Aufgabe 1

a)

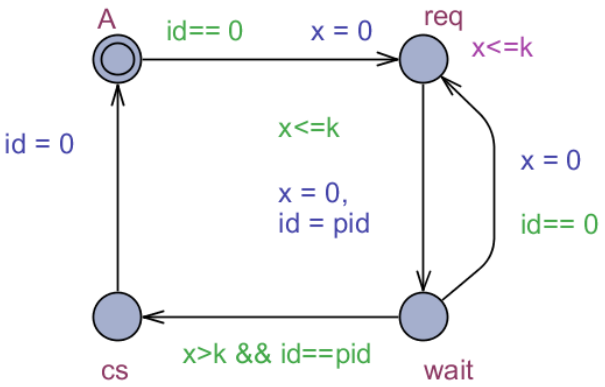

Abbildung 1: Geladenes Programm fisher.xml

b)

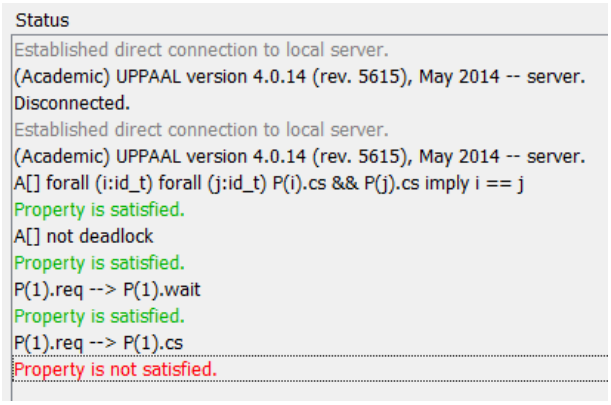

Abbildung 2: Ergebnisse der Verifizierung der Properties

c)

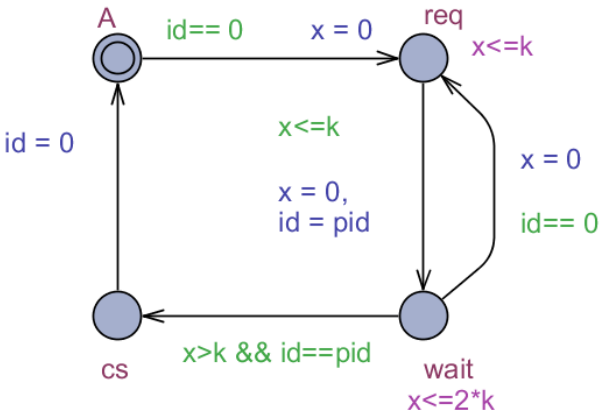

Abbildung 3: Zeitautomat mit hinzugefügter Invariante in *wait*

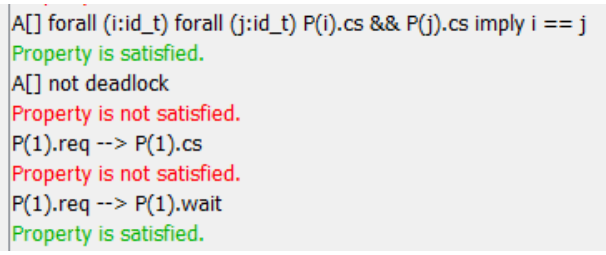

Abbildung 4: Ergebnisse der Verifizierung der Properties mit hinzugefügter Invariante in *wait*

Im Vergleich zu Teilaufgabe b) enthält das Modell durch Hinzunahme der Invariante $x \leq 2*k$ im Zustand *wait* einen Deadlock und erfüllt somit nicht die Eigenschaft $A[]$not deadlock. Der Deadlock entsteht dabei folgendermaßen:

Der erste Prozess $P(1)$ betritt zunächst den Zustand *wait*. Aufgrund des Guards $x > k$ muss Zeit verstreichen. Als nächstes spring der zweite Prozess $P(2)$ in den *wait*-Zustand. Dabei befindet sich die Clock $x \in [2, 4]$. $P(1)$ müsste nun aufgrund der neuen Invariante den Zustand verlassen, da die für den Zustand *cs* freigegebene $id == 2$ und $pid == 1$ ist, wird der Guard $id == pid$ nicht erfüllt. Weil $P(1)$ somit nicht auf einen anderen Zustand wechseln und *id* nicht auf 0 zurückgesetzt werden kann, kommt es zu einem Deadlock.

