Formale Methoden für eingebettete Systeme

19. Juli 2005

Vorwort

Dies ist eine Mitschrift zur Vorlesung von Prof. Dr.-Ing. Kowalewski im Sommersemester 2005. Ich weise ausdrücklich darauf hin, daß es sich hierbei um kein offizielles Skript handelt und somit auch keine Garantie auf Vollständigkeit und Richtigkeit des Textes gegeben werden kann.

Bei Fehlern, Verbesserungsvorschlägen, Kommentaren, etc. wäre ich dankbar für eine kurze eMail an `Matthias.Lebok@rwth-aachen.de`

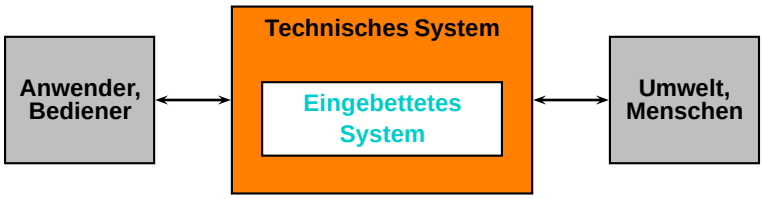
Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Formale Methoden für eingebettete Systeme

1.1.1 Eingebettete Systeme



Produktions-Automatisierung
z.B.
Leitsysteme in
- Fertigungstechnik
- Verfahrenstechnik

Produkt-Automatisierung
z.B.
- Fahrzeugelektronik,
- Avionik,
- Medizintechnik

1.1.2 Formale Methoden

Formale Methoden = Auf mathematisch/logisch strengen Darstellungsmitteln und Verfahren beruhende Vorgehensweisen zur Beschreibung, Analyse und Realisierung von Entwürfen.

Beispiele für formale Methoden:
- Spezifikation
- Verifikation
- Synthese

Keine formalen Darstellungen:
z.B.:
- Programmiersprachen
- Simulationsmodelle

Im Sprachgebrauch der Informatik:
formale "Semantik" **vs. ausführbare "Semantik"**

1.1.2.1 Formale vs. ausführbare Semantik

Beispiele für formale Modelle:

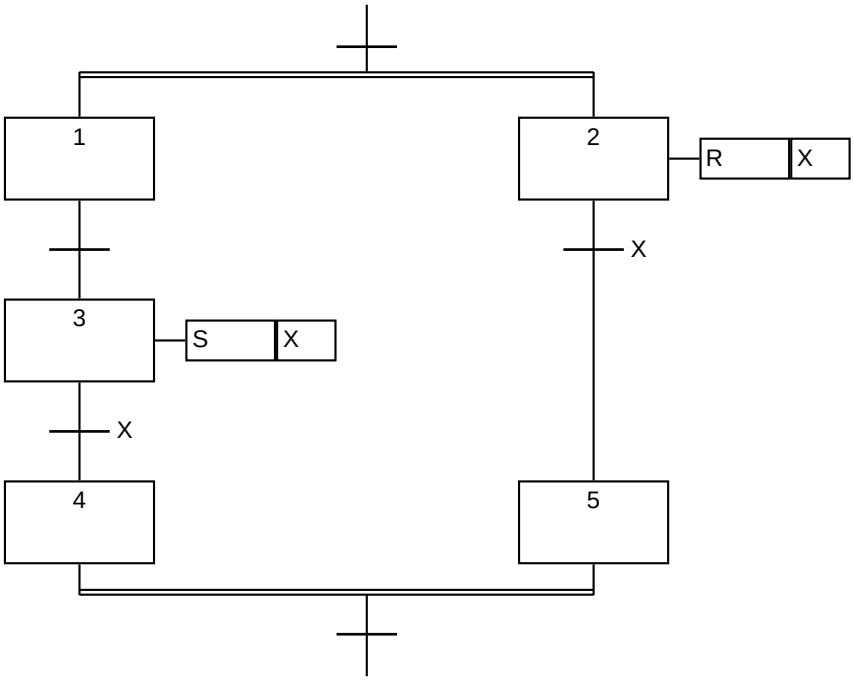
- Zustandsübergangssysteme

- Petrinetze
- Logiken
- Gleichungen, DGLn

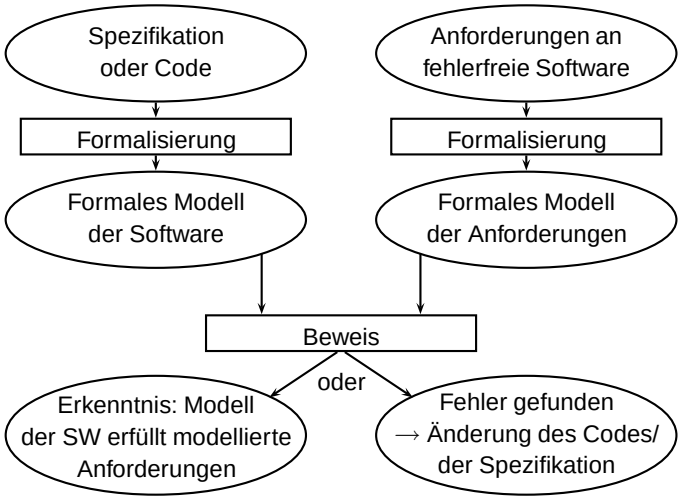
Zwei Kennzeichen:

- Formale **Syntax**
- Vollständige, eindeutige und widerspruchsfreie **Semantik**

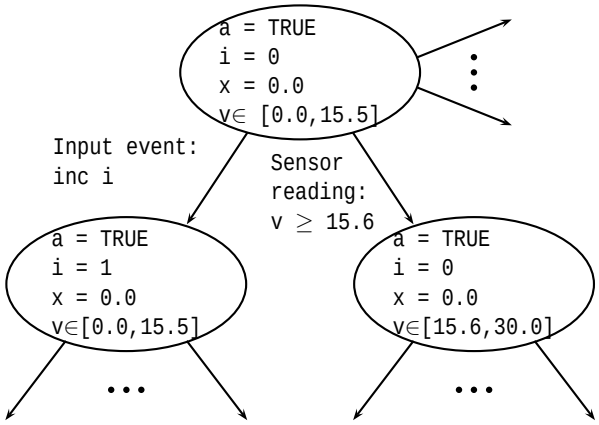
Nur ausführbar, nicht formal:
Bsp.: SFC



Formale Verifikation (Beispiel Software)



Formales Modell der Software – Beispiel



Semantik?

Formales Modell der Anforderungen

Generische Anforderungen:
z.B.: Software verhält sich deterministisch.

Spezifische Anforderungen:
z.B.:

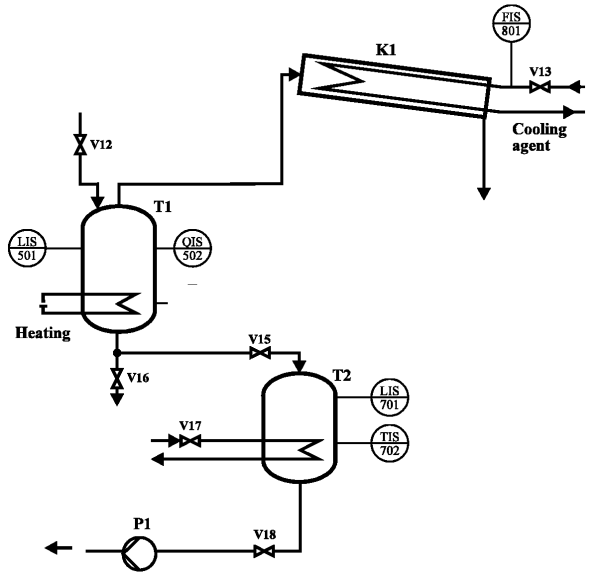
Nicht formal:	`i` liegt immer zwischen 0 und 99, bis zu einem bestimmten Ereignis, ab dem es 100 bleibt. In jedem Fall wird es nicht negativ.
Formal:	$(0 \leq \texttt{i} \leq 99) \cup (\texttt{i} = 100) \wedge \Box \texttt{i} < 0$
Oder:	Zustand z ist nicht erreichbar. (Bsp.: Die Steuerung versucht nie die Pumpe zu starten, wenn das dahinterliegende Ventil geschlossen ist.)

1.1.3 Zwei Beispiele für Anwendungen

- Verfahrenstechnik

- Automobilelektronik

Beispiel 1: Verfahrenstechnik[1]



Vorgesehener Produktionsablauf:

- Füllen von Behälter T1
- Aufheizen
- Verdampfen bis gewünschte Konzentration erreicht ist
- Heizung ausschalten und T1 leeren
- Nachbearbeitungsschritt in T2

Notabschaltung bei Kühlungsausfall

Randbedingungen:

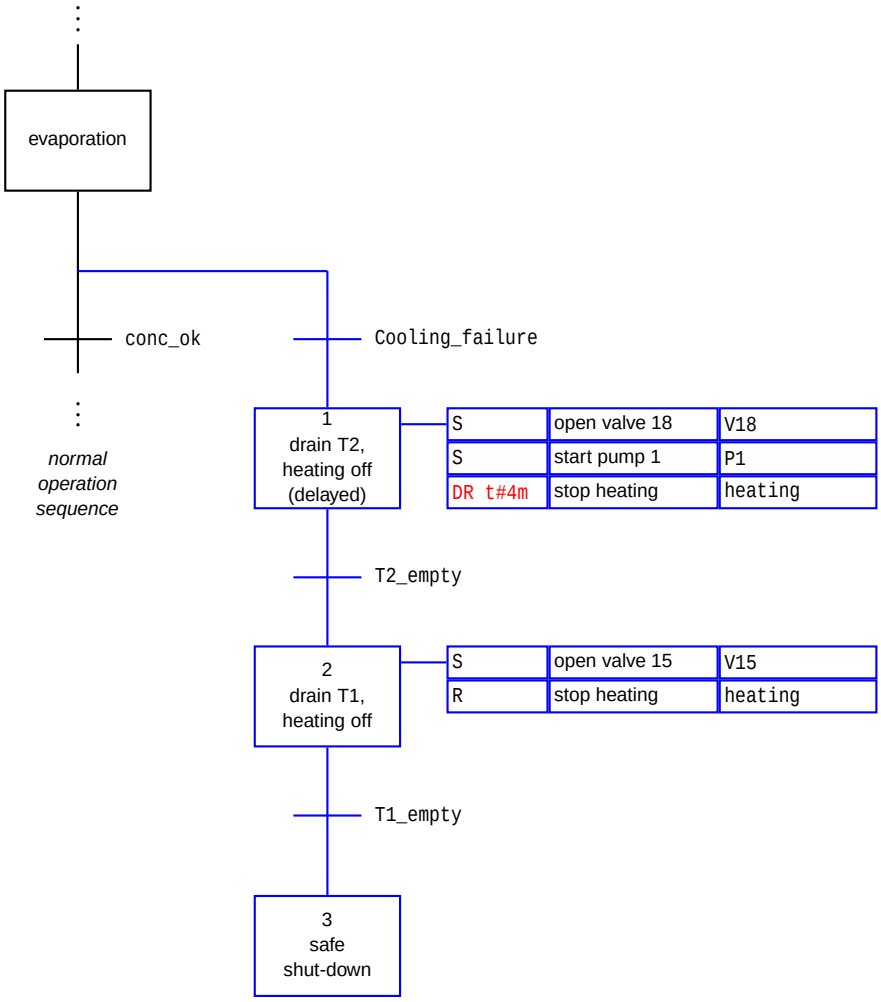
1. Bei fortgesetzter Wärmezufuhr steigen Temperatur und Druck (geschlossenes System).
2. Bei Abkühlen unter einen Grenzwert beginnt das Material im Verdampfer zu kristallisieren.

→ **Gegenläufige Anforderungen:**

- Heizung muß **früh genug** abgeschaltet werden, um gefährlichen Druckanstieg zu vermeiden.
- Heizung muß **lange genug** eingeschaltet bleiben, um Kristallisierung zu vermeiden.

[1] Kowalewsk et al., Europ. Journals of Control, Special Issue on Hybrid Systems, 2001

Vorschlag für Steuerungsprogramm[2]

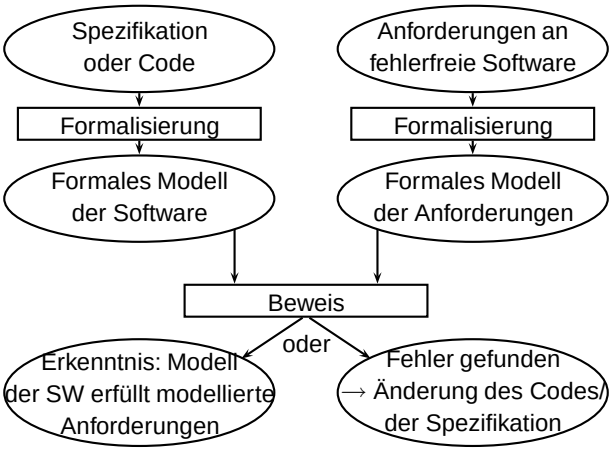


Problemstellung

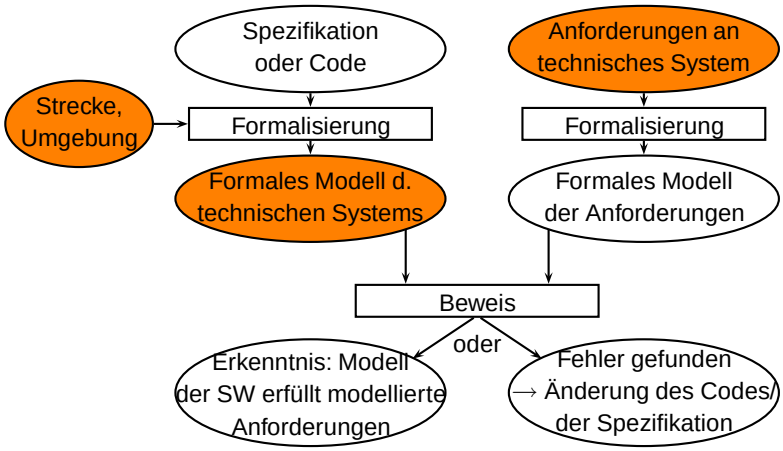
- Erüllt das Steuerungsprogramm die folgenden Anforderungen?
  - Der Druck darf oberen Grenzwert nicht überschreiten.
  - Die Temperatur darf Kristallisationspunkt nicht unterschreiten.
- Zu überprüfen ist:
  - logischer Entwurf des SFCs
  - Wahl des Wertes für den Parameter Wartezeit (`t#4m`)
- Anforderungen sind **für den gesteuerten Prozess** (“geschlossenen Kreis”, “technisches System”) und nicht für das Steuerungsprogramm formuliert.

[2] Sequential Function Chart (SFC) nach IEC 1131-3

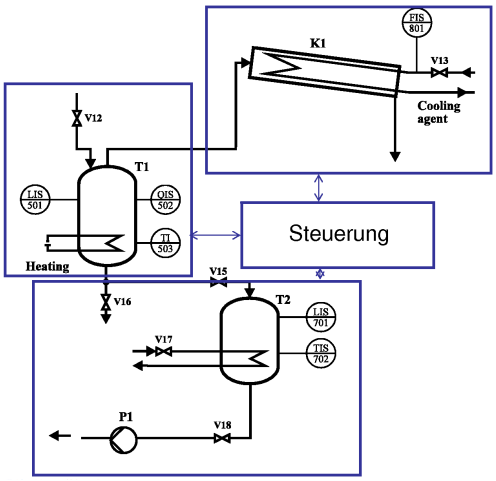
Formale Verifikation von Automatisierungssystemen



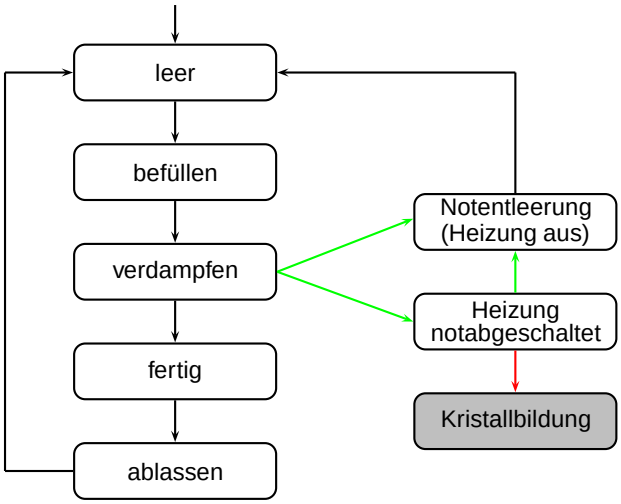
Formale Verifikation von eingebetteten Systemen



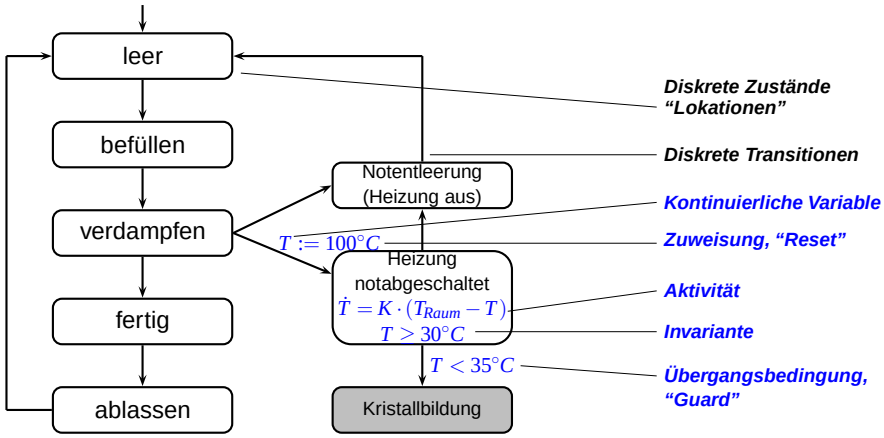
Formalisierung



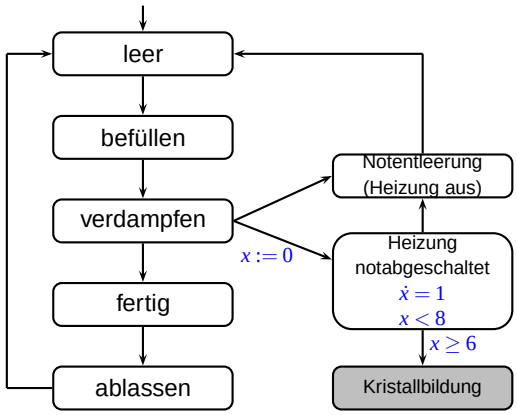
Diskretes Modell für Tank 1



Hybrider Automat[3]



