15. Sprachen zur Verhaltensspezifikation: Verhaltensmodellprüfung (Behavioral Model Checking)

1) Modellprüfung – Überblick
2) Realzeit-Modellprüfung

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de
Version 11-1.2, 04.01.12

Obligatorische Literatur

- Markus Müller-Olm, David Schmidt, Bernhard Steffen. Model-Checking. A Tutorial Introduction. Springer LNCS, Volume 1694, 1999, p 848ff
  - http://www.springerlink.com/content/l437dulbgk67jl6m/
- [BW04] Timed Automata: Semantics, Algorithms and Tools, Johan Bengtsson and Wang Yi. In Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004
  - http://www.it.uu.se/research/group/darts/papers/texts/by-lncs04.ps
- [BDL04] A Tutorial on Uppaal, Gerd Behrmann, Alexandre David, and Kim G. Larsen. In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04). LNCS 3185.
  - http://www.cs.auc.dk/~adavid/publications/21-tutorial.pdf

Andere Literatur

- E. Clarke's Kurs über Model Checking
  http://www.cs.cmu.edu/~emc/15817-s05/
- E. Clarke. Model Checking. Springer LNCS 1346, 1997
  http://www.springerlink.com/index/v1h70v370p172844.pdf
- E. M. Clarke, E. A. Emerson, A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Transactions on Programming Languages and Systems, vol 8, number 2, pages 244—263, apr 1986,
  - An early version appeared in Proc. 10th ACM Symposium on Principles of Programming Languages, 1983
  - http://www.acm.org/pubs/toc/Abstracts/0164-0925/5399.html

15.1 Modellprüfung - Überblick

Problem

- Wie kann man Aussagen über Verhalten (spezifiziert durch Zustandssysteme) machen?
  - Verifikationen
  - Konservative Abschätzungen wie Typprüfungen
- Jenseits von der Ausstattung mit Typen (Metamodelle)
  - Mit Logik?

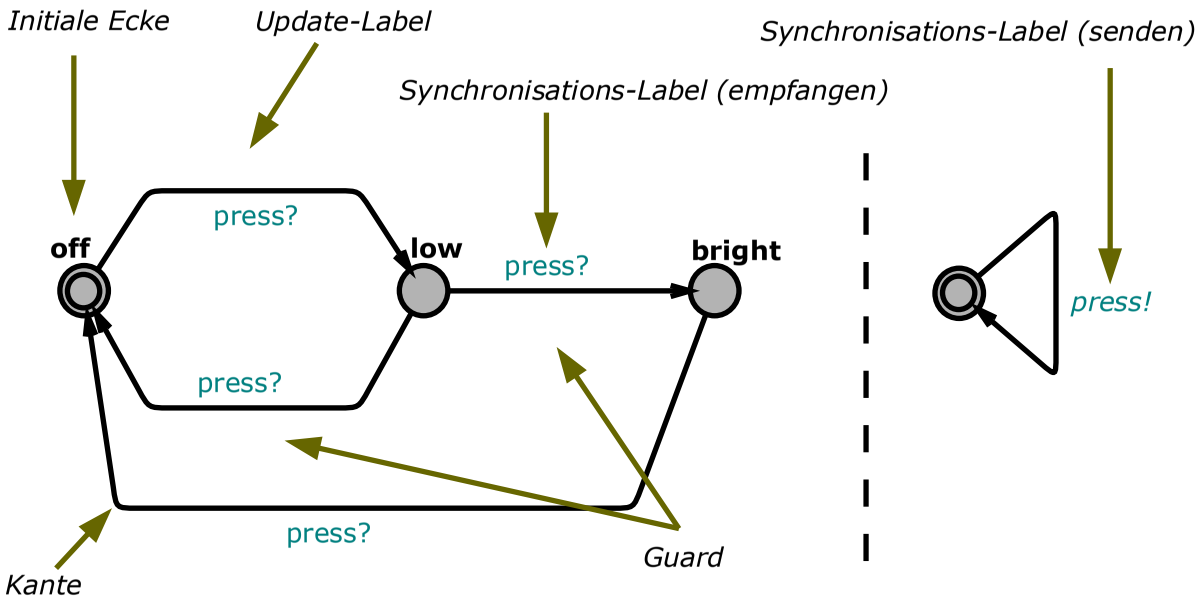
- Lösung 1: Spezifiziere alles, auch das Zustandssystem, in Logik und lasse eine Deduktionsmaschine laufen
- Lösung 2: mit dem Graph-Logik-Isomorphismus:
  - Spezifikation des Zustandssystems als endlicher Graph (statt mit Logik)
  - Dann ergibt sich der dynamische Zustandsraum als unendlicher Baum von Zuständen
  - Auswertung von Logik-Formeln durch “Weiterhangeln” im Zustandsraum