Aufgrund dessen kann ebenso die Verifizierung $P(1).\text{req} - - > P(1).\text{cs}$ nicht erfüllt werden.

d)

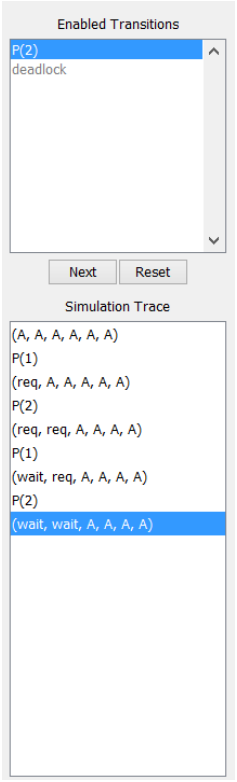

Abbildung 5: Diagnostic Trace des Deadlocks des Automaten aus Teilaufgabe 1c)

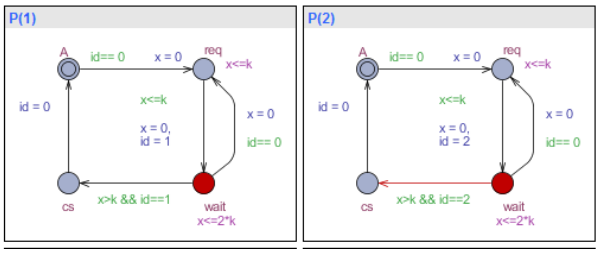

Abbildung 6: Zugehörige Deadlocks zum oben angegebenem Trace

Aufgabe 2

a)

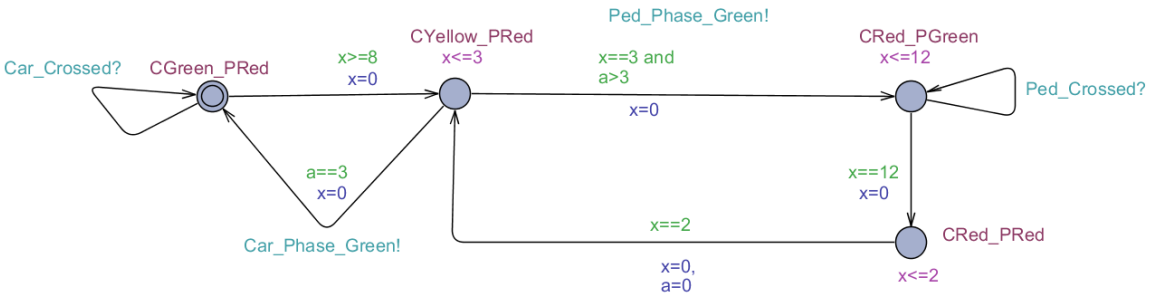

Abbildung 7: Modell *Contr* - Zum Zeitpunkt des Bildes wurden *Ped* und *Car* bereits erstellt

Patiiiii schau mal ob die Erklärungen für die Verifizierungen passen oder dass eher umgangssprachlich sein muss. Es ist spät und wir haben weder Lust noch nerven um das durchzudenken

b)

```
Contr.CRed_PGreen --> Contr.CGreen_PRed
A[] not deadlock
E<> Contr.CRed_PRed
```

Abbildung 8: Entwickelte Properties zum Modell *Contr*

```
E<> Contr.CRed_PRed
Property is satisfied.
Contr.CRed_PGreen --> Contr.CGreen_PRed
Property is satisfied.
A[] not deadlock
Property is satisfied.
```