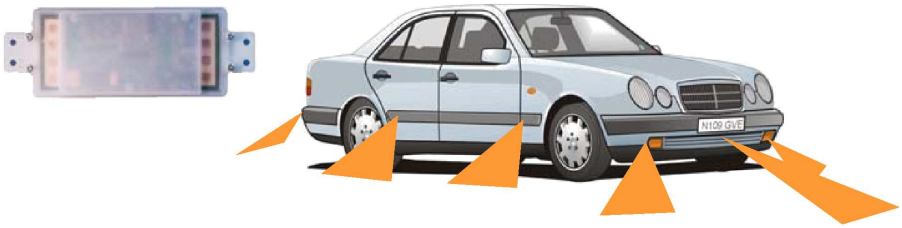
Echtzeitautomat[4]



[3] Henzinger, Alur, Sifakis, 1992
[4] Alur, Dill, 1990

Beispiel 2: Fahrzeug-Umfeld-Sensorik:

Nahbereichsradar-Technologie (24 GHz):



Anwendungen:

- Einparkhilfe
- **Automatisches Einparken**
- ACC Stop & Go, Staufolgefahren
- Pre-Crash-Detection
- **Kollisionsvermeidung**
- Toter-Winkel-Überwachung
- **Abbiege- und Spurwechselassistent**

Problemstellung

Bisher:

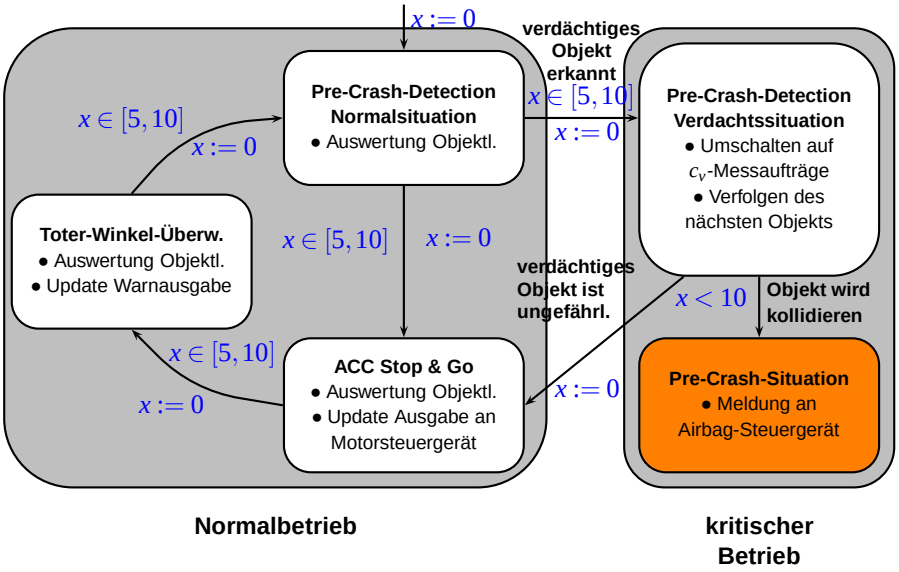
- Anwendungen unabhängig voneinander entwickelt und realisiert.
- Jede Anwendung: eigene Sensoren, Steuergerät, SW

Jetzt:

- Bisheriger Ansatz wegen Kosten- und Bauraum nicht mehr möglich.
- Einführung neuer Anwendungen erfordert gemeinsame Nutzung von Sensoren und Integration von Diensten auf wenigen Steuergeräten.
- Scheduling für Ressourcenzuteilung notwendig

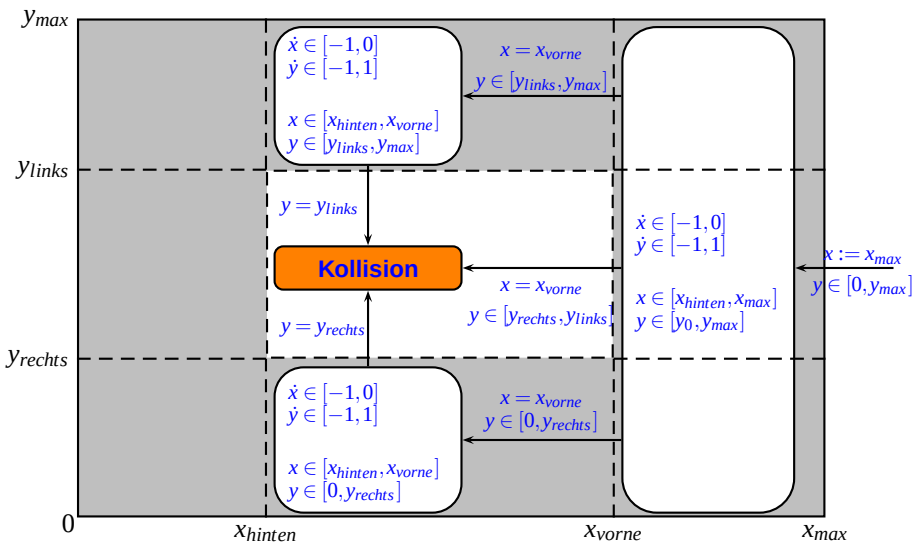
Problem: Garantiert das Scheduling, daß in jeder Situation die Echtzeitanforderungen der einzelnen Anwendungen eingehalten werden?

Echzeitautomat für Ressourcen-Zuteilung

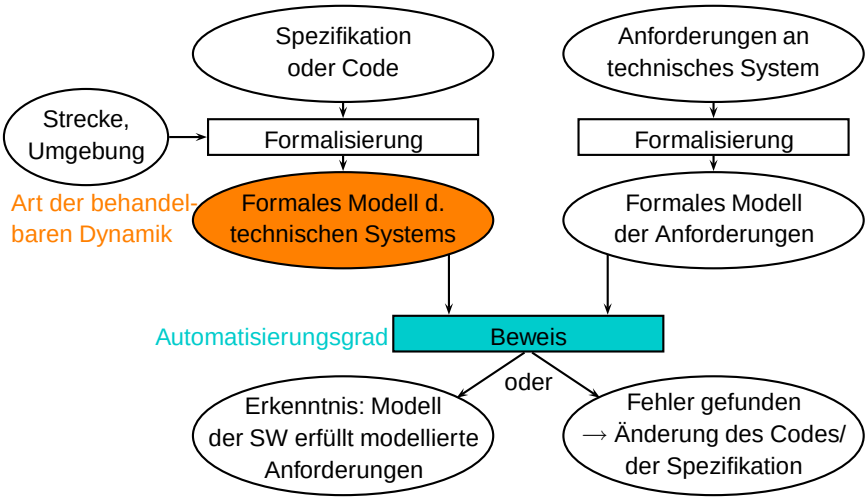


Modell des Auto-Umfeldes als hybrider Automat

Beispiel: ein Objekt von vorne



1.2 Stand der Technik

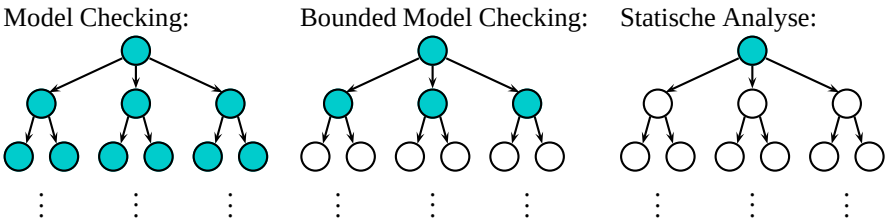


Automatisierungsgrad

Deduktive Analyse (Theorem Proving)

Algorithmische Analyse

- Vollständige Suche im Zustandsraum (Modelchecking)
- Begrenzte Suche im Zustandsraum (Bounded Modelchecking)
- Statische Analyse (Abstrakte Interpretation)



Prinzip der abstrakten Interpretation

```
...
FOR i=1..100 DO
  ...
  a=b/(i-90)
  ...
END_FOR
...
```

Kein Durchlaufen der Schleife, sondern Rechnen mit $i \in [1, 100]$.

Art der behandelbaren Dynamik

Diskret:

- Kommerzielle Werkzeuge
- Behandlung von Zeitanforderungen durch “Zählen von Zustandsübergängen”

Diskret + Zeit:

- Modelchecking: belastbare akademische Werkzeuge (z.B. Uppaal)
- Konservative Abschätzung von Ausführungszeiten: kommerzielle Werkzeuge verfügbar (AbsInt, Saarbrücken)

Hybrid:

- Forschungsthema, akademische Werkzeuge nicht belastbar
- Lösungsansatz: Abstraktion durch einfachere Dynamik, gegebenenfalls Konkretisierung, wo nötig

Kommerzielle Werkzeuge und industrielle Akzeptanz

Model Checker:

- *SCADE* – Esterel Technologies Avionik
- *SILDEX* – TNI Valiosys Avionik
- *OSC-Verifier* – OSC Embedded Systems, Oldenburg Automobil: v.a. **Testfallgenerierung**
- *Autofocus* – Validas (TU München)

Abstrakte Interpretation:

- *Polyspace C-Verifier* – Polyspace Avionik, Automobil: v.a. **Code-Analyse**
- *aiT* – AbsInt, Saarbrücken Avionik, Automobil: WCET

1.3 Ausblick

Inhalt der Vorlesung:

- Formale Beschreibungsmittel (vor allem für Verhalten von Systemen)
  - deklarativ/operationell
  - diskret/mit Zeit/hybrid
- Formale Verifikation
- Formale Testunterstützung
- Formale Synthese

Kapitel 2

Grundlagen

Uns interessieren **dynamische** Eigenschaften von Systemen, keine statischen.[1]

$\Longrightarrow$ **Verhalten** eines Systems **über der Zeit** muß beschrieben und analysiert werden können.

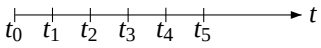
2.1 Zeitachsen

1. kontinuierlich ("dicht")

   z.B. $t \in \mathbb{R}_0^+$ (oder $t \in \mathbb{R}$)
   (Häufig wird nur positive Zeit betrachtet, $0 =$ "Urknall", aber auch Beginn bei $-\infty$ möglich.)

2. diskret

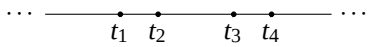
   z.B. $t \in \mathbb{N}_0$

   $$t_{n+1} - t_n = \Delta t = \text{const.}$$

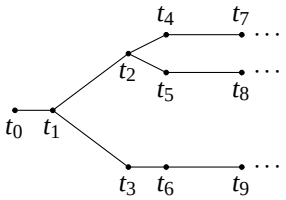
   "äquidistante Zeitschritte"

3. Ordnung von Zeitpunkten ohne Quantifizierung

   (a) Totalordnung $t_0 < t_1 < t_2 < \ldots$

   (a) Halbordnung

   (z.B. abgeleitet aus Kausalbeziehungen $\longrightarrow$ Petrinetz)

[1] **Beispiel** Bremst ein Auto, wenn man auf die Bremse tritt? Nicht: Farbe des Autos.

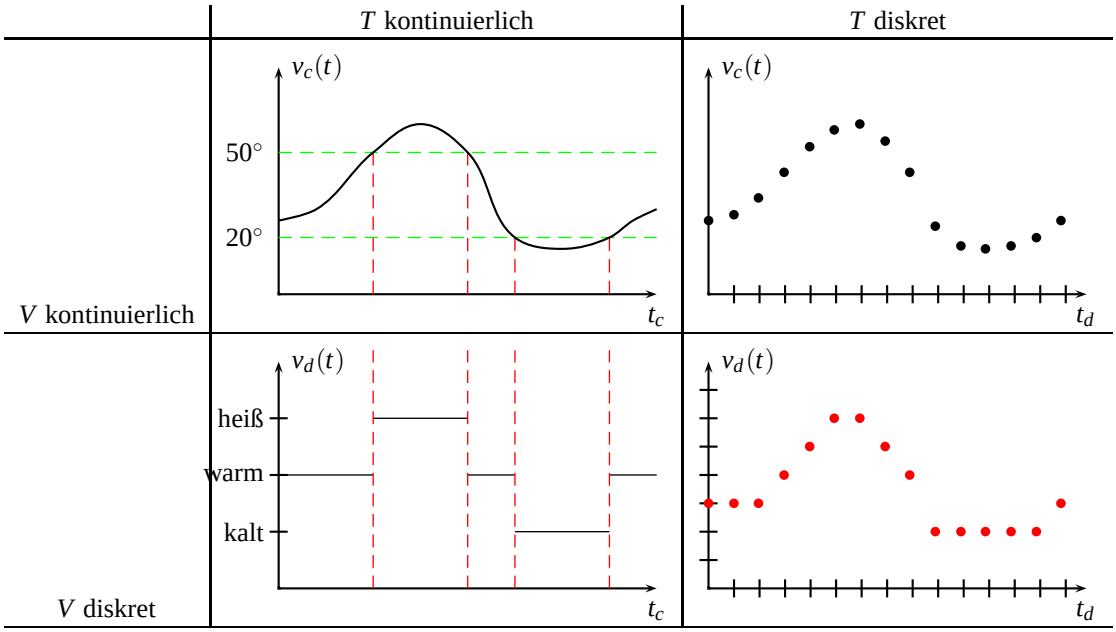
2.2 Signale (Zeitabhängige Variablen)

2.2.1 Definition

T: Menge aller Zeitpunkte
V: **Wertemenge** einer Variablen

$$\text{Signal } v : T \longrightarrow V$$

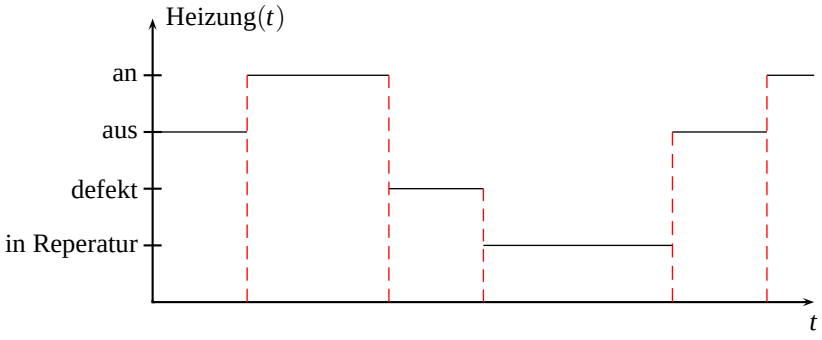
2.2.2 Klassen von Signalen



Für nicht quantifizierte Zeit ist diese Darstellung nicht sinnvoll.

Beispiel Temperatur (s.o.)

anderes Beispiel für linken unteren Quadranten:

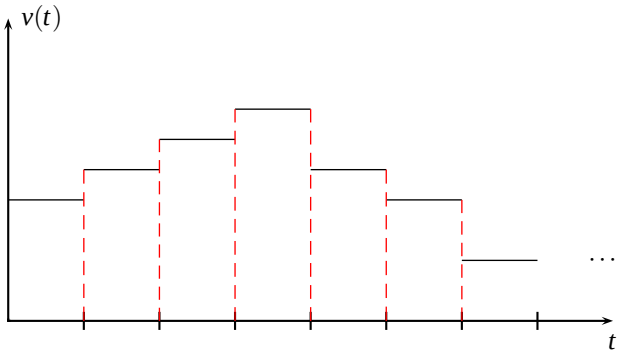


$\Longrightarrow$ Diskrete Wertemenge ist nicht notwendigerweise geordnet.

V kontinuierlich
$\downarrow$
V diskret $\quad = \quad$ “Diskretisierung” (für Informatiker: = eine Abstraktion)

T kontinuierlich
$\downarrow$
T diskret $\quad = \quad$ “Abtastung” (“Sampling”)

"Abtastung und Halten" ("Sample and Hold"):



"Quasikontinuierliche" Signale

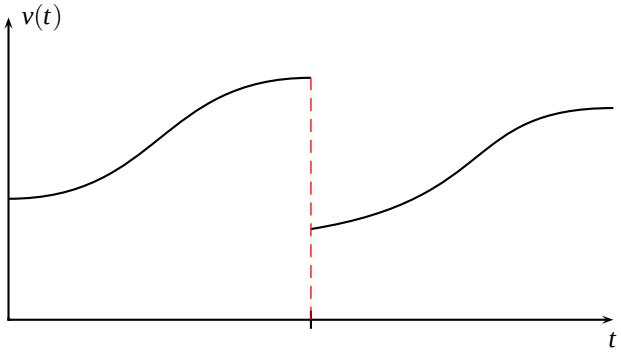
= relativ feine Diskretisierung, z.B. durch A/D-Wandler zur Verarbeitung im Rechner
im Gegensatz zu grober Diskretisierung auf wenige (häufig symbolisch interpretierte) Werte

Beispiel Temperaturwert in 12Bit-Auflösung ($\longrightarrow$ kontinuierliche Regelungstechnik) vs. $\{\text{kalt}, \text{warm}, \text{heiß}\}$
($\longrightarrow$ diskrete Steuerungstechnik)
$\longrightarrow$ Zweite Klasse wird zur besseren Unterscheidung auch als "ereignisdiskret" bezeichnet.

"Hybride" Signale

= Mischung aus (wert-)kontinuierlichen und (wert-)diskreten Signalen (in der Regel bei kontinuierlicher Zeitachse)

1. Beispiel:

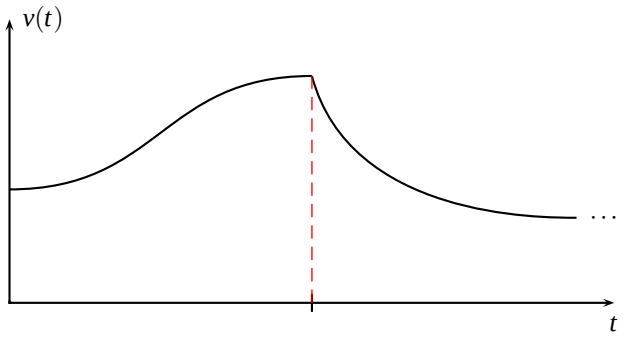


$$\lim_{\Delta t \longrightarrow 0} v(t' - \Delta t) \neq \lim_{\Delta t \longrightarrow 0} v(t' + \Delta t)$$

$\Longrightarrow$ "**Sprung**" (engl.: "Jump") als hybrides Phänomen.[2]

[2] Gewöhnlich wird weiteres Phänomen auch als hybrid bezeichnet, obwohl der Werteverlauf kontinuierlich bleibt: "Schalten" (engl.: "Switching")

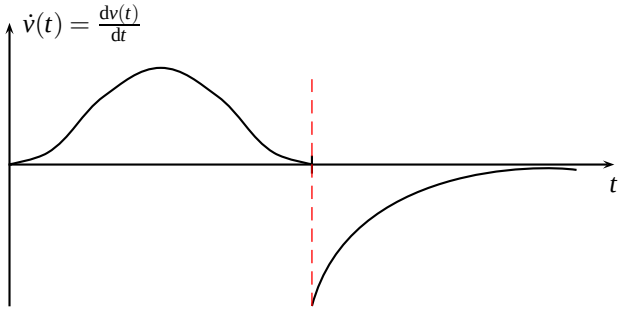
2. Beispiel



$$\lim_{\Delta t \longrightarrow 0} v(t' - \Delta t) = \lim_{\Delta t \longrightarrow 0} v(t' + \Delta t)$$

aber

$$\lim_{\Delta t \longrightarrow 0} \frac{\mathrm{d}v(t' - \Delta t)}{\mathrm{d}t} \neq \lim_{\Delta t \longrightarrow 0} \frac{\mathrm{d}v(t' + \Delta t)}{\mathrm{d}t}$$



Schalten = Sprung in erster zeitlicher Ableitung

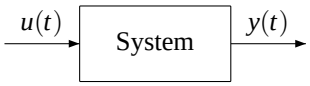
Im Folgenden vereinfacht:

- kontinuierliches Signal = wertkontinuierliches (incl. quasikontinuierliches) Signal
- diskretes Signal = ereignisdiskretes (also “grob wertdiskretes”) Signal

Wann werden welche Signalklassen zur Modellierung verwendet?

- kontinuierliche Signale:
  - physikalisch/chemische Vorgänge
  - Approximation von diskreten Vorgängen (z.B. Verkehrsflüsse)
- diskrete Signale:
  - von Menschen interpretierte oder spezifizierte Vorgänge (z.B. Ablauf eines Produktionsprozesses oder Programms, Einteilung in sichere und unsichere Zustände, Ausgabe diskreter Messglieder)
- hybride Signale:
  - Mischung von physikalischen und “Man-made”-Vorgängen (z.B. Batch-Prozess, Chemie)
  - Approximation von schnellen Vorgängen im Zusammenwirken mit langsamen Vorgängen

2.3 Systeme

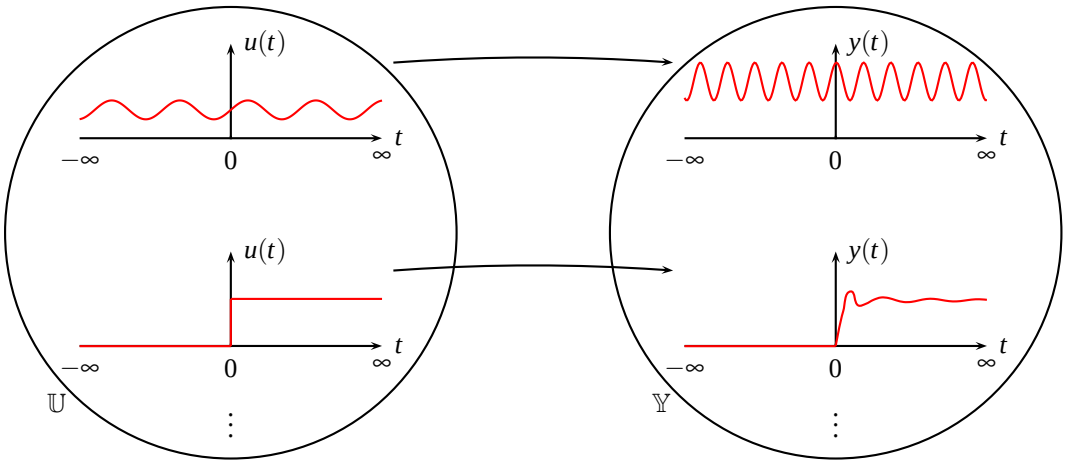


(Bei Ingenieuren beliebt: Ein-/Ausgangsdarstellung)

$u(t)$: Eingangssignal
$y(t)$: Ausgangssignal

Definition System S: Menge **aller** Eingangssignale $\mathbb{U} \longrightarrow \mathbb{Y}$ Menge aller Ausgangssignale

Beispiel



In der ingenierwissenschaftlichen Systemtheorie ist die Abbildung S vollständig definiert, d.h. für jedes Element aus $\mathbb{U}$ muß ein Ausgangssignal definiert sein.

= “freie” Eingänge (System kann keinen Eingangsverlauf “ablehnen”.)

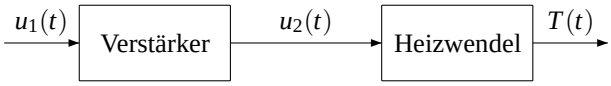
$u(t), y(t)$ kontinuierlich $\longrightarrow$ kontinuierliche Systeme
$u(t), y(t)$ diskret $\longrightarrow$ diskrete Systeme
Mischung oder $u(t), y(t)$ hybrid $\longrightarrow$ hybride Systeme

Statische vs. dynamische Systeme

Statisches System: $y(t) = f(u(t))$
z.B. $y(t) = 2 \cdot u(t)$

$\Longrightarrow$ Zu jedem Zeitpunkt t kann Wert von $y(t)$ aus aktuellem Wert $u(t)$ bestimmt werden. Die Vergangenheit ist irrelevant.

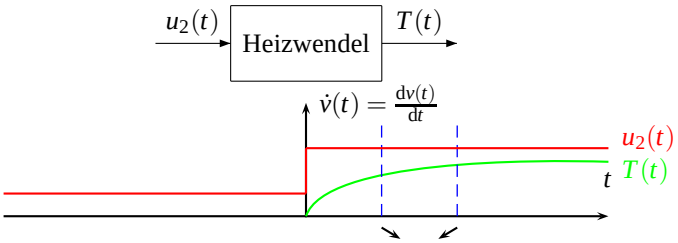
Beispiele: Operationsverstärker (lokalisierung!)



Schaltfunktion

Dynamisches System:

Beispiel:



Bei gleichem Wert von u unterschiedliche Werte von y!

$\Longrightarrow$ Vergangenheit ist relevant!
Salopp: Dynamische Systeme sind Systeme "mit Gedächtnis".

diskretes Beispiel: Schaltwerk

Beschreibung dynamischer Systeme

1. Möglichkeit:
   Operatoren, die die Abbildung $S : \mathbb{U} \longrightarrow \mathbb{Y}$ realisieren.

   Für kontinuierliche Signale gebräuchlich:

   - Übertragungsfunktion im Laplacebereich:

     $$y(t) = \mathfrak{L}^{-1}\{G(s) \cdot \mathfrak{L}\{u(t)\}\}$$

   - "Faltungsintegral"

     $$y(t) = \int_{-\infty}^{\infty} u(\tau) \cdot g(t-\tau)\,d\tau$$

   Für diskrete Signale?

2. Möglichkeit:
   Einführung einer **Hilfsgröße**, die die Information über die Vergangenheit (des Eingangsgrößenverlaufs) repräsentiert, die zur Bestimmung von y benötigt wird.

   (Wie nennt man eine solche Hilfsgröße?)

   $\longrightarrow$**Zustand** x (auch ein Signal nach unserer Definition)

Zustandsdarstellung dynamischer Systeme

1. kontinuierliche Systeme (auch $x(t)$ kontinuierlich!)

   (a) kontinuierliche Zeit:

   $$\begin{aligned} \frac{dx(t)}{dt} &= f(x(t),u(t)) \\ y(t) &= g(x(t),u(t)) \end{aligned}$$

   "Zustandsraumdarstellung" kontinuierlicher dynamischer Systeme
   Kalman, Brocket, 60er Jahre

   (b) diskrete Zeit:

   $$\begin{aligned} x(k+1) &= f(x(k),u(k)) \\ y(k) &= g(x(k),u(k)) \end{aligned}$$

2. diskrete Systeme ($u, y, x \in$ Menge von Symbolen!)
   (diskrete Zeit oder nicht-quantifizierte Zeit:)

$$\begin{array}{rrcll} & x(k+1) & = & f(x(k),u(k)) & \\ & y(k) & = & g(x(k),u(k)) & = \text{ Mealy-Automat} \\ \text{oder} & y(k) & = & g(x(k)) & = \text{ Moore-Automat} \end{array}$$

2.4 Berechnungsmodelle

Historisch anderer Zugang in der Informatik als in der ingenieurwissenschaftlichen Systemtheorie:

Modellierungsgegenstand = **Berechnungsvorgang**

- Bis 70er Jahre:

  "**Transformatorische**" Systeme
  Eingabe muß **in endlicher Zeit** in **korrekte** Ausgabe transformiert werden.

  Beispiel Berechnung einer mathematischen Funktion, Lösung eines Gleichungssystems, Sortieren eines Datensatzes, Suchen eines Datums etc.

  Fragestellung

  1. Terminiert die Berechnung?
  2. Ist das Ergebnis richtig?

  Terminierung + **partielle Korrektheit** = totale Korrektheit

  Nicht terminierende Berechnungen gelten als nutzlos.

  $\Longrightarrow$ Betrachtung immer über **endlichen** Zeitraum.

  Modelle für transformatorische Systeme:

  – Endliche Automaten über endlichen Wörtern
  – Turing-Maschine
  – Logiken mit Konstrukten der Art

  Vorzustand - Berechnungsschritt - Nachzustand
  $\langle xPy \rangle$

  (z.B. Hoare, Floyd)

- Seit ca. 70er Jahre:

  "Reaktive" Systeme
  (Keine einzelne Berechnung, sondern) **ständig** bereit, auf Änderungen in der Umgebung durch entsprechende Ausgaben **zu reagieren**.

  Beispiel Server, **Steuerungen**, Betriebssysteme $\Longrightarrow$ **eingebettete** Systeme

  Fragestellung

  1. Reagiert das System **richtig**? Korrektheit (Sicherheit)
  2. Reagiert das System **rechtzeitig**? Echtzeitfähigkeit
  3. Ist das System **immer wieder** bereit zu reagieren? Reaktivität (Lebendigkeit)

$\Longrightarrow$ Betrachtung über **unendliche** Zeiträume notwendig.

Modelle für **reaktive** Systeme: (Beispiele)

- Moore-Automaten
- ω-Automaten, z.B. Büchi-Automaten
- Kripke-Strukturen
- Temporale Logik

2.5 Kripke-Strukturen, Temporale Logik

Ursprung Modale Logik, Philosophie

- Logische Aussagen sind **nicht universell** wahr oder falsch
- Gültigkeit hängt von der "Welt" ab, in der sie gemacht werden
- Es gibt mehrere "Welten"

Darstellung der "Welten" und Wechsel zwischen den "Welten" in **Kripke-Struktur**: M**:**

Definition $M = (X, X_0, f, V, \lambda)$

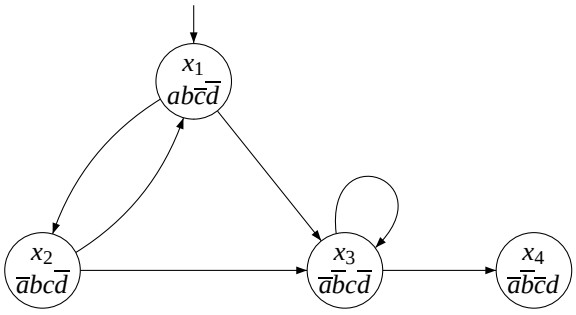
X:	Menge von Zuständen (= "Welten")
$X_0 \subseteq X$:	Menge von Anfangszuständen
$f \subseteq X \times X$:	(links-) totale Zustandsübergangsrelation (oder $f : X \longrightarrow 2^X = \mathfrak{P}\{X\}$)
V:	Menge von atomaren Formeln (logische Aussagen)
$\lambda : X \longrightarrow 2^V$:	Zustandsmarkierung (Bedeutung: Welche Formeln sind in welchem Zustand gültig?)

Kripke-Struktur = markiertes Transitionssystem

Beispiel

$$\begin{aligned} X &= \{x_1, x_2, x_3, x_4\}, X_0 = \{x_1\} \\ V &= \{a, b, c, d\} \end{aligned}$$

f und λ dargestellt im Zustandsgraph:



(gestrichene Formeln kann man weglassen)

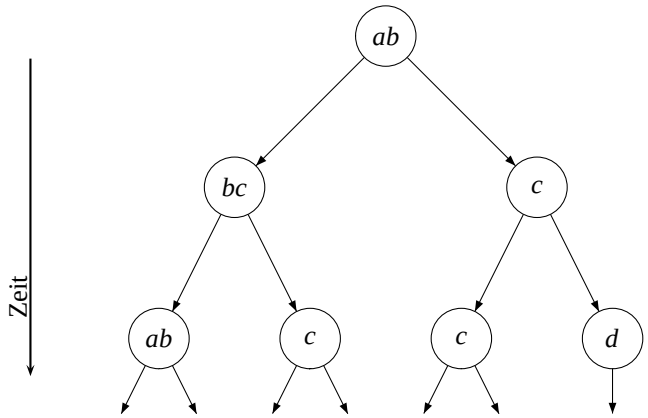
Für uns **temporale** Interpretation wichtig:
("Welten" wechseln mit der Zeit.)

$\Longrightarrow$ Gültigkeit von logischen Aussagen ändert sich mit der Zeit.
Beispiel: Borussia Dortmund ist Deutscher Meister

Übliche Interpretation:
Wechsel von Zuständen = Berechnungsschritt
$\lambda \mathrel{\widehat{=}}$ Werte der Programmvariablen zwischen den Berechnungsschritten

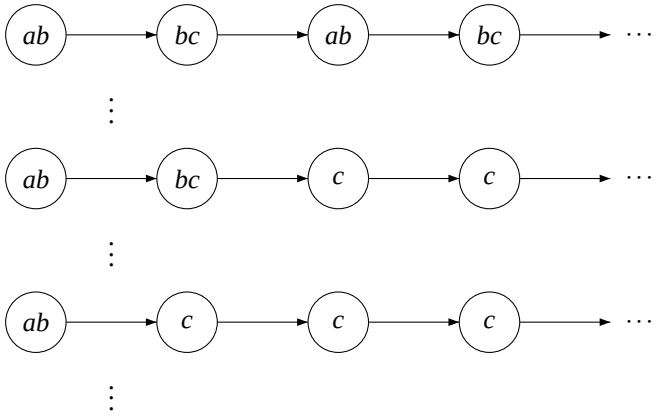
"**Entfaltung**" der Kripke-Struktur über die Zeit $\longrightarrow$ **Computation Tree**

Beispiel



$\vdots$ (unendliche Tiefe)

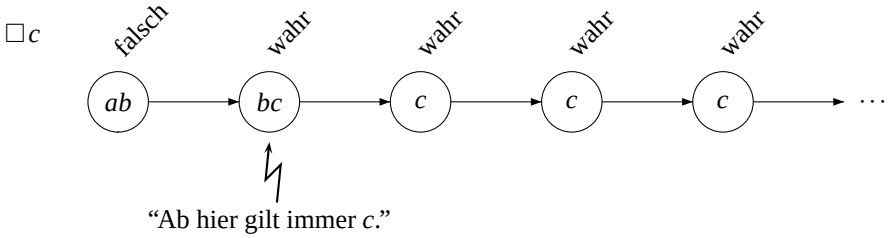
$\Longrightarrow$ Menge von Pfaden:



In jedem Knoten eines Pfades kann man nun **Aussagen über die Zukunft** machen.

$\longrightarrow$ **Temporale Operatoren**

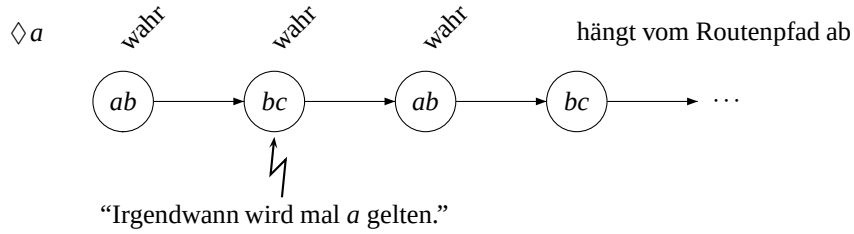
1. Beispiel



"Ab hier gilt immer *c*."

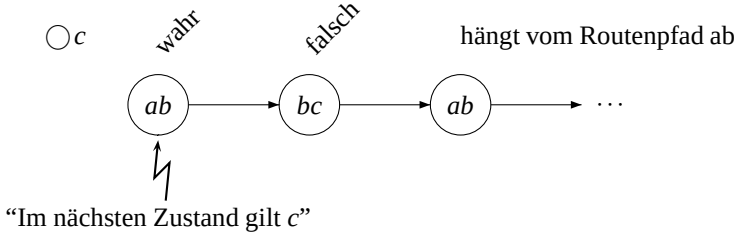
Englisch: Globally *c*, Always *c*
Symbol: G*c* oder □*c* ("Box")

2. Beispiel



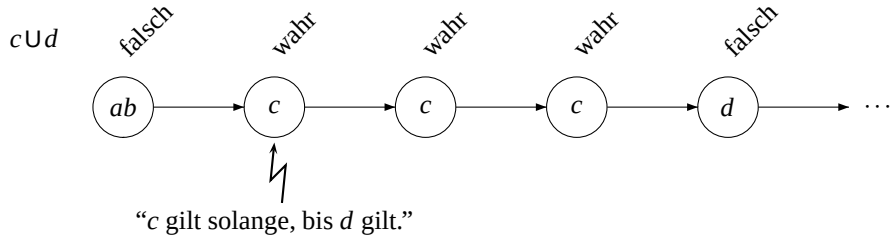
Englisch: Future *a*, Sometimes *a*
Symbol: F*a* oder ◊*a* ("Diamond")

3. Beispiel



Englisch: Next *c*
Symbol: X*c* oder ◯*c*

4. Beispiel



Englisch: *c* until *d*
Symbol: *c* U *d*

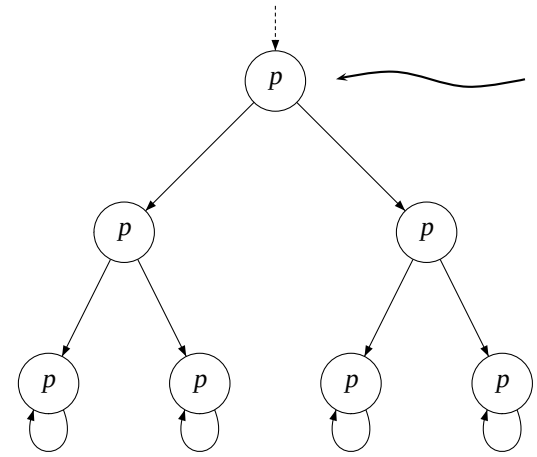
Um in den **Knoten des Baumes** Aussagen über die Zukunft zu machen, benötigt man zusätzlich **Pfadquantoren**:

A *p*: Für **alle** vom betreffenden Knoten ausgehenden Pfade gilt *p*

E *p*: Es existiert mindestens ein von dem betreffenden Knoten ausgehender Pfad, für den *p* gilt.

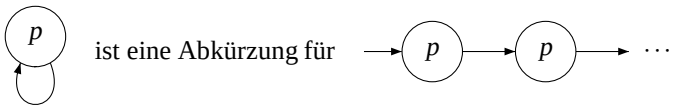
$\Longrightarrow$ 8 Kombinationen von Pfadquantor mit temporalem Operator:

1.

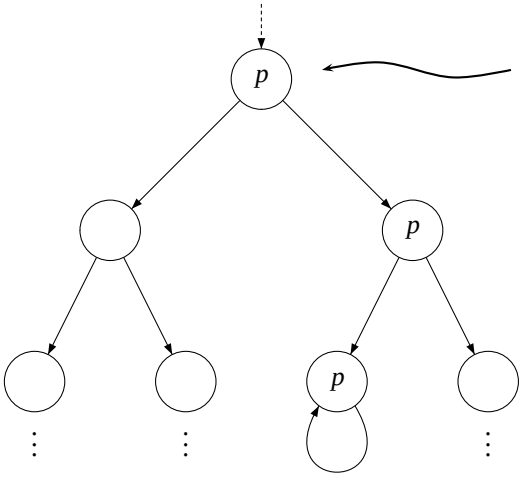


Hier gilt: AG p

= "Auf allen von hier ausgehenden Pfaden gilt immer p."
oder:
"p ist eine Invariante."



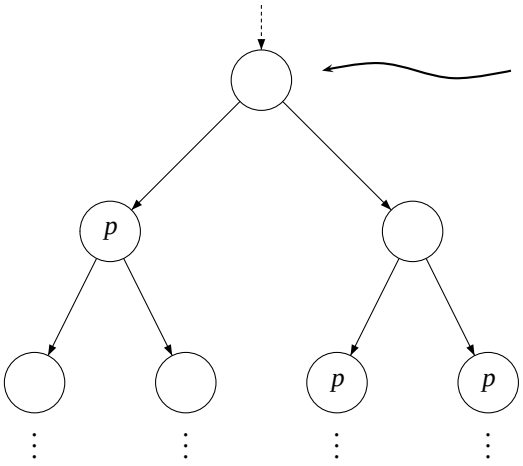
2.



Hier gilt: EG p

= "Es gibt mindestens einen von hier ausgehenden Pfad, für den immer p gilt."
oder:
"Immer p ist möglich."

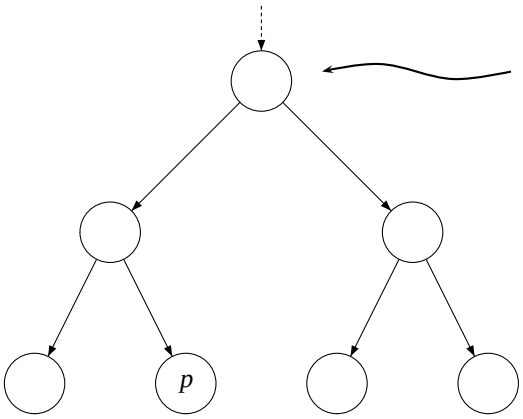
3.



Hier gilt: AF p

= "Für alle von hier ausgehenden Pfade gilt, dass irgendwann p gilt."
oder:
"p ist unvermeidlich."

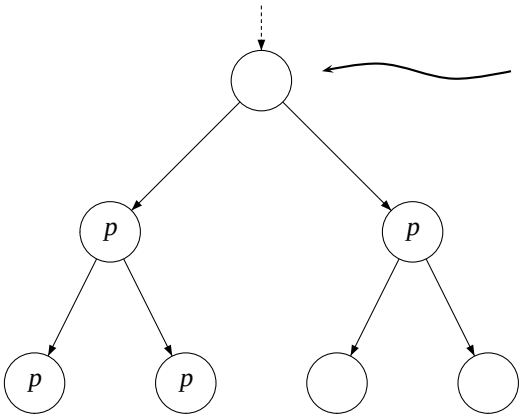
4.



Hier gilt: EF p

$=$ "Es gibt einen von hier ausgehenden Pfad, auf dem irgendwann p gilt."
oder:
"p ist möglich."
($\Longleftrightarrow$ Erreichbarkeitsproblem: Ist ein Zustand erreichbar, für den p gilt?)

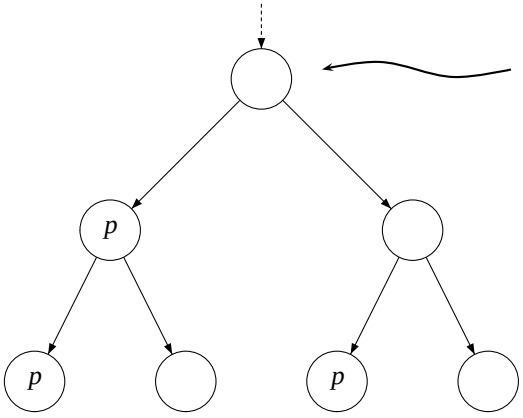
5.



Hier gilt: AX p

$=$ "Auf allen von hier ausgehenden Pfaden gilt im nächsten Knoten p."

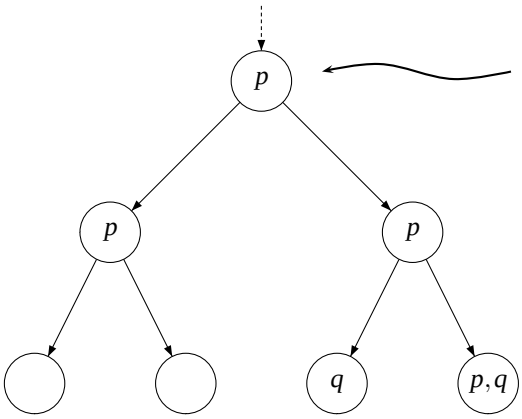
6.



Hier gilt: EX p

$=$ "Es gibt mindestens einen Pfad, auf dem im nächsten Zustand p gilt."

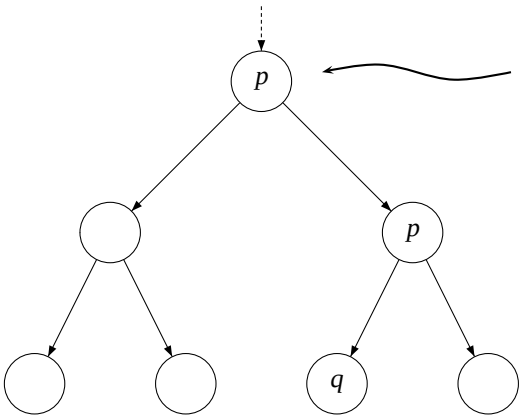
7.



Hier gilt: $p \,\mathsf{AU}\, q$

$=$ "Auf allen von hier ausgehenden Pfaden gilt mindestens solange in jedem Knoten p, bis ein Knoten kommt, in dem q gilt."

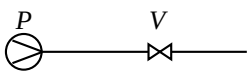
8.



Hier gilt: $p \,\mathsf{EU}\, q$

$=$ "Es gibt mindestens einen von hier ausgehenden Pfad, auf dem mindestens solange in jedem Knoten p gilt, bis q gilt."

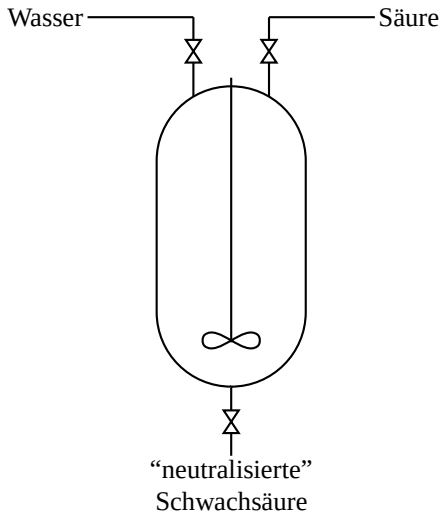
Beispiele Verfahrenstechnik:

- $\mathsf{AG}(\neg$Überlauf$)$: Es passiert nie ein Überlauf
- $\mathsf{AG}($Pumpe an $\longrightarrow$ davor liegendes Ventil offen$)$
- $\mathsf{AG}($Notaus gedrückt $\longrightarrow \mathsf{AF}($Anlage stromlos$))$
- $\mathsf{AG}($Start $\longrightarrow \mathsf{AF}($Charge fertig$))$
- $\mathsf{AG}(\mathsf{AF}\,$Charge fertig$)$
- 

$\mathsf{AG}(P \longrightarrow V)$

Noch ein Beispiel:

Beispiel Neutralisationsreaktor:



Variablen:

- Wasserventil: BOOL
- Säureventil: BOOL
- offen = TRUE

Anfangszustand: Behälter leer, alle Ventile zu

Alter Verfahrenstechnikerspruch:
"Erst das Wasser, dann die Säure, sonst geschieht das Ungeheure."

In Temporaler Logik?

Die Pfadquantoren können negiert werden:

	$\neg \mathsf{A}\, p$:		Nicht für alle Pfade gilt p.
$=$	$\mathsf{E} \neg p$	$=$	Es gibt mindestens einen Pfad, für den $\neg p$ gilt.
	$\neg \mathsf{E}\, p$:		Es gibt keinen Pfad, für den p gilt.
$=$	$\mathsf{A} \neg p$	$=$	Für alle Pfade gilt $\neg p$.

$\Longrightarrow$ Kombinationen von Pfadquantor und temporaler Operator können umgeformt werden:

$$\mathsf{AG}\, p = \neg \mathsf{EF} \neg p$$

Beispiel: $\mathsf{AG}(\neg \text{Überlauf}) = \neg \mathsf{EF}(\text{Überlauf})$

$$\mathsf{AF}\, p = \neg \mathsf{EG} \neg p$$

(Uns interessieren keine Eigenschaften in irgendwelchen Knoten in einzelnen Pfaden oder einzelnen Knoten im Berechnungsbaum, sondern:)

Systemeigenschaften:

$$M, x_0 \models p$$

Für das System M (Modell) gilt (**im Anfangszustand**) p.

Das Verhalten des Systems kann repräsentiert sein durch

1. den Berechnungsbaum,
2. die Menge P^ω aller Pfade im Berechnungsbaum.

$\Longrightarrow$ Aussagen über das Systemverhalten können folgende Form haben

1. Für den Berechnungsbaum gilt:

   $\langle$Formel mit Pfadquantoren und temporalen Operatoren$\rangle$

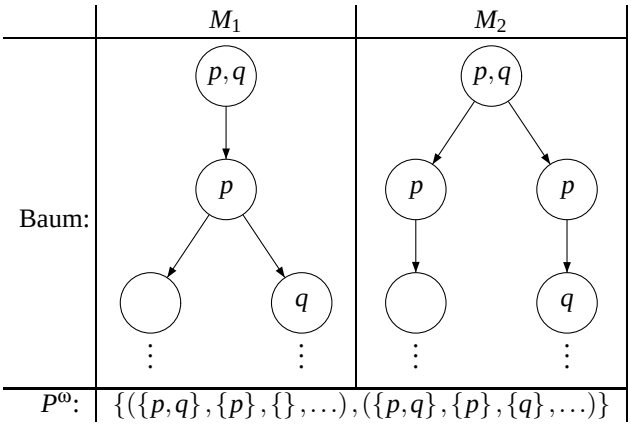
2. Für alle Pfade in der Menge P^ω gilt:

   $\langle$Formel mit temporalen Operatoren ohne Pfadquantoren$\rangle$

   (hier: Pfadquantoren nicht mehr notwendig. $\longrightarrow$ Warum benötigt man überhaupt Pfadquantoren?)

1. und 2. ist nicht äquivalent!

Beispiel



Beispiel für eine Eigenschaft, die in M_1 gilt, aber nicht in M_2:

$$M_1, x_0 \models \mathsf{AX}(\mathsf{EX}\, q \wedge \mathsf{EX} \neg q)$$

Wenn man nur die Menge der Pfade betrachtet, ist dieser Unterschied nicht zu erkennen.

Man benutzt den Zeitbegriff, um beide Ansätze zu unterscheiden:

Betrachtung der Pfade (2.) $\longrightarrow$ "lineare Zeit"
Betrachtung des Berechnungsbaums (1.) $\longrightarrow$ "verzweigende Zeit" (branching time)

Entsprechend werden die dazugehörigen Logiken definiert:
Branching time logic, Computation tree logic (CTL) vs. linear time logic (LTL)

Salopp (formale Definition $\longrightarrow$Literatur)

CTL*:	beliebige Verwendung von Pfadquantoren und temporalen Operatoren
CTL:	Einschränkung: Pfadquantoren und temporale Operatoren müssen immer zusammen in einem Paar benutzt werden. Beispiel: $\mathsf{AG}(\mathsf{EF}\, p)$, nicht $\mathsf{A}(\mathsf{FG}\, p)$ (wie schon immer gemacht).

LTL: Einschränkung: Es wird nur (implizit) ein Allquantor am Anfang der Formel verwendet,[3] sonst nur temporale Operatoren.
Beispiel: $\mathsf{A}(\mathsf{FG}\,p)$ geschrieben als $\mathsf{FG}\,p$

- CTL und LTL sind echte Untermengen von CTL*.
- CTL und LTL sind unterschiedlich aussagekräftig.

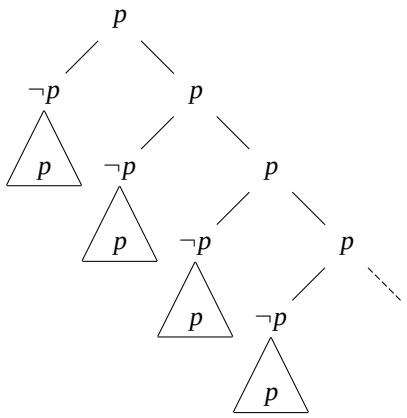
Beispiele:

In CTL, aber nicht in LTL: $\mathsf{AG}(\mathsf{EF}\,p)$
In LTL, aber nicht in CTL: $\mathsf{FG}\,p$
in CTL*, aber nicht in LTL/CTL: $\mathsf{AG}(\mathsf{EF}\,p) \vee \mathsf{FG}\,p$

Nachtrag Clarke-Buch: LTL-Formel $\mathsf{FG}\,p$ nicht in CTL darstellbar.

Frage $\mathsf{FG}\,p \equiv \mathsf{AF}(\mathsf{AG}\,p)$

Gegenbeispiel:



- $\mathsf{FG}\,p$?
  1. Bestimme Menge aller Pfade.
  2. Für jeden Pfad gilt:
     - Entweder $\mathsf{G}\,p$ (der eine rechts außen)
     - oder: Es gibt nur einen Knoten mit $\neg p$.

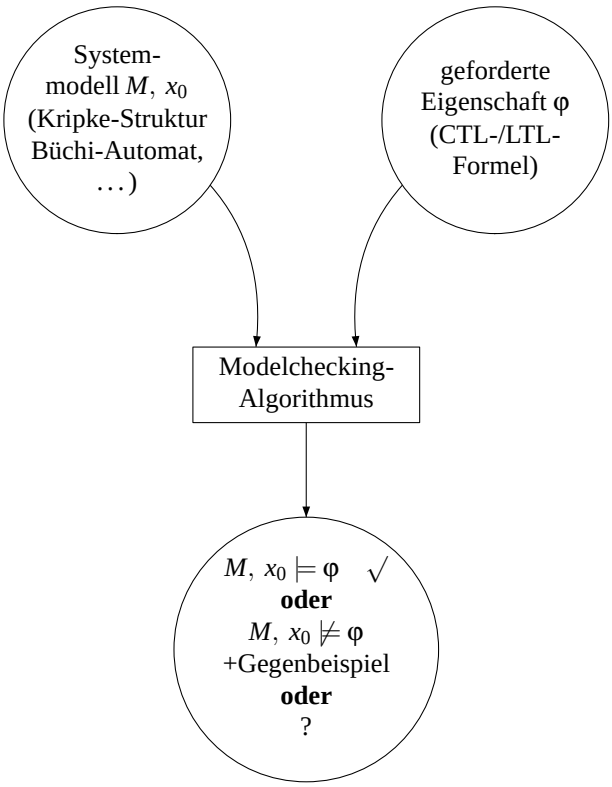
  $\Longrightarrow \mathsf{FG}\,p$ gilt.

- $\mathsf{AF}(\mathsf{AG}\,p)$?

  Es muß möglich sein, auf allen Pfaden einen Knoten zu finden, ab dem $\mathsf{AG}\,p$ gilt.
  Das ist auf dem Pfad rechts außen nicht möglich, da immer die Möglichkeit besteht, noch nach links in einen Knoten $\neg p$ abzubiegen.
  $\Longrightarrow \mathsf{AF}(\mathsf{AG}\,p)$ gilt nicht.

[3] = Für alle Pfade gilt, …

2.6 Modelchecking



2.6.1 CTL-Modelchecking

1. Umformung der CTL-Formel für φ, so daß nur noch verwendet werden: $\neg$, $\vee$, EX, EU, EG

$$\begin{aligned}
\mathsf{AG}\,p &= \neg\mathsf{EF}\neg p \\
\mathsf{AF}\,p &= \neg\mathsf{EG}\neg p \\
\mathsf{AX}\,p &= \neg\mathsf{EX}\neg p \\
\mathsf{A}(p\,\mathsf{U}\,q) &= \underbrace{\mathsf{AF}\,q}\wedge\neg\mathsf{E}(\neg p\,\mathsf{U}\neg(p\vee q)) \\
&= \neg\mathsf{EG}\neg q\wedge\neg\mathsf{E}(\neg p\,\mathsf{U}\neg(p\vee q)) \\
&= \neg(\mathsf{EG}\neg q\vee\mathsf{E}(\neg p\,\mathsf{U}\neg(p\vee q))) \\
\mathsf{EF}\,p &= \mathsf{E}(\mathsf{true}\,\mathsf{U}\,p)
\end{aligned}$$

2. Für jede Unterformel von innen (atomare Propositionen) nach außen in der Verschachtelung: Absuchen des Transitionssystems und markieren der Zustände mit den gültigen Unterformeln.

Beispiel $\varphi = \mathsf{E}(f\,\mathsf{U}(\neg\mathsf{EX}g))\vee\mathsf{EG}\neg h$
von innen nach außen:

(a) f, g, h
(b) $\mathsf{EX}g, \neg h$
(c) $\neg\mathsf{EX}g, \mathsf{EG}\neg h$
(d) $\mathsf{E}(f\,\mathsf{U}\neg\mathsf{EX}g)$

(e) φ

$\Longrightarrow$ Folgende Teilalgorithmen werden gebraucht:

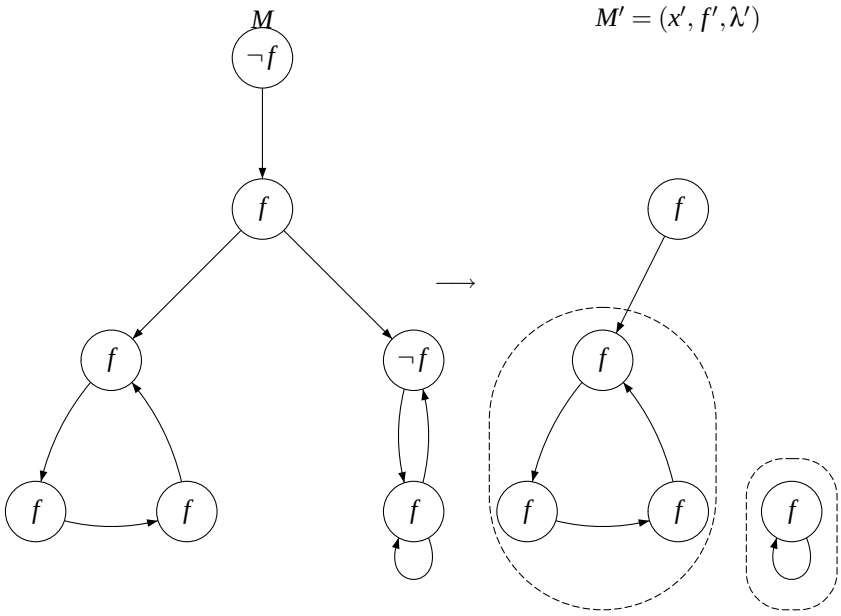
(a) $M, x_0 \models p$
(b) $M, x_0 \models \neg p$
(c) $M, x_0 \models p_1 \vee p_2$
(d) $M, x_0 \models \mathsf{EX}\, p$
(e) $M, x_0 \models \mathsf{E}(p_1 \,\mathsf{U}\, p_2)$
(f) $M, x_0 \models \mathsf{EG}\, p$

(a) – (d): Alle Zustände einmal besuchen und Gültigkeit analysieren anhand der bereits untersuchten Unterformeln für diese Zustand bzw. seine Folgezustände (\EX).

(e): $M, x_0 \models \mathsf{E}(f_1 \,\mathsf{U}\, f_2)$:
Für alle Zustände, in denen f_2 gilt: Rückwärtssuche: alle Zustände, die über einen Pfad erreichbar sind, auf dem jeder Zustand mit f_1 markiert ist, werden mit $\mathsf{E}(f_1 \,\mathsf{U}\, f_2)$ markiert.

(f): $M, x_0 \models \mathsf{EG}\, f$

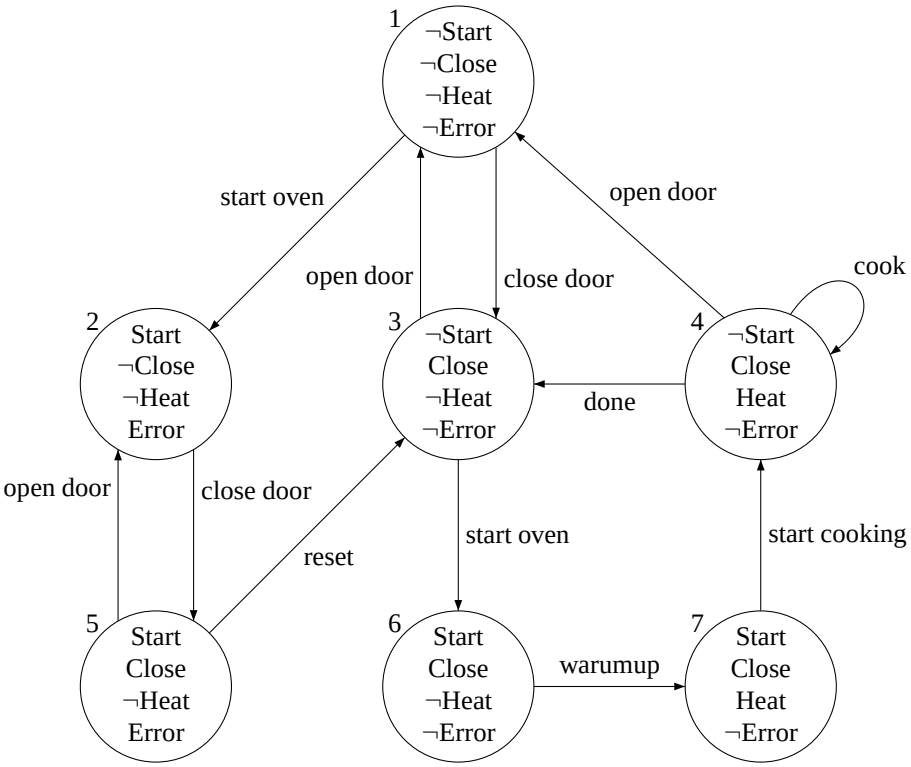
(a) Eliminierung aller Zustände mit $\neg f$.
Beispiel:



(b) Bestimmung der Menge *SSC* aller nicht-trivialer stark zusammenhängender Teilgraphen in M'.

(c) $M, x \models \mathsf{EG}\, f \iff$ 1. $x \in X'$
2. Es existiert ein Pfad von x in einem nicht-trivialen stark zusammenhängenden Teilgraphen von M'

Beispiel Mikrowelle[4]



Eigenschaftsvektor (tupel) $e = (\text{Start}, \text{Close}, \text{Heat}, \text{Error})$

zu überprüfende CTL-Formel:

$$\begin{aligned}
\mathsf{AG}\,(\text{Start} \longrightarrow \mathsf{AF}\,\text{Heat}) &= \neg\,\mathsf{EF}\,\neg\,(\neg\text{Start} \vee \mathsf{AF}\,\text{Heat}) \\
&= \neg\,\mathsf{EF}\,(\text{Start} \wedge \neg\,\mathsf{AF}\,\text{Heat}) \\
&= \neg\,\mathsf{EF}\,(\text{Start} \wedge \neg\neg\,\mathsf{EG}\,\neg\text{Heat}) \\
&\quad (\mathsf{EF}\,f \text{ als Abkürzung für } \mathsf{E}\,(\text{true}\,\mathsf{U}\,f)) \\
&= \neg\,\mathsf{E}\,(\text{true}\,\mathsf{U}\,(\text{Start} \wedge \mathsf{EG}\,\neg\text{Heat}))
\end{aligned}$$

Notation: $S(p) =$ Menge aller Zustände, die mit der Unterformel p markiert sind.

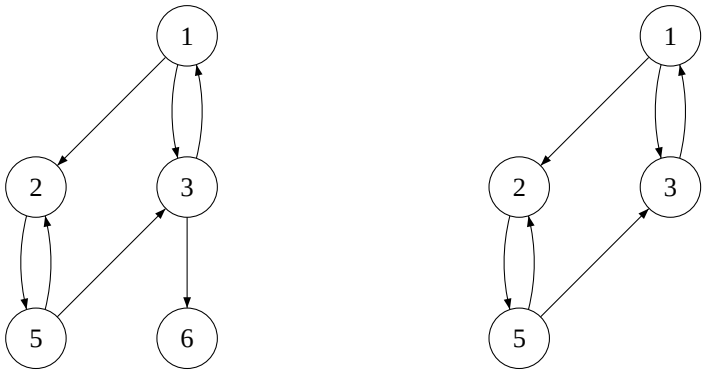
$$\neg\,\mathsf{E}\,(\text{true}\,\mathsf{U}\,(\text{Start} \wedge \mathsf{EG}\,\neg\text{Heat}))$$

1. $S\,(\text{Start}) = \{2,5,6,7\}$, $S\,(\neg\text{Heat}) = \{1,2,3,5,6\}$
   (in Graph markieren)

[4] aus Clarke et al, Model Checking, MIT Press 1999

2. $S(\mathsf{EG}\,\neg\text{Heat}) = \{1,2,3,5\}$

M': (Menge nicht-trivial st. zusammenhängender Teilgraphen) *SSC*:



3. $S(\text{Start} \wedge \mathsf{EG}\,\neg\text{Heat}) = \{2,5\}$
4. $S(\mathsf{E}(\text{true}\,\mathsf{U}\,(\text{Start} \wedge \mathsf{EG}\,\neg\text{Heat}))) = \{1,2,3,4,5,6,7\}$
   Beginnen mit $\{2,5\}$, dann rückwärts
5. $S(\neg\mathsf{E}(\text{true}\,\mathsf{U}\,(\text{Start} \wedge \mathsf{EG}\,\neg\text{Heat}))) = \emptyset$
   $\Longrightarrow$ Anforderung ist nicht erfüllt.

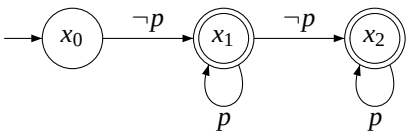
2.6.2 LTL-Modelchecking

M, x_0: Büchi-Automat A_M ($F \subseteq X$: Menge von akzeptierenden Zuständen; Lauf ist akzeptierend, wenn mindestens ein Zustand aus F unendlich oft besucht wird.)

φ: LTL-Formel

Beispiel

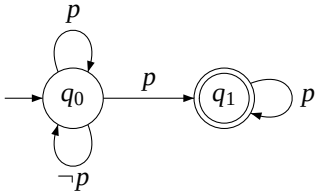
A_M: φ



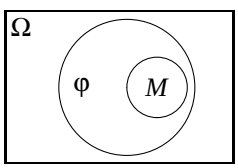
$\mathsf{FG}\,p$

$$\mathfrak{L}\{M\} = (\neg p p^\omega) + (\neg p p^* \neg p p^\omega)$$

1. $\mathsf{FG}\,p \mathrel{\widehat{=}} (\neg p + p)^* p^\omega$
2. dazugehöriger Büchi-Automat A_φ :



$$M, x_0 \models \varphi \iff \mathfrak{L}\{A_M\} \subseteq \mathfrak{L}\{A_\varphi\}$$
$$\iff \mathfrak{L}\{A_M\} \cap \mathfrak{L}\{A_\varphi\} = \emptyset$$



Benötigt wird:

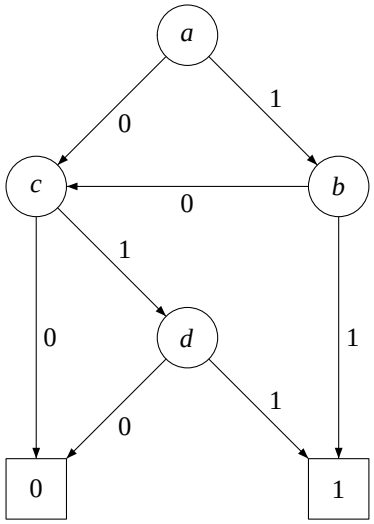
(a) Schnitt
(b) Komplement
(c) Test auf Leere

2.6.3 Symbolic Model Checking

(z.B.: SMV)

- bisher: Suche über einzelne Zustände
  Problem: Davon gibt es viele (exponentiell in Anzahl der Teilsysteme)
- Effizienzgewinn (praktisch) durch "symbolische" Repräsentation von Zustands**mengen**.
  = Logische Formel repräsentiert die Menge von Zuständen, in denen sie gilt.
- Effiziente Datenstruktur für diese Formeln: OBDDs (Ordered Binary Decision Diagrams; Akers, 1960s, Bryant 1980s)

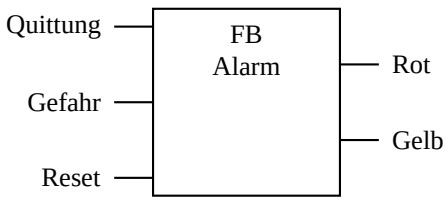
Beispiel $(a \wedge b) \vee (c \wedge d)$



$\longrightarrow$ Details siehe Clarke et al.

Anwendungsbeispiel

Funktionsblock für ein SPS (Speicherprogrammierbare Steuerung) zur Alarmverarbeitung



Anforderungsbeschreibung:

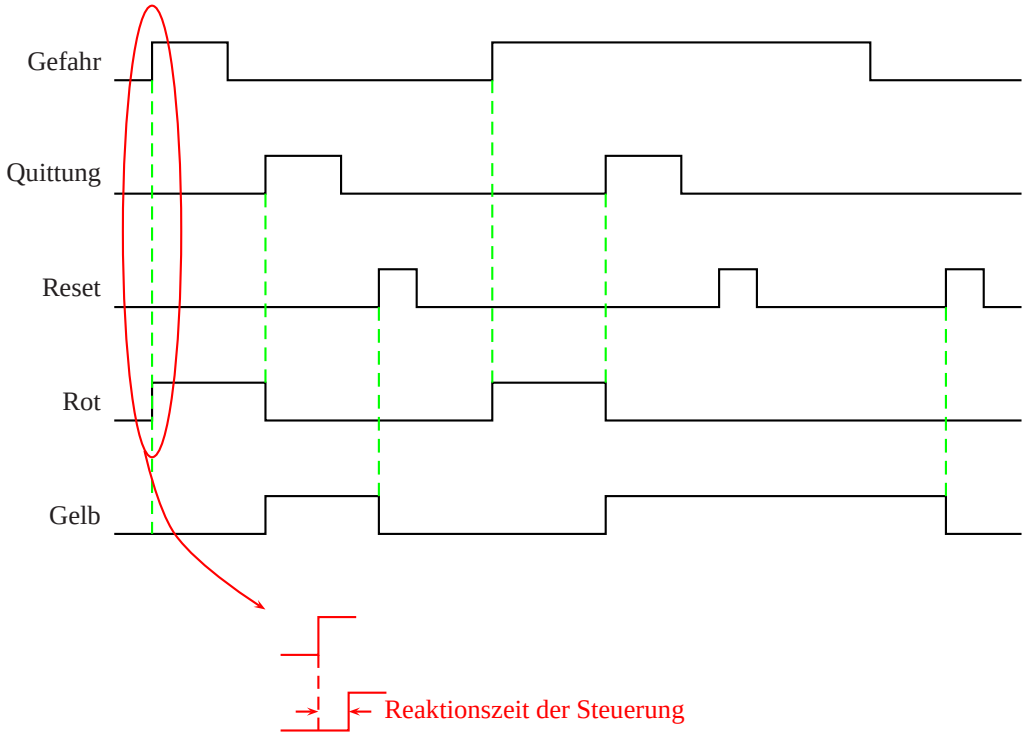
1. Bei Eintritt von Gefahr muss Rot angehen.
2. Nach Quittung durch den Bediener wird Rot abgeschaltet und Gelb angeschaltet.
3. Nach Beseitigen der Gefahr kann Gelb durch Reset gelöscht werden.

Vorschlag für Lösung (Sprache: AWL = Anweisungsliste):

```
VAR_IN
  Gefahr, Quittung, Reset: BOOL;
END_VAR
VAR_OUT
  Rot, Gelb: BOOL := FALSE;
END_VAR

LD   Gefahr
S    Rot
LD   Quittung
S    Gelb
R    Rot
LD   Reset
ANDN Gefahr
R    Gelb
```

~~$AG(Gefahr \longrightarrow Rot)$~~
~~$AG(AF\,Rot)$~~
$AG\,(Gefahr \longrightarrow AF\,Rot)$
$AG\,(Gefahr \longrightarrow (Gelb \vee AF\,Rot))$
$\longrightarrow$ Hilfreich für besseres Verständnis der Anforderungen: **Zeitdiagramm**:



1. $\mathsf{AG}\,((\text{Gefahr} \wedge \neg\text{Gelb}) \longrightarrow \mathsf{AF}\ \text{Rot})$[5]

2. $\mathsf{AG}((\text{Rot} \wedge \text{Quittung} \longrightarrow \sout{\mathsf{AF}(\neg\text{Rot} \wedge \text{Gelb}))}$
   $\longrightarrow \mathsf{A}(\text{Rot}\,\mathsf{U}(\neg\text{Rot} \wedge \text{Gelb}))$

3. $\mathsf{AG}\,((\text{Reset} \wedge \neg\text{Gefahr}) \longrightarrow \mathsf{AF}\,(\neg\text{Gelb}))$

Modell des Systems:

1. **Wie wird das Programm abgearbeitet?** (Berechnungsmodell)

   Expertenwissen nötig:

   (a) **Wie ist die Semantik von AWL (Anweisungsliste)?**

   AWL:

   - Anweisung :`<label>: <Operator><Operand>`
   - ein Akkumulator (hier 1 Bit): Bezeichnung u.a.: AE “Aktuelles Ergebnis”
   - Operationen:

```
LD   x: AE := x
ANDN x: AE := AE∧¬x
S/R  x: x := { 1/0  if AE = 1
             { x    else
```

   $\Longrightarrow$ Zusammenfassung des AWL-Programms zu “Berechnungsschritten:

```
Schritt 1 { LD   Gefahr } if Gefahr = 1 then Rot = 1 {else Rot = Rot}
          { S    Rot   }

Schritt 2 { LD   Quittung }
          { S    Gelb     } if Quittung = 1 then {Gelb := 1; Rot := 0}
          { R    Rot      }

Schritt 3 { LD   Reset  }
          { ANDN Gefahr } if Reset = 1 AND Gefahr = 0 then Gelb := 0
          { R    Gelb   }
```

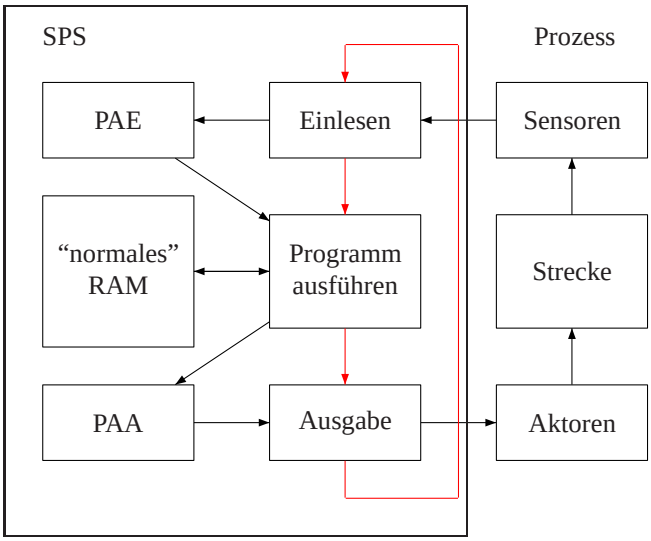
[5] AF in der Praxis häufig zu schwach:
Beispiel: $\mathsf{AG}\,((\text{Gefahr} \wedge \neg\text{Geld}) \longrightarrow \underline{\mathsf{AF}}\,\text{Rot})$
Rot muß eigentlich innerhalb einer Mindestzeit angehen.
Abhilfen:

(a) $\mathsf{AG}\,((\text{Gefahr} \wedge \neg\text{Gelb}) \longrightarrow \underline{\mathsf{AX}}\,\text{Rot})$ $\longrightarrow$ häufig zu stark (s. Modellierung weiter unten)

(b) Echtzeitlogiken: $\mathsf{AG}\,((\text{Gefahr} \wedge \neg\text{Gelb}) \longrightarrow \mathsf{AF}_{<5\text{sec}}\,\text{Rot})$ $\longrightarrow$ andere Modellklasse (hybride Systeme)

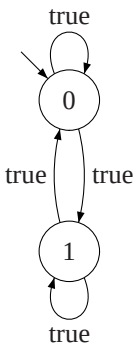
(c) Takte/Zyklen “zählen”: $\mathsf{AG}((\text{Gefahr} \wedge \neg\text{Gelb}) \longrightarrow \underbrace{\mathsf{AX}(\mathsf{AX}(\ldots(\mathsf{AX}\,\text{Rot})\ldots))}_{\text{geeignete Anzahl}}$ $\longrightarrow$ problematisch: Was ist ein Zyklus?

(b) **Wie funktioniert eine SPS?**
“Permanent zyklischer Betrieb”

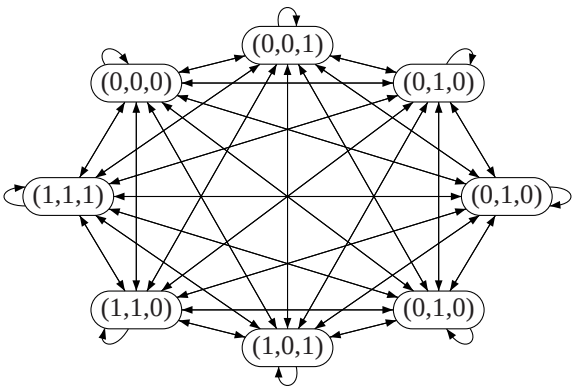


PAE: Prozessabbild der Eingänge
PAA: Prozessabbild der Ausgänge

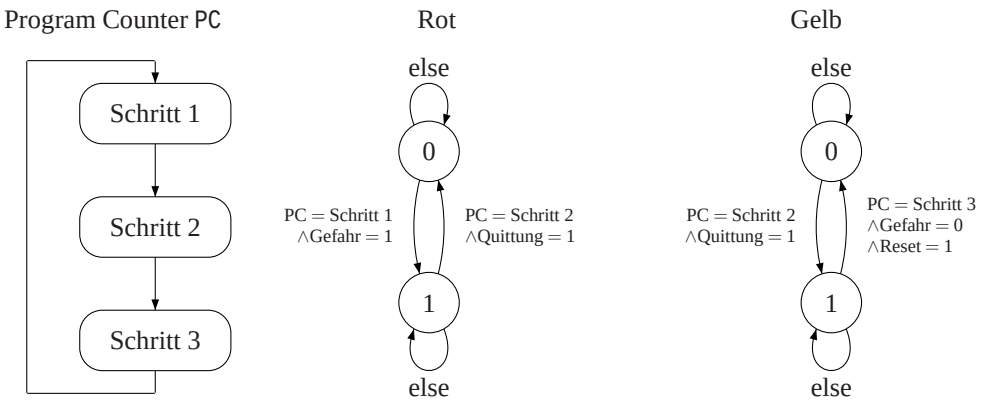
2. **Wie verhält sich die Strecke?**
Schlechtester Fall: Man weiß nichts.
Hier: Gefahr, Quittung, Reset sind **freie Eingänge** $\Longrightarrow$ Keine Einschränkung sinvoll.
Modell für Gefahr, Quittung, Reset:



Bediener kann allen möglichen Mist machen.
Synchrones Produkt von Modellen für Gefahr, Quittung, Reset (nur zur Illustration, macht Werkzeug):



1. Versuch der Modellierung:



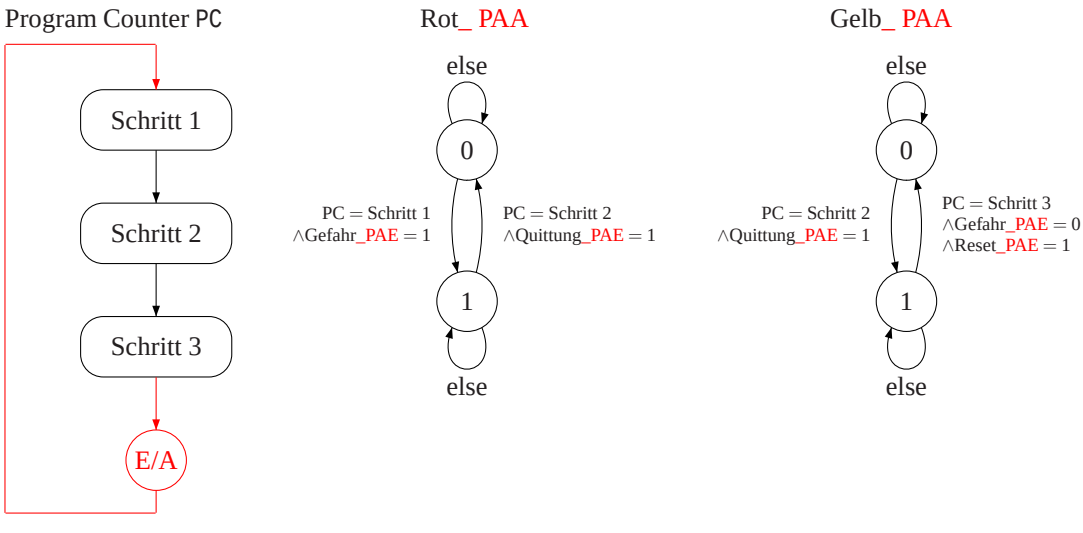
Warum funktioniert das nicht?

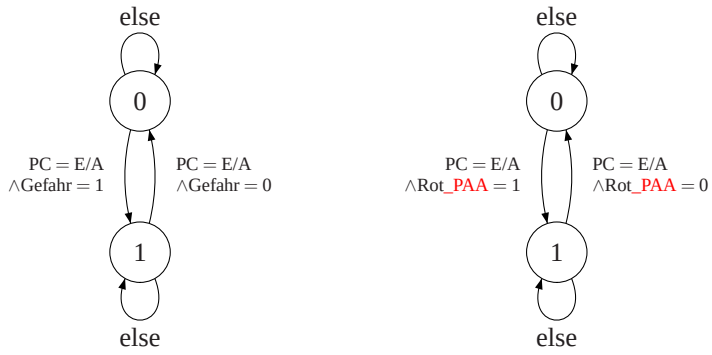
Wirkliches Verhalten wird nicht wiedergegeben.

Eingänge: Können sich hier von Schritt zu Schritt ändern.

Ausgänge: Wert in Zwischenschritten ohne Relevanz für Ein-/Ausgangsverhalten.

2. Versuch: Modellierung von PAE und PAA:





(analog für Quittung_PAE, Reset_PAE) (analog für Gelb)

$\Longrightarrow$ SMV-Code: `alarm.smv`

```
-- Beispiel Alarmverarbeitung
-- Stefan Kowalewski, 28.06.99
-- Überarbeitung 16.06.05



MODULE main


VAR
  -- Eingänge:
  Gefahr: boolean;
  Quittung: boolean;
  Reset: boolean;

  -- Speicherabbild der Eingänge:
  Gefahr_PAE: boolean;
  Quittung_PAE: boolean;
  Reset_PAE: boolean;

  -- Ausgänge:
  Rot: boolean;
  Gelb: boolean;

  -- Speicherabbild der Ausgänge:
  Rot_PAA: boolean;
  Gelb_PAA: boolean;

  -- Status der Programmbearbeitung:
  Status: {EA_Phase, Schritt1, Schritt2, Schritt3};

ASSIGN
  init(Rot) := 0;
  init(Gelb) := 0;
  init(Rot_PAA) := 0;
  init(Gelb_PAA):= 0;
  init(Status) := EA_Phase;

  -- Zyklischer Betrieb:
  next(Status) := case
                    Status = EA_Phase : Schritt1;
                    Status = Schritt1 : Schritt2;
                    Status = Schritt2 : Schritt3;
                    Status = Schritt3 : EA_Phase;
                  esac;

  -- Einlesen des PAE:
  next(Gefahr_PAE) := case
                        Status = EA_Phase : Gefahr;
                        1 : Gefahr_PAE;
                      esac;
```

```
  next(Quittung_PAE) := case
                          Status = EA_Phase : Quittung;
                          1 : Quittung_PAE;
                        esac;

  next(Reset_PAE) := case
                       Status = EA_Phase : Reset;
                       1 : Reset_PAE;
                     esac;

  -- Abarbeitung des Programms:
  next(Rot_PAA) := case
                     Status = Schritt1 & Gefahr_PAE : 1;
                     Status = Schritt2 & Quittung_PAE : 0;
                     1 : Rot_PAA;
                   esac;

  next(Gelb_PAA) := case
                      Status = Schritt2 & Quittung_PAE : 1;
                      Status = Schritt3 & Reset_PAE & !Gefahr : 0;
                      1: Gelb_PAA;
                    esac;

  -- Auslesen des PAA:
  next(Rot) := case
                 Status = EA_Phase : Rot_PAA;
                 1 : Rot;
               esac;

  next(Gelb) := case
                  Status = EA_Phase : Gelb_PAA;
                  1 : Gelb;
                esac;

SPEC
AG((Gefahr & !Gelb) -> AF(Rot))
-- AG(!(Rot & Gelb))
```

2.6.4 Analyse von Echtzeit-Systemen mit Uppaal

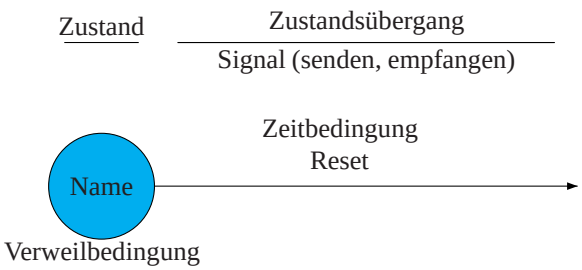
Uppaal Tool für die

- Spezifikation
- Simulation
- Verifikation

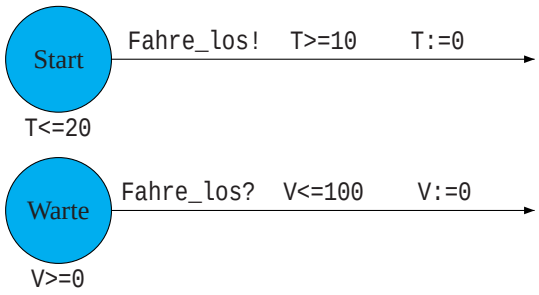
von Echtzeit-Systemen

Timed Automata Formalismus zur Spezifikation von Echtzeitsystemen (Endlicher Automat und Uhren)

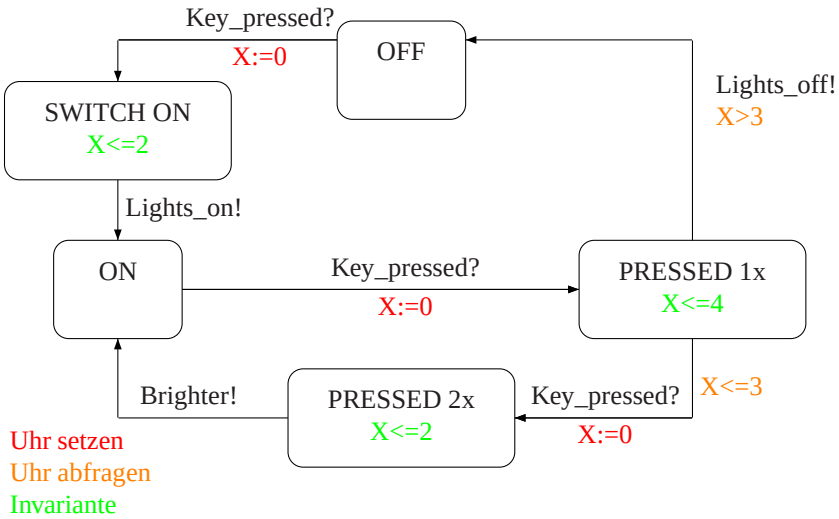
Modellierungssprache



Beispiel



Beispiel Timed Automata



Uppaal Modell

- Menge paralleler, kommunizierender Timed Automata
- Beliebig viele Uhren (alle gleich schnell)
- zusätzlich Variablen mit endlichem Wertebereich

Uppaal's Query Language

- Atomare Eigenschaften:

  `P.loc` Prozeß P in Location I
  `X == n` Uhr X gleich n (Integerwert)
  `X <= n`, `X < n` entsprechend

- Kombinatoren: `e1 and e2`, `not e1`, `e1 or e2`, `e1 implies e2`
- "Safety Properties":

  `A[] e` "always on all paths e"
  `E <> e` "eventually on some path e"

Mutual Exclusion

- In keinem Zustand sollen sowohl $P1$wie $P2$ in ihrer Critical Section sein.

↕

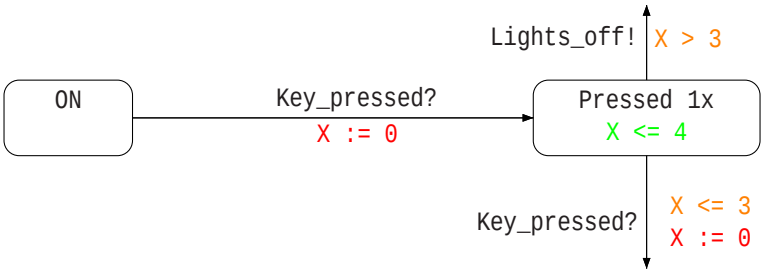
- In allen Zuständen gilt, dass nicht gleichzeitig $P1$ und $P2$ in ihrer Critical Section sind.

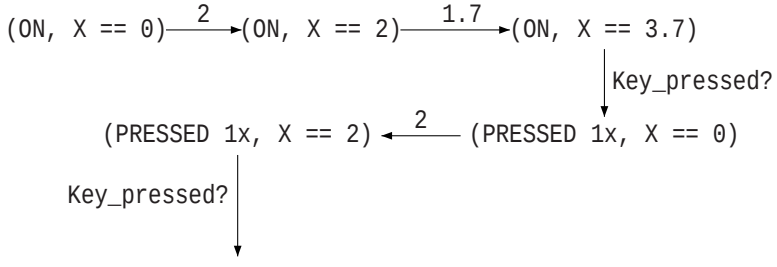
↕

- `A[] not (P1.CS and P2.CS)`

Model Checking Verfahren zum Überprüfen von Eigenschaften einer Spezifikation (z.B. Deadlocks, Mutual Exclusion, toter Code, im allgemeinen Korrektheit)
check if the system is a **model** of the formula

Ausführung





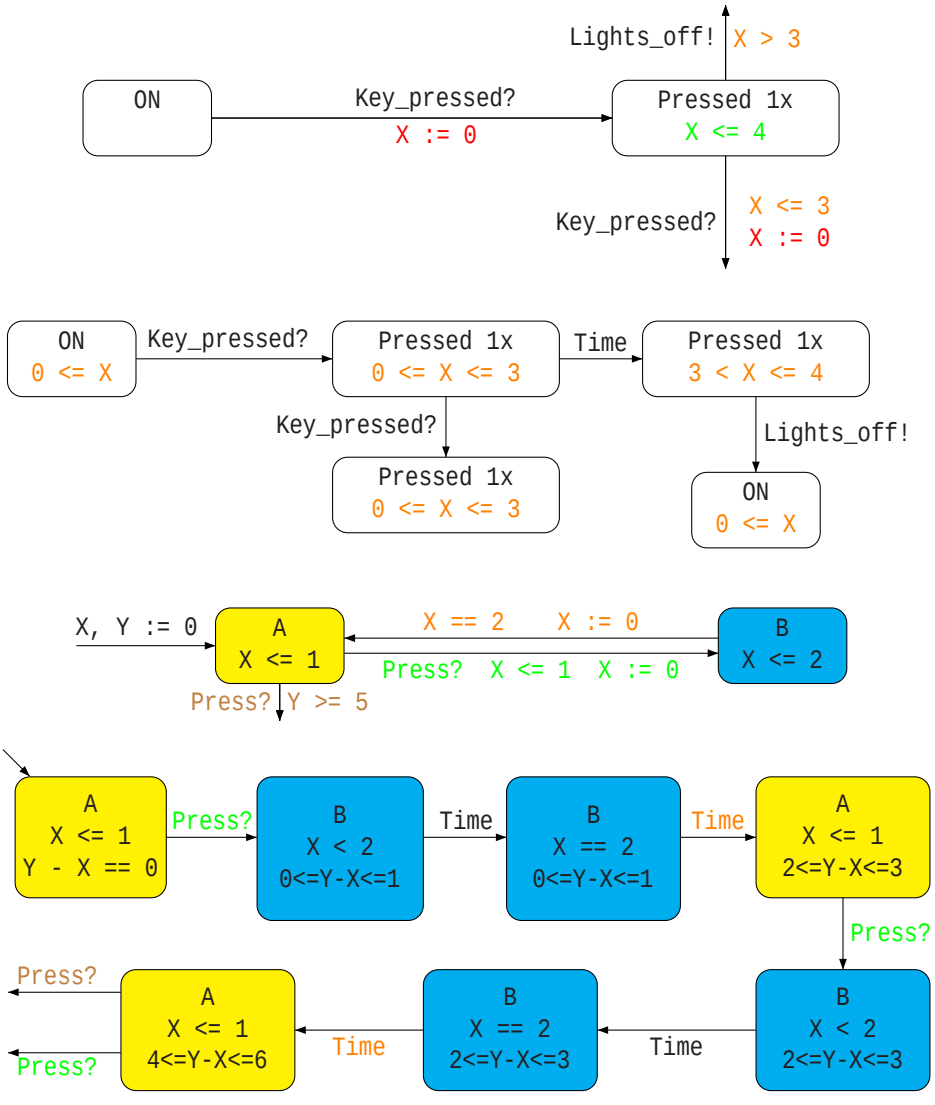
Zustandsraum

- Zustand eines Timed Automata = (Location, Timer1, …, Timer$N$)
- Zustandsraum = Locations × Reals × … × Reals
- Problem: unendlicher Zustandsraum (sogar überabzählbar)

Methoden

- Model Checking generell: Durchsuchen des gesamten Zustandsraums
- Symbolic Model Checking: Zusammenfassen ähnlicher Zustände zu Mengen
- Model Checking = Erreichbarkeits-Analyse

Symbolische Ausführung



Der Zug

- Wenn der Zug vom Sensor vor dem Bahnübergang erfaßt wird, wird ein Signal `Nähere_ mich` gesendet.
- Danach braucht der Zug noch 5 Sekunden bis er im Bereich des Bahnübergangs ist.
- Der Zug braucht 10 Sekunden, um den Bereich des Bahnübergangs zu durchqueren.

Die Schranke

- Die Schranke reagiert auf ein Steuersignal `Impuls`
- Je nach Position reagiert sie durch Heben oder Senken.
- Für Heben und Senken braucht die Schranke jeweils 2 Sekunden.

Die Steuereinheit

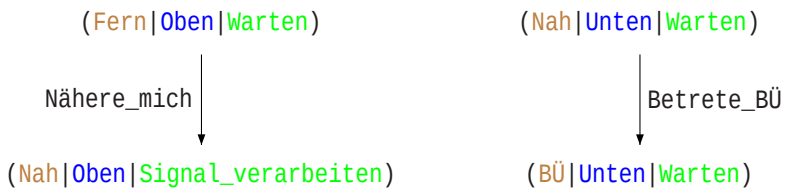
- Die Steuereinheit wartet auf das Signal `Nähere_ mich`
- Wenn sie das Signal `Nähere_ mich` erhält, schickt sie einen Impuls an die Schranke
- Die Steuereinheit braucht 2 Sekunden um das Signal `Nähere_ mich` zu verarbeiten und das Signal `Impuls` zu senden.

Die Sicherheitsbedingung

- Wenn ein Zug sich im Bereich des Bahnübergangs befindet, muß die Schranke unten sein.
- Zug@BÜ $\Longrightarrow$ Schranke@Unten

Parallele Komposition

Parallele Komposition mach aus vielen Graphen einen Graphen



Erreichbarkeitsanalyse

Gibt es im Graphen des Gesamtsystems eine vom Anfangszustand

(“Fern”|“Oben”|“Warten”)

erreichbaren Zustand

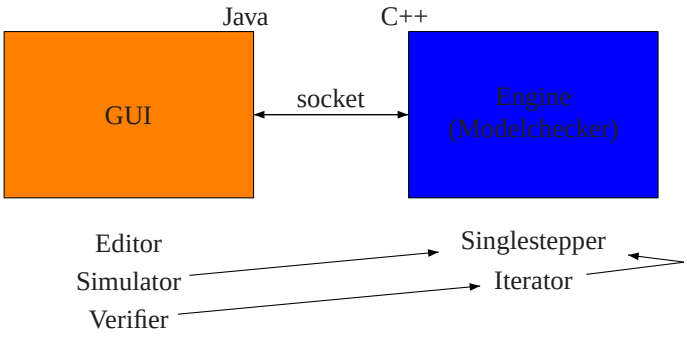
(Z|S|SE)

mit Z == “BÜ” und S! = “Unten”?

Wichtige Eigenschaften der Timed Automata

- Endliche symbolische Darstellung existiert
- Symbolische Darstellung erhält die Semantik
- Erreichbarkeit und Bisimulation entscheidbar

Software Architecture



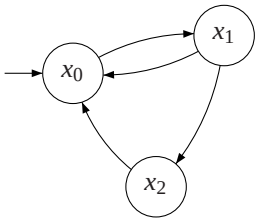
Runs on Linux, Windows, Solaris
Runs remotly

2.7 Hybride Systeme

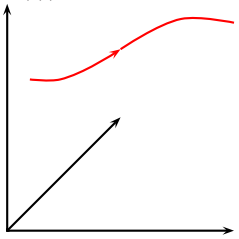
Hybrides System Modell für Systemdynamik, das diskrete und kontinuierliche Dynamikanteile enthält. (Systeme sind nicht hybrid, sondern werden hybrid modelliert.)

Zwei Ausgangspunkte und Richtungen:

Diskrete Systeme (Informatik)
$x_{k+1} = f(x_k, u_k)$



Kontinuierliche Systeme (Ingenieurwesen)
$\dot{x}(t) = f(x(t), u(t))$



hybride Systeme

erster Schritt:
Echzeitautomaten

kontinuierliche Ergänzung:
Uhren

$\dot{x}(t) = 1$ (überall, bis auf Reücksetzzeitpunkte)

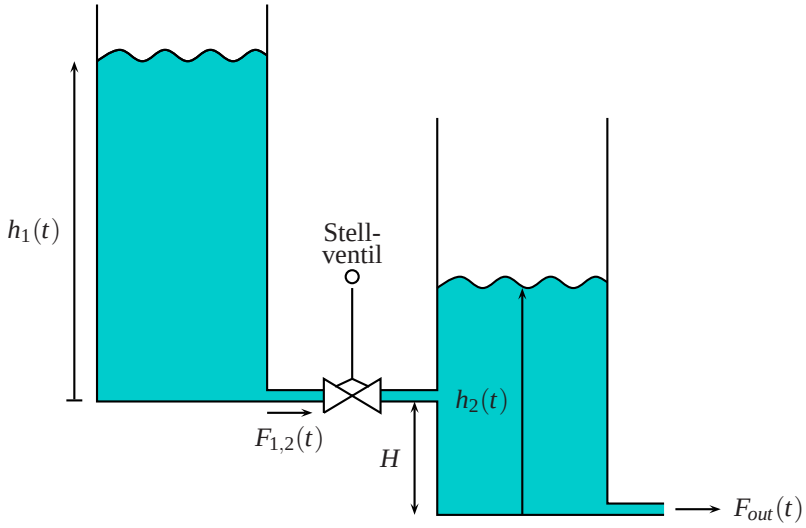
erster Schritt:
stückweise kontinuierliche (geschaltete) **nichtlineare Systeme**
diskrete Ergänzung:
Umschalten der rechten Seite der Zustands-DGL

$$\underline{\dot{x}}(t) = \begin{cases} f_1(\underline{x}, \underline{u}) & \text{if } \underline{x} \in \text{Reg}_1 \\ \vdots & \\ f_m(\underline{x}, \underline{u}) & \text{if } \underline{x} \in \text{Reg}_m \end{cases}$$

Reg_i = Region (Teilmenge) im Zustandsraum

Beispiel für stückweise kontinuierliches nichtlineares System:

Strecke:

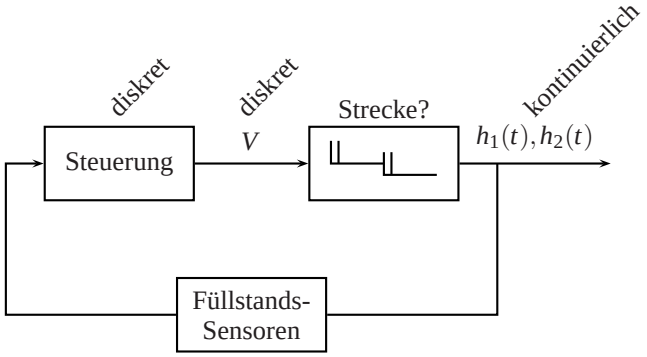


Meßgrößen: $h_1(t), h_2(t)$ (kontinuierlich)

Stellgröße: Ventilstellung $V \in \{\text{ganz\_ offen, halb\_ offen}\}$ (diskret)

Steuerung:

$$V = \begin{cases} \text{ganz\_ offen} & \text{wenn } h_1(t) > 0.8 \\ \text{halb\_ offen} & \text{sonst.} \end{cases}$$



$\Longrightarrow$ Gesamtsystem ist hybrid, da kontinuierliche[6] Strecke mit diskreter Steuerung verknüpft wird.

Aber: Strecke ist alleine schon hybrid!

Modell der Strecke:

$$\begin{aligned} \dot{h}_1(t) &= \frac{F_{in}(t) - F_{1,2}(t)}{A_1} \\ \dot{h}_2(t) &= \frac{F_{1,2}(t) - F_{out}(t)}{A_2} \end{aligned}$$

[6] "kontinuierlich/diskret" = "sinnvollerweise kontinuierlich/diskret modelliert"

A_1, A_2 = Bodenflächen der Tanks

$$F_{out}(t) = k_2 \cdot \sqrt{h_2} \quad \text{(Bernoulli: } F \sim \sqrt{2gh} = k \cdot \sqrt{h}\text{)}$$

$$F_{1,2} = \begin{cases} k_1 \cdot \sqrt{h_1} & \text{wenn } h_2 < H \\ k_1 \cdot \sqrt{h_1 - (h_2 - H)} & \text{wenn } h_2 \geq H \text{ und } h_1 \geq h_2 - H \\ 0 & \text{sonst (Rückschlagklappe)} \end{cases}$$

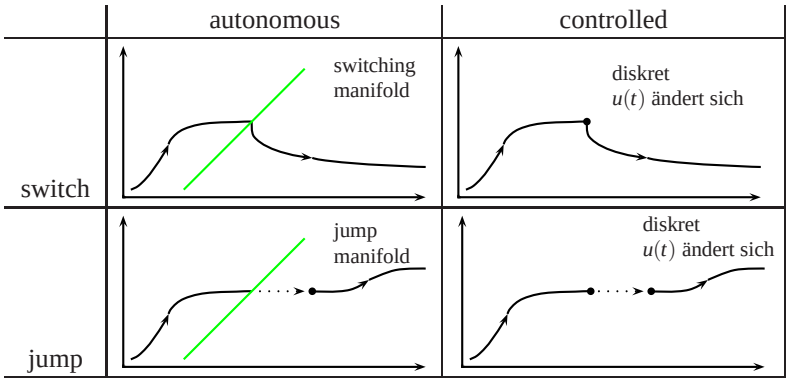
(stückweise kontinuierliches (geschaltetes) nichtlineares System)
autonomes Schalten(autonomous switching)

mit Eingangsgröße:

$$k_1 = \begin{cases} k_{1,GO} & \text{wenn } V = \text{ganz\_ offen} \\ k_{1,HO} & \text{wenn } V = \text{halb\_ offen} \end{cases}$$

gesteuertes Schalten (controlled switching)

Taxonomie nach **Branicky**:



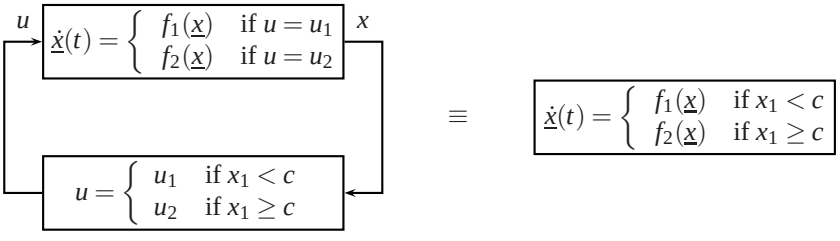
Unterschied autonomous/controlled?

autonomous: Schalten/Springen immer an gleicher Stelle

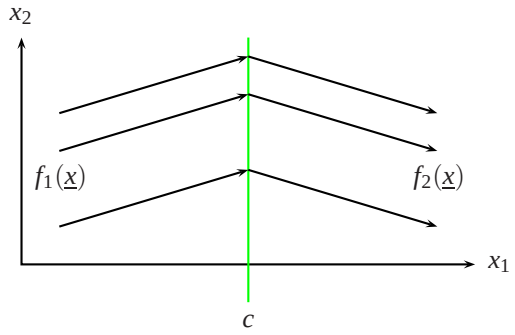
controlled: wird von außen hervorgerufen, kann an verschiedenen Stellen sein.

Unterschied verschwindet bei direkter Rückkopplung:

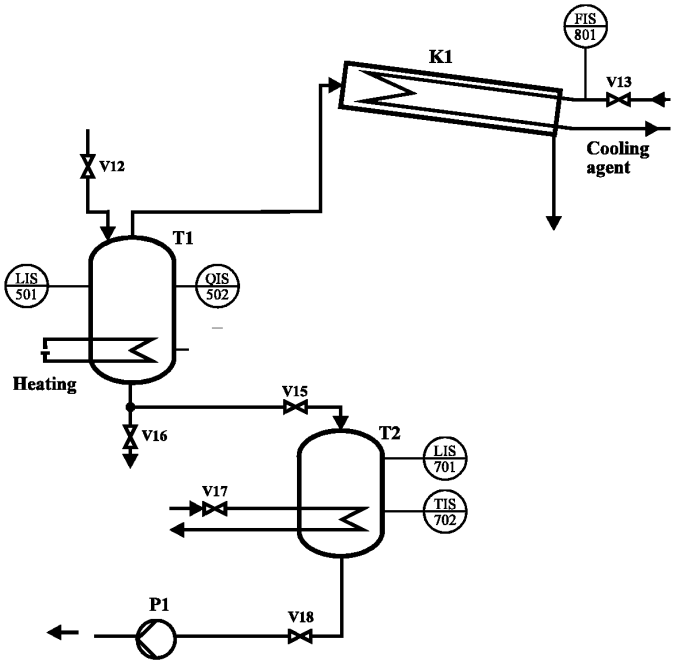
Beispiel:



Resultierendes Verhalten:



Beispiel für "Hybridisierung" diskreter Systeme (= "andere Richtung")



Vorgesehener Produktionsablauf:

- Füllen von Behälter T1
- Aufheizen
- Verdampfen bis gewünschte Konzentration erreicht ist
- Heizung ausschalten und T1 leeren
- Nachbearbeitungsschritt in T2

Notabschaltung bei Kühlungsausfall

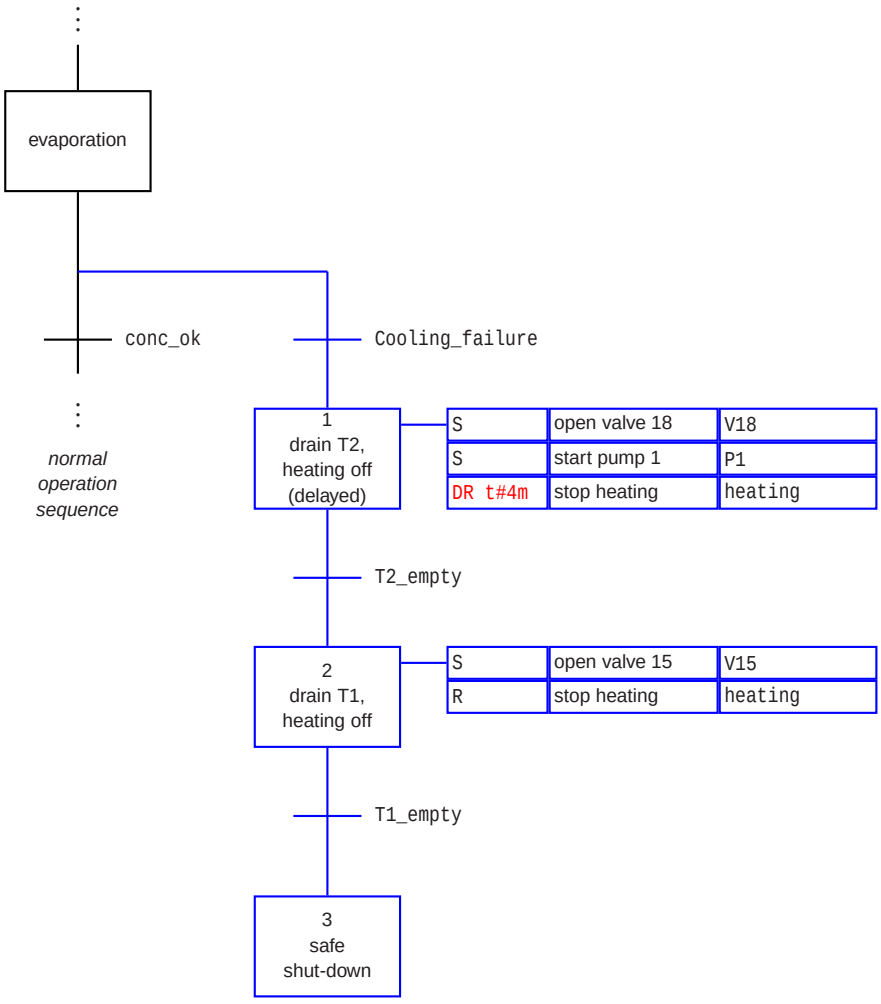
Randbedingungen:

1. Bei fortgesetzter Wärmezufuhr steigen Temperatur und Druck (geschlossenes System).
2. Bei Abkühlen unter einen Grenzwert beginnt das Material im Verdampfer zu kristallisieren.

→ **Gegenläufige Anforderungen:**

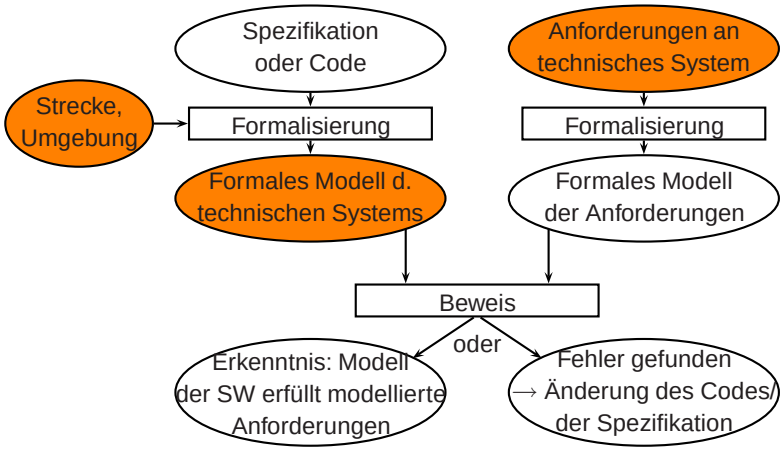
- Heizung muß **früh genug** abgeschaltet werden, um gefährlichen Druckanstieg zu vermeiden.
- Heizung muß **lange genug** eingeschaltet bleiben, um Kristallisierung zu vermeiden.

Vorschlag für Steuerungsprogramm[7]

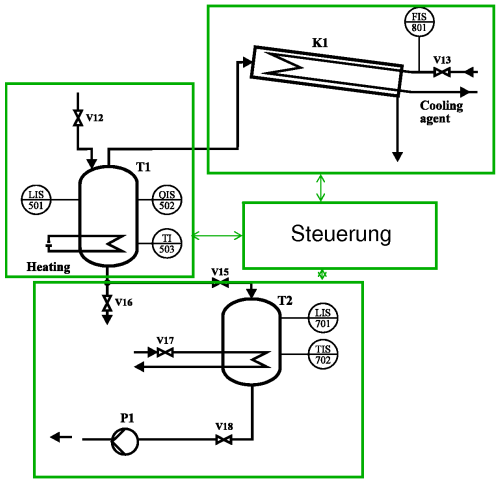


[7] Sequential Function Chart (SFC) nach IEC 1131-3

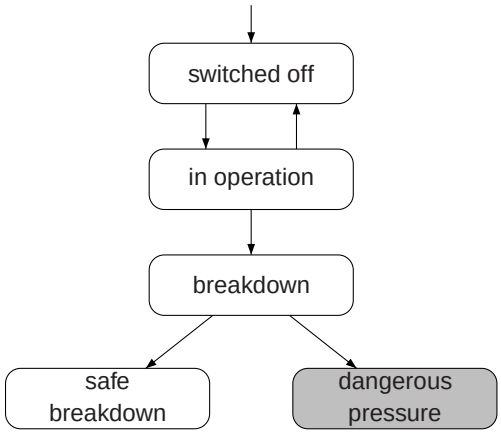
Formale Verifikation von Automatisierungssystemen



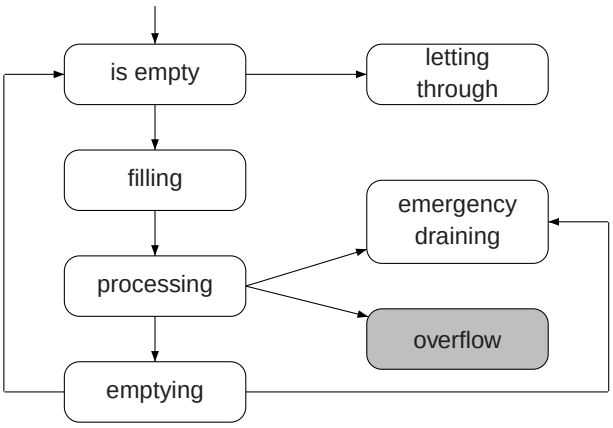
Formalisierung



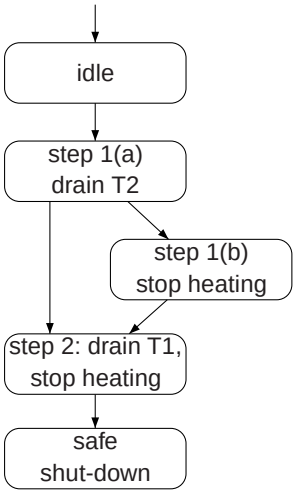
Diskretes Modell des Kondensators K1



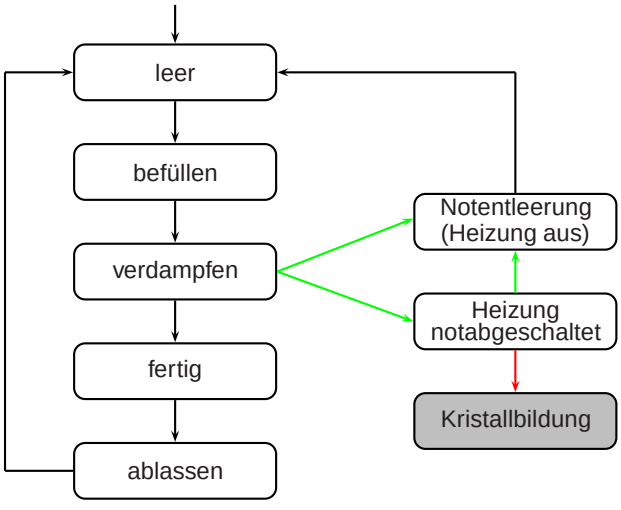
Diskretes Modell von Behälter T2



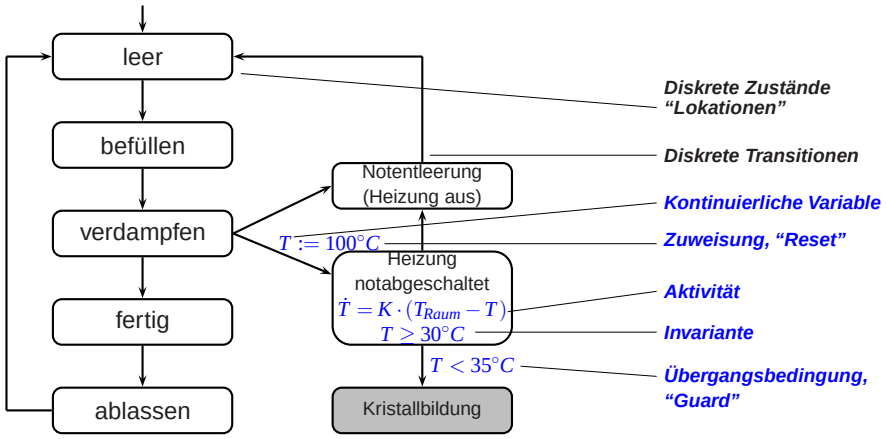
Diskretes Modell der Steuerung



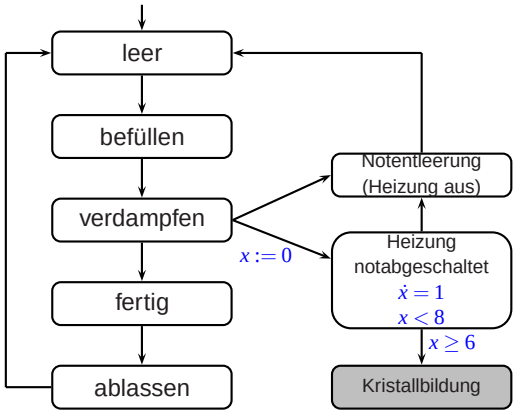
Diskretes Modell für Tank 1



Hybrider Automat[8]



Echtzeitautomat[9]



Erreichbarkeitsanalyse

Gegeben:

- ein hybrider Automat (HA) mit einem hybriden Anfangszustand
  (= diskrete Lokation + kontinuierliche Region)
- ein hybrider Zielzustand.

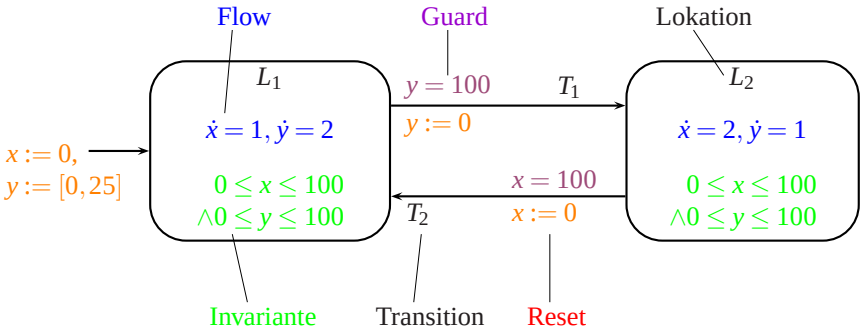
Frage Existiert eine Trajektorie (ein "Lauf" des HA), die vom Anfangs– zum Zielzustand führt?

Werkzeuge:

- **Kronos** (Verimag, Grenoble), **Uppaal** (Uppsala & Aalborg):
  Echtzeitautomaten ($\dot{x} = 1$)
- **ältere Version von Uppaal:** Integrator–Automaten ($\dot{x} \in \{0, 1\}$)
- **Hytech** (Berkeley): "Lineare" hybride Automaten ($\dot{x} \in [\dot{x}_{min}, \dot{x}_{max}]$)

[8] Henzinger, Alur, Sifakis, 1992
[9] Alur, Dill, 1990

Beispiel

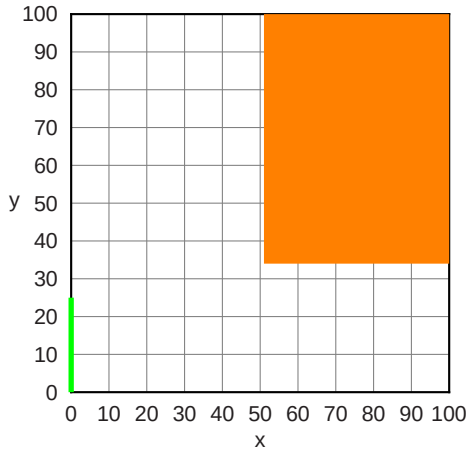


Frage Gibt es eine Trajektorie von

$$(\text{L1}, x = 0 \wedge y \in [0,25])$$

nach

$$(*, x \geq 51 \wedge y \geq 34)?$$

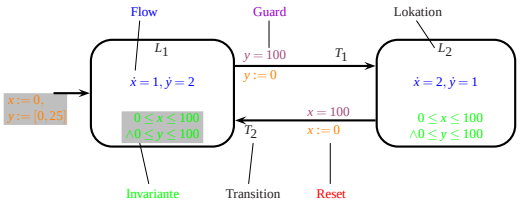


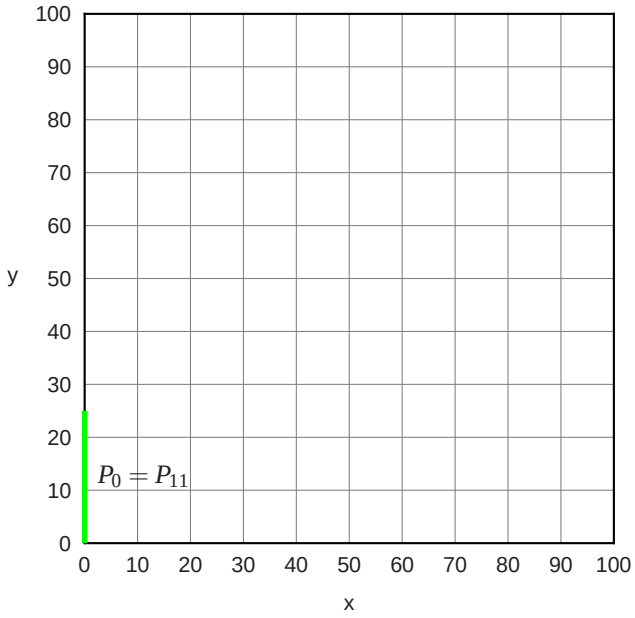
Erreichbarkeitsalgorithmus

0. Initialisiere den HA: $L :=$ Anfangslokation, $P :=$ Anfangspolyeder
1. Schneide P mit der Invariante von L.
2. Lasse P entsprechend der Aktivität von L wachsen.
3. Schneide P mit der Invariante von L.
4. Stop, falls L schon mit P besucht wurde. Falls nicht:

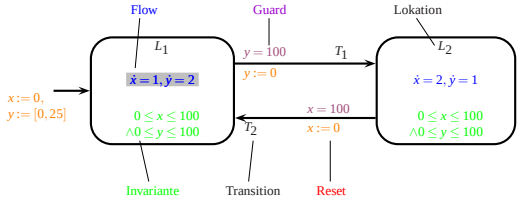
   Für alle Transitionen T von L:
5. Schneide P mit dem Guard von T. Falls leer, nehme nächste Transition.
6. Falls nicht, setze P entsprechend Reset–Ausdruck zurück.
7. Setze $L :=$ Ziel–Lokation von T, gehe zu Schritt 1.

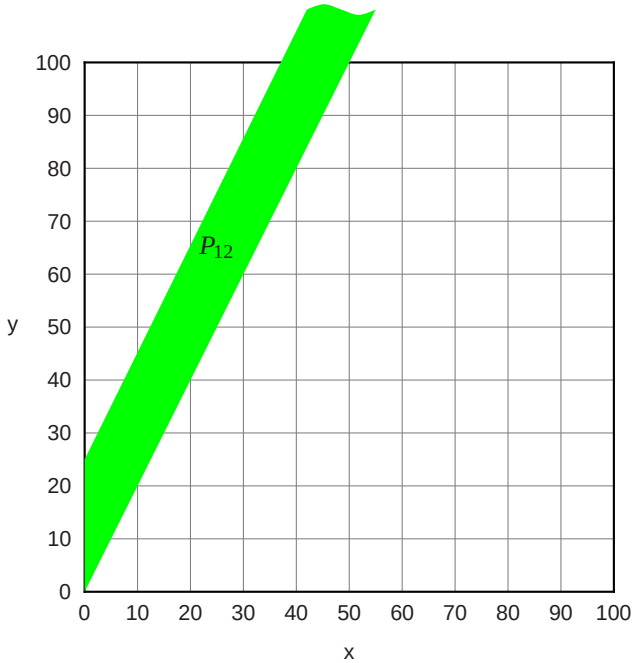
0. Initialisiere den HA:
   $L :=$ Anfangslokation, $P :=$ Anfangspolyeder
1. Schneide P mit der Invariante von L.



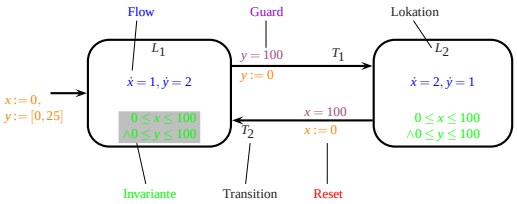


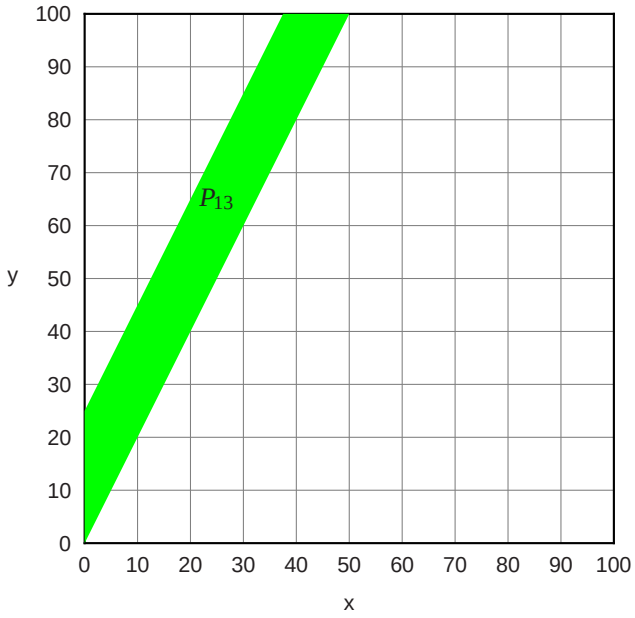
2. Lasse P entsprechend der Aktivität von L wachsen.



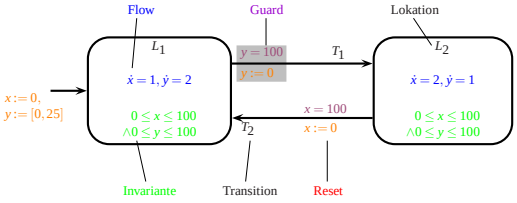


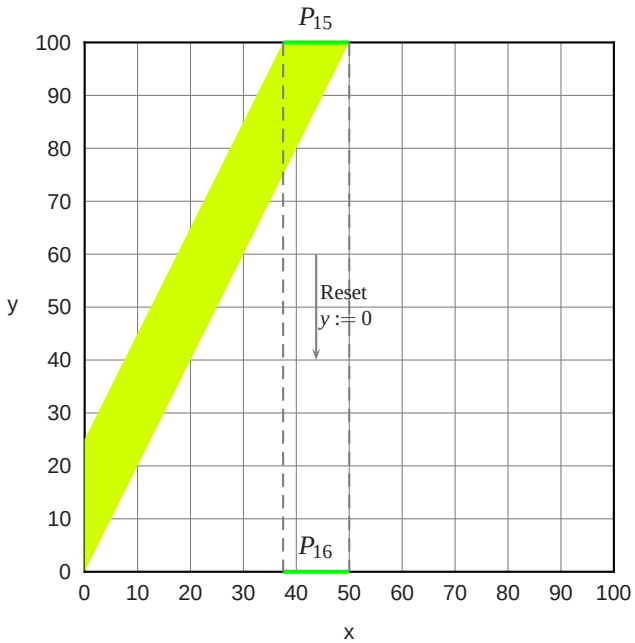
3. Schneide P mit der Invariante von L.



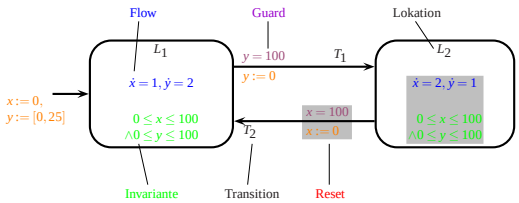