Behavior Model Checking (Prüfung des Semantischen Modells)

- Voraussetzung:
  - Modellierung eines Zustandssystems als Graph
    - z. B. Automaten, Petri-Netze
    - haupsächlich nebenläufige Systeme
  - Beschreiben von Anforderungen an das System und an seine Zustände mit Logik
    - z. B. mit temporaler Logik (Pfadbasierter Logik)
- **Verifikation der Anforderungen**
  - durch Berechnung aller möglichen Zustände des Systems sowie aller in diesen gültigen Prädikate

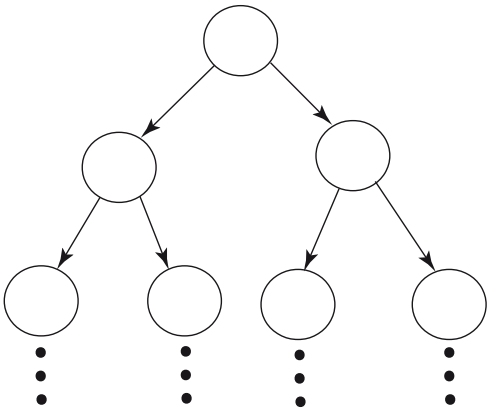
Automat für Doppelklick-Lichtschalter

- Können wir etwas über diese beiden kommunizierenden Automaten beweisen?



Model-Checking

- Vom Gesamtsystem wird **Berechnungsbaum** (oder Zustandsraum, reachability tree) erstellt
- Der Berechnungsbaum ergibt sich durch Interpretation der Automaten
  - Er enthält alle möglichen Zustände des Systems
  - Wurzel ist der eindeutige Startzustand des Systems
- Anfragen an den Berechnungsbaum (Queries): Für welche erreichbaren Zustände gilt welches Prädikat?

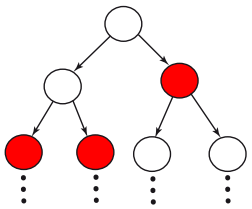


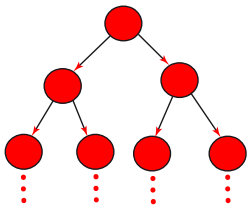
Operatoren

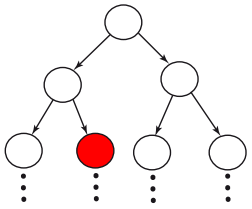
- Seien φ, ψ aussagenlogische Zustandsformeln
  - Seiteneffekt-freier Bedingung, nach *wahr* oder *falsch* auswertbar
  - Beispiele:
    - `x > 1`
    - `(P1.Ecke3 and (P2.i == 5) or (clock > 200)`
    - `not deadlock`
    -
- **Pfadquantoren**: Auf welchen Pfaden erreichbarer Zustände gilt die Aussage?
  - **A** auf allen Pfaden (all paths)
  - **E** auf mindestens einem Pfad (some paths)
- **Temporaloperatoren**: Wann gilt die Aussage im Pfad?
  - **F** irgendwo entlang des Pfades (*future, finally*)
  - **G** in allen Zuständen entlang des Pfades (*globally*)

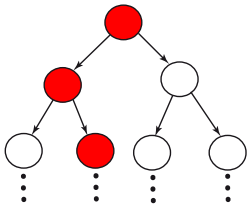
Operatoren (2)

- Pfad-Temporalquantor-Operatoren über Zustandsformeln:
  - **AF** φ auf allen Pfaden irgendwann gilt einmal (all-exists: on all paths exists)
  - **AG** φ auf allen Pfaden in allen erreichbaren Zustanden (all-globally: on all paths in all states)
  - **EF** φ irgendwo entlang eines Pfades irgendwann (exists-finally; future: on a path finally exists)
  - **EG** φ irgendwo in einem Pfad in allen Zuständen entlang des Pfades (exists-globally)

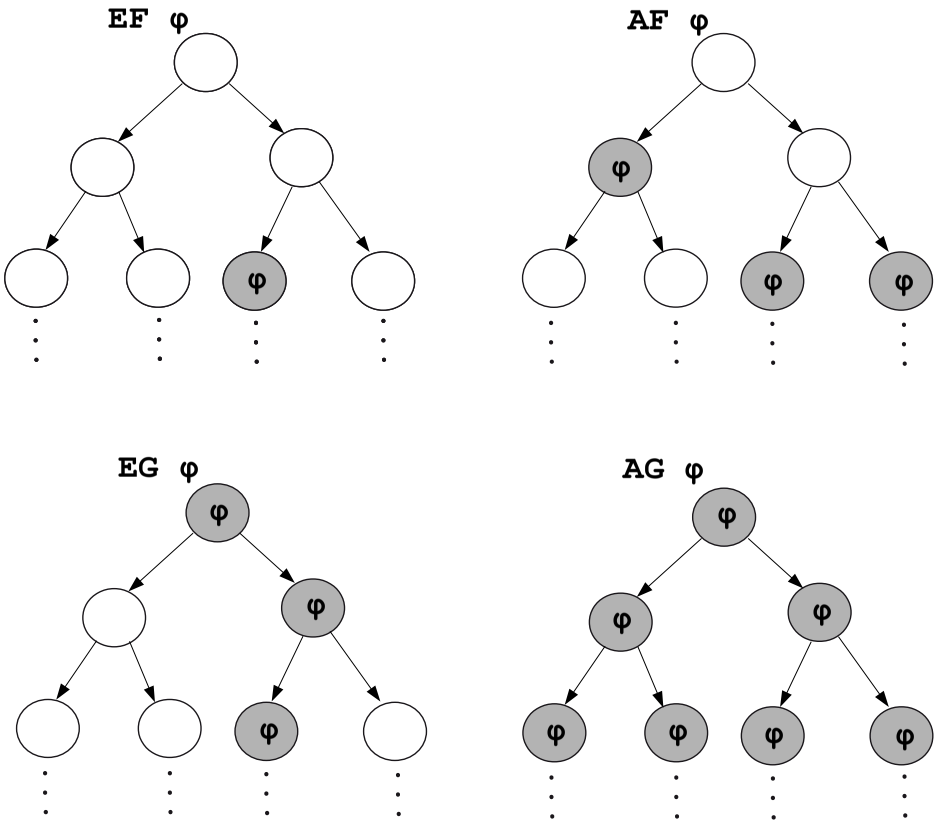






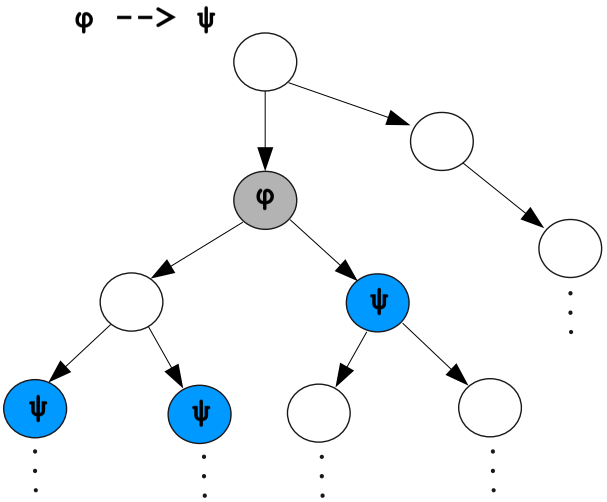


Queries: Beispiele



Operatoren (2)

- Leads-To Operator φ --> ψ
  - Gilt φ in einem Zustand, so wird in der Zukunft irgendwann ψ gelten



Queries: konkrete Beispiele für den Doppelklick-Lichtschalter

- **EF** (Switch.bright)
  - Ecke *bright* im Prozess Schalter kann erreicht werden
  - ist erfüllt
- **AG** (not deadlock)
  - Jeder Zustand im Zustandsraum hat einen Nachfolger
  - Ist erfüllt
- Switch.low **-->** (Switch.off || Switch.bright)
  - Wird Switch.low erreicht, gilt auf jedem Pfad irgendwann (Switch.off || Switch.bright)
  - Ecke *low* wird in jedem Fall verlassen
  - Prädikat ist nicht erfüllt, da User nicht zwangsläufig „drückt“

15.2. Realzeit-Modellprüfung (Real-Time Model Checking)

.. mit UPPAAL..

Achtung: Folien mit 15.1, Standard-Model-Checking, vergleichen, um die Unterschiede zu sehen

Real-time Model-Checking

- Modellierung eines Systems in formaler Sprache
  - z. B. Automaten, Petri-Netze
- Hier: Beschreiben von Anforderungen an das System mit temporaler Logik
- **zusätzlich: Bedingungen über "real-time variables" (Realzeitbedingungen, Realzeitaspekt)**

- **Verifikation der Anforderungen**
  - durch Berechnung aller möglichen Zustände des Systems **und Überprüfung der Realzeitbedingungen**

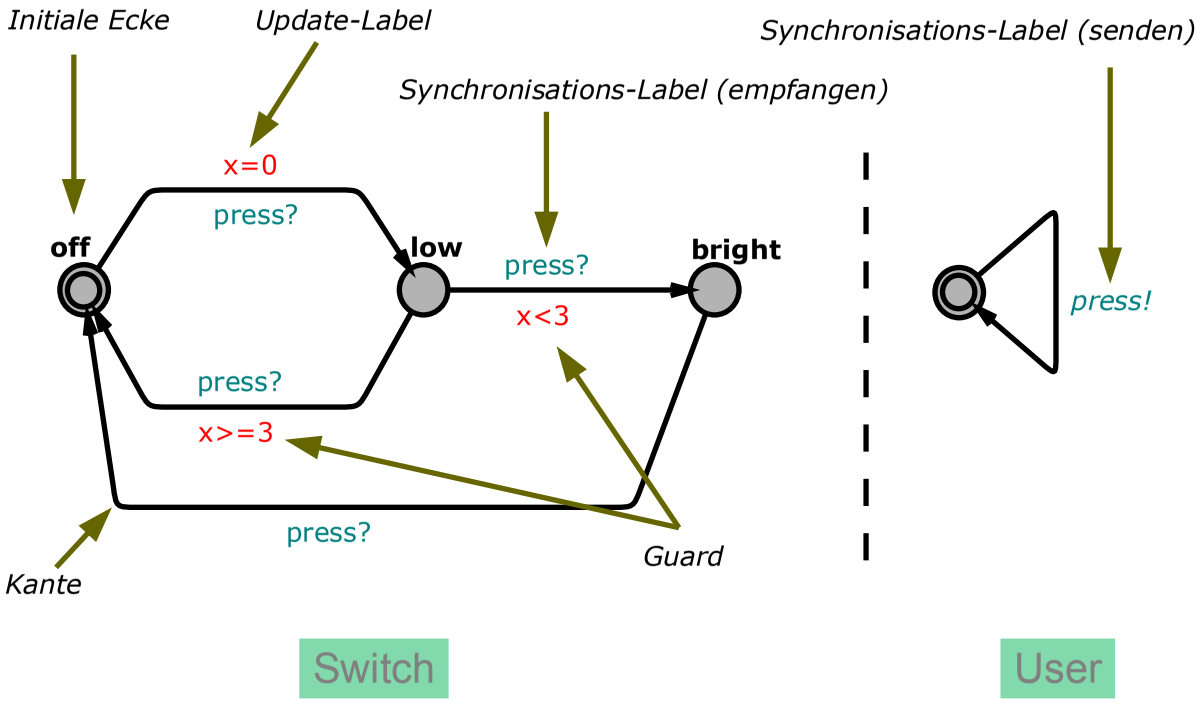
UPPAAL

http://www.uppaal.org/

- Model-Checking-Tool für Echtzeitsysteme
  - Verifikationsengine in C++
- graphische Benutzeroberfläche in Java
- Anforderungen werden mittels einer Untermenge von CTL (computation tree logic) formuliert
- UPPAAL – Modellierungssprache:
  - parallel laufende erweiterte **Timed Automata (TA)**: Endliche Automaten mit Uhren
  - Uhren haben als Wertebereich reelle Zahlen
  - alle Uhren schalten synchron
  - Uhren können zurückgesetzt werden
- Systemzustand: Zustand aller TA, Uhren, Variablen
- Erweiterungen:
  - diskrete Variablen (int, boolean; strukturierte Datentypen, Arrays)
  - Konstanten
  - C-ähnliche Funktionen
  - Synchronisation der Automaten über *Channels*

Timed Automata in UPPAAL – Beispiel Doppelklicklichtschalter

- Achtung: Jetzt mit Realzeit-Variablen, -Guards (Bedingungen)



Timed Automata in UPPAAL – Zustände

- Zustände (Ecken, Knoten)
  - Name
  - Invariante: Bedingung die innerhalb des Zustandes immer gelten muss
- Invariante kann normal, *dringend (urgent)* oder *verpflichtend (committed)* sein
- **urgent**: Zustand muss ohne Zeitverzögerung wieder verlassen werden
  - Äquivalent: Hinzufügen einer Uhr x, die auf allen eingehenden Kante auf 0 gesetzt wird, und Invariante x≤0
- **committed**: sobald Zustand verpflichtet, muss der nächste Zustandsübergang Ausgangskante einer verpflichtenden Ecke sein

Timed Automata in UPPAAL – Kanten

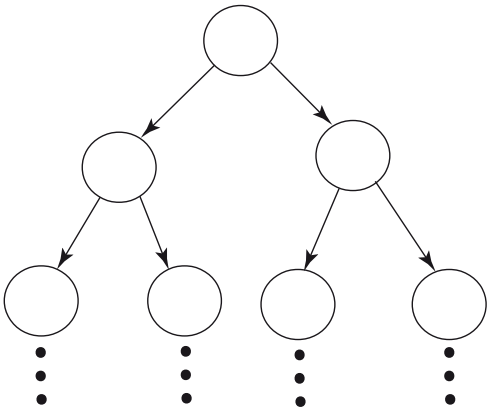
- Transitionen (Kanten)
- *Selections:* Zuweisungen die in Guards und Updates verwendet werden können
- *Guards:* Bedingungen unter denen die Kante schalten kann. (z. B. Zeitbedingungen).
- *Updates:* Wertzuweisungen an Uhren und anderen Variablen.
- *Synchronisations* über Channels: eine *c!* beschriftete Kante schaltet synchron mit *c?* beschrifteter Kante, wenn
  - beide aktiv sind (d. h. Guards erfüllt)
  - und *c* ein Channel ist

UPPAAL – Aufbau

- Automaten werden als Schablonen (*Templates*) modelliert
  - diese werden instanziiert, d.h. parametrisiert werden
  - (Templates sind Fragmentkomponenten, siehe CBSE)
- Deklarationsteil (textuell)
  - globale und lokale Deklarationen und Definitionen
  - Deklaration von Variablen und Channels
  - Definition von Konstanten und Funktionen
- Systemdeklarationen:
  - parametrisieren Templates
- graphischer Teil
  - nutzt Variable etc. aus dem Deklarationsteil

UPPAAL – Real-time Model-Checking ähnlich zum Model-Checking

- Vom Gesamtsystem wird Berechnungsbaum (oder *Zustandsraum*) erstellt
- Enthält alle möglichen Zustände des Systems
- Wurzel ist der eindeutige Startzustand des Systems
- Anfragen an den Berechnungsbaum (Queries): Für welche erreichbaren Zustände gilt welches **Prädikat über Realzeitvariablen**?



UPPAAL – Query Syntax

E<> φ, **E[]** φ, **A<>** φ, **A[]** φ, φ **-->** ψ

wobei φ, ψ Zustandsformeln über Realzeitvariablen

Seiteneffekt freier Ausdruck, nach *wahr* oder *falsch* auswertbar

```
x > 1
(P1.Ecke3 and (P2.i == 5) or (clock > 200)
not deadlock
```

Pfadquantoren: Auf welchen Pfaden gilt die Aussage?

A auf allen Pfaden (all)

E auf mindestens einem Pfad (some)

Temporaloperatoren: Wann gilt die Aussage?

<> (F) irgendwo entlang des Pfades (*future*)

[] (G) in allen Zuständen entlang des Pfades (*globally*)

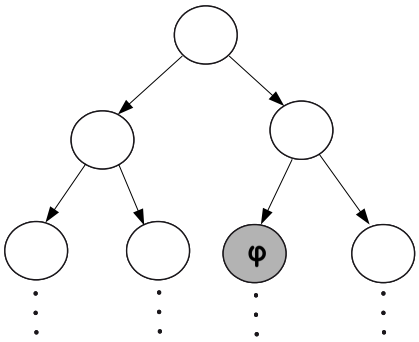
Leads-To Operator φ **-->** ψ

Gilt φ, so wird in der Zukunft irgendwann ψ gelten

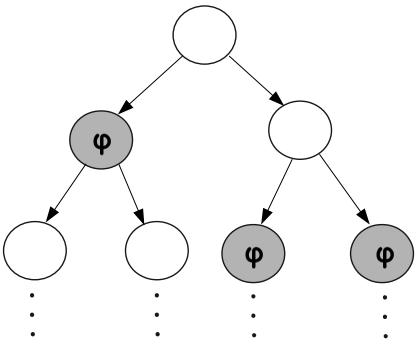
Kein verschachtelten Pfadquantoren (z. B.: **A[]** (**E<>** φ) *)*

UPPAAL – Queries: Beispiele (vgl. Standard Model Checking)

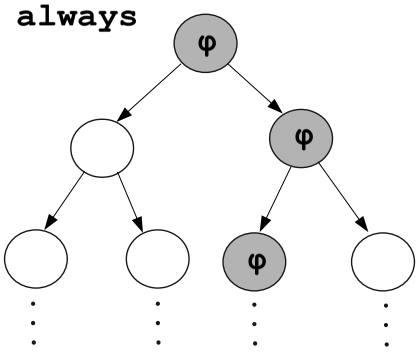
`E<> φ (EF φ)`
`Exists path with real-time`
`Condition eventually`



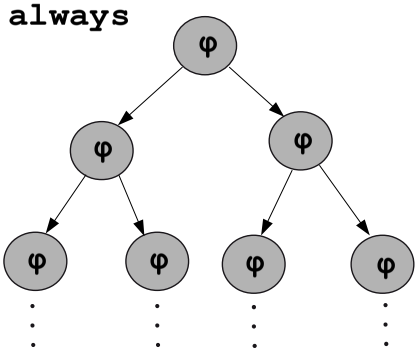
`A<> φ (AF φ)`
`All paths: with real-time`
`condition eventually`



`E[] φ (EG φ)`
`Exists path with real-time`
`Condition always`



`A[] φ (AG φ)`
`On all paths: real-time condition`
`always`



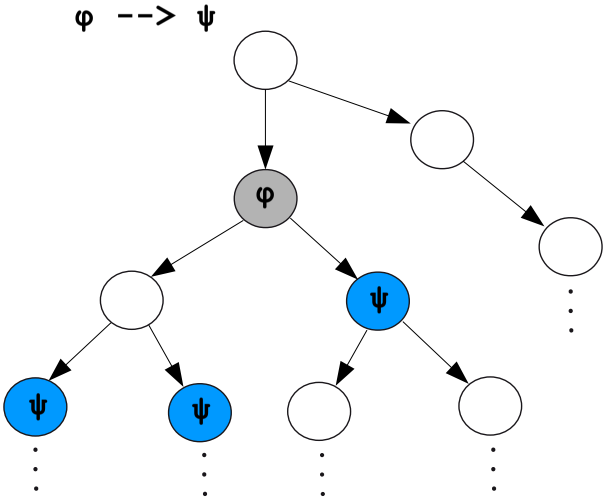
UPPAAL – Queries: Beispiele (2)

Leads-To Operator φ **-->** ψ

Gilt φ in einem Zustand, so wird in der Zukunft irgendwann ψ gelten

Beispiel: From state with φ, on all paths: holds ψ eventually

`φ --> ψ`



UPPAAL – Queries: Realzeit-Aussagen über Doppelklickautomat

E<> Switch.bright

Ecke *bright* im Prozess Schalter kann erreicht werden

ist erfüllt

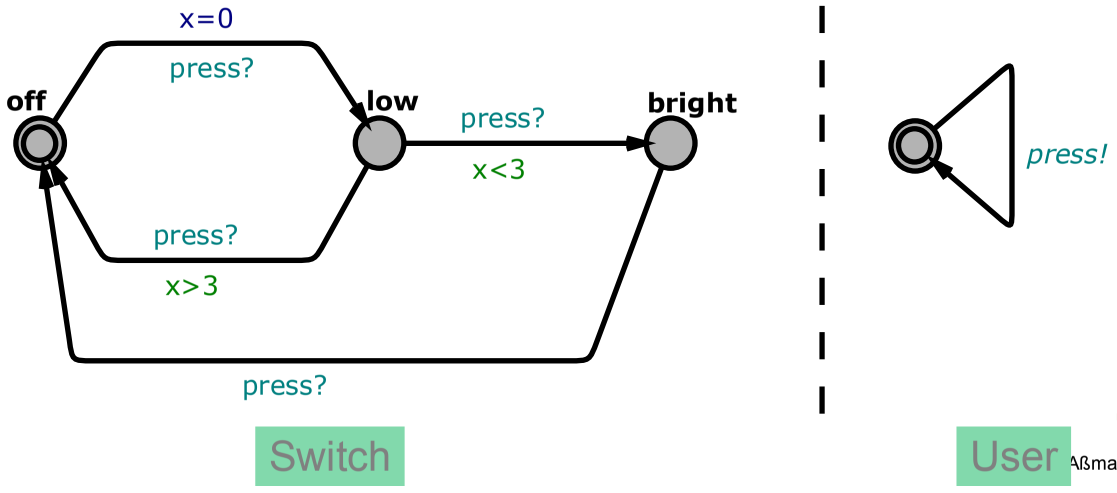
Switch.low **-->** (Switch.off || Switch.bright)

Ecke *low* wird in jedem Fall verlassen

nicht erfüllt, da User nicht zwangsläufig „drückt“

A[] not deadlock

Jeder Zustand hat einen Nachfolger



End

Slides courtesy to Martin Rothmann