Abbildung 9: Verifizierungsergebnisse der oben angegebenen Properties

Erklärung der Formeln für das Modell *Contr*:

i) $E\langle\rangle$ Contr.CRed_PRed
Es existiert ein Pfad auf dem der Zustand *Contr.CRed_PRed* erreicht wird.

ii) $A[]$not deadlock
Es gibt keinen Deadlock innerhalb des Modells.

iii) Contr.CRed_PGreen $\rightarrow$ Contr.CGreen_PRed
Ausgehend vom Zustand *Contr.CRed_PGreen* wird irgendwann der Zustand *Contr.CGreen_PRed* erreicht.

c)

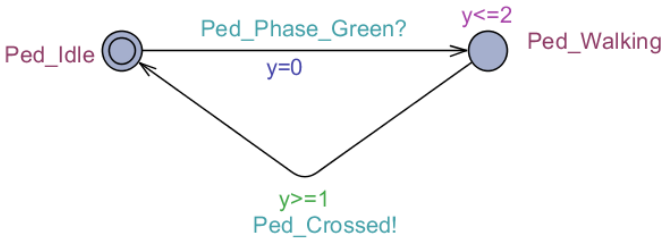

Abbildung 10: Modell *Ped*

d)

```
A[] not (Contr.CRed_PGreen&&Ped.Ped_Walking&&y>2)
A[] not(Contr.CGreen_PRed&&Ped.Ped_Walking)
```

Abbildung 11: Entwickelte Properties zum Modell *Contr||Ped*

A[] not (Contr.CRed_PGreen&&Ped.Ped_Walking&&y>2)
Property is satisfied.
A[] not(Contr.CGreen_PRed&&Ped.Ped_Walking)
Property is satisfied.

Abbildung 12: Verifizierungsergebnisse der oben angegebenen Properties

Erklärung der Formeln für das Modell $Contr||Ped$

i) $A[]$ not(Contr.CRed_PGreen&&Ped.Ped_Walking&&$y > 2$)
Auf sämtlichen Pfaden gelten nicht gleichzeitig $Contr.CRed\_PGreen$ und $Ped.Ped\_Walking$ zu einem Zeitpunkt, an dem $y > 2$ gilt.

ii) $A[]$ not(Contr.CGreen_PRed&&Ped.Ped_Walking)
Auf sämtlichen Pfaden gilt nicht gleichzeitig $Contr.CGreen\_PRed$ und $Ped.Ped\_Walking$.

e)

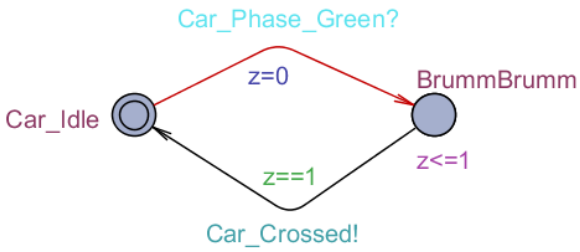

Abbildung 13: Modell Car

f)

```
A[] (Ped.Ped_Idle&&Contr.CRed_PGreen imply y>2)
A[] (Car.BrummBrumm imply Ped.Ped_Idle)
```

Abbildung 14: Entwickelte Properties zum Modell $Contr||Ped||Car$

A[] (Ped.Ped_Idle&&Contr.CRed_PGreen imply y>2)
Property is not satisfied.
A[] (Car.BrummBrumm imply Ped.Ped_Idle)
Property is satisfied.

Abbildung 15: Verifizierungsergebnisse der oben angegebenen Properties

Erklärung der Formeln für das Modell $Contr||Ped||Car$

i) $A[]$ (Ped.Ped_Idle&&Contr.CRed_PGreen imply $y > 2$)
Auf sämtlichen Pfaden implizieren die gleichzeitig geltenden Zustände $Ped.Ped\_Idle$ und $Contr.CRed\_PGreen$, dass die Clock $y > 2$ ist.

ii) $A[]$ (Car.BrummBrumm imply Ped.Ped_Idle)
Auf sämtlichen Pfaden impliziert dass bei geltendem Zustand $Car.BrummBrumm$ ebenso $Ped.Ped\_Idle$.

g)

```
A[] (Car.BrummBrumm imply not Ped.Ped_Walking)
```

Abbildung 16: Zu überprüfunde Propertie des Modells $Contr||Ped||Car$

```
A[] (Car.BrummBrumm imply not Ped.Ped_Walking)
Property is satisfied.
```

Abbildung 17: Verifizierungsergebnis der oben angegebenen Propertie

i) $A\,[]$ (Car.BrummBrumm imply not Ped.Ped_Walking)
Auf sämtlichen Pfaden impliziert der geltende Zustand $Car.BrummBrumm$ dass $Ped.Ped\_Walking$ ungültig ist.

h)

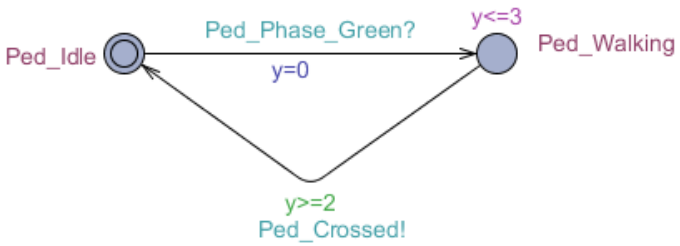

Abbildung 18: Modell Ped mit langsameren Fußgänger

```
A[] (Car.BrummBrumm imply not Ped.Ped_Walking)
Property is satisfied.
```

Abbildung 19: Verifizierungsergebnis der Propertie aus g)

i)

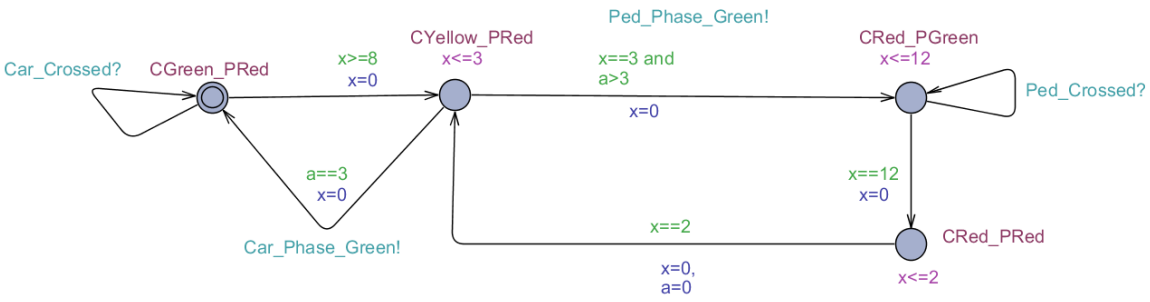

Abbildung 20: Modell $Contr$ mit veränderten Time Constraints zur Verkürkung der Grünphase der Autos

```
A[] (Car.BrummBrumm imply not Ped.Ped_Walking)
Property is not satisfied.
```

Abbildung 21: Verifizierungsergebnis der Propertie aus g) unter Verwendung der oben angegebenen alternativen Version von $Contr$

Indem die Dauer der Rot/Grün, Rot/Rot und der Gelb/Rot Phasen auf insgesamt eine Sekunden verkürzt wurden, ist es möglich, dass die Grünphase der Autos nach einer Sekunde eintritt. Da der Fußgänger allerdings zwei bis drei Sekunden zum Überqueren der Straße benötigt, überschneiden sich die Übergangszeiten, wodurch das Modell nicht mehr sicher ist.

Timing Constraints nochmal explizit angeben ?

j)

Zwischen zwei grünen Amplephasen für Fußgäger liegen zwei mal die “Dead Phase“, zwei mal die “Yellow Phase“ und mindestens acht Sekunden grüne Ampelphase für die Autos, was insgesamt $2*2+2*3+8 = 18$ Sekunden ergibt, in dieser Zeit können nicht mehr als vier Fußgänger auftauchen, wenn sie alle fünf Sekunden auftauchen.