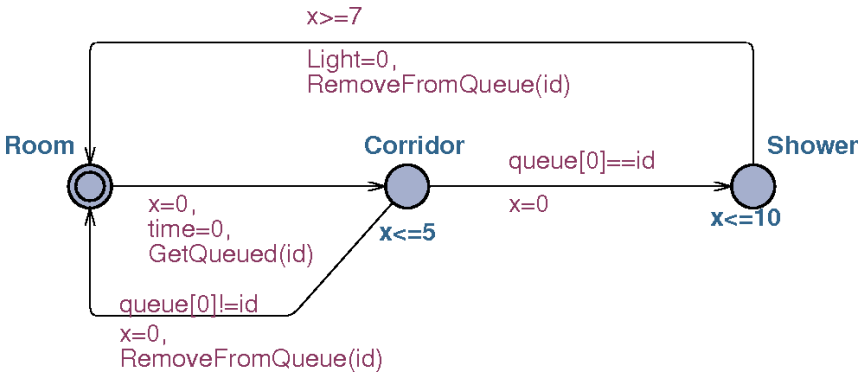


Abbildung 2: Druzba Hotel Modell mit Fairness.

**Eigenschaften** Folgende Eigenschaften sind erfüllt ($\surd$) oder nicht erfüllt ($\neg\surd$):

(a) Safety: ($\surd$)
```
A[] not deadlock
```

(b) Safety: ($\surd$)
```
Hotelguest(0).Corridor and Hotelguest(0).time>5 -> Hotelguest(0).Shower
or Hotelguest(0).Room
```

(c) Liveness: ($\neg\surd$)
```
E[] exists (i:id_t) not Hotelguest(i).Room
```

(d) Mutex Requirement: ($\surd$)
```
A[] forall (i:id_t) forall(j:id_t)
Hotelguest(i).Shower && Hotelguest(j).Shower imply i==j
```

(e) Bounded Liveness: ($\surd$)
```
A[] (Hotelguest(1).Shower imply Hotelguest(1).time<=15)
```

(f) Fairness: ($\neg\surd$)
```
E<> exists (i:id_t) exists (j:id_t)
Hotelguest(i).Corridor and Hotelguest(i).time>4 and Hotelguest(j).Shower
and Hotelguest(j).time<2
```

**Kompexität** Betrachtet man die Komplexität, so stellt man fest, dass für kleine $n$ ($n =< 5$) sich die Eigenschaften recht schnell überprüfen lassen. Für $n = 8$ wird der Zustandsraum bereits größer als $4GB$, sodass wir hier keine Aussagen treffen können. Insbesondere sollten sie sich vor Augen führen, dass durch die Quantoren $E <>$ nach einem erfüllenden Zustand gesucht wird und hierfür der gesamte Zustandsraum aufgebaut werden muß. Für $n = 7$ läßt sich Property (f) noch beweisen, die Laufzeit beträgt jedoch $> 10$ Stunden und der Zustandsraum wächst auf über $3.5GB$ an.

---

**Praktikums-Webseite:** `http://i12www.ira.uka.de/~engelc/lehre/keypraktWS0607/`
*Christian Engel*: Email: `engelc@ira.uka.de`
*Frank Werner*: Zi. 308, Tel. 608-7322, Email: `werner@ira.uka.de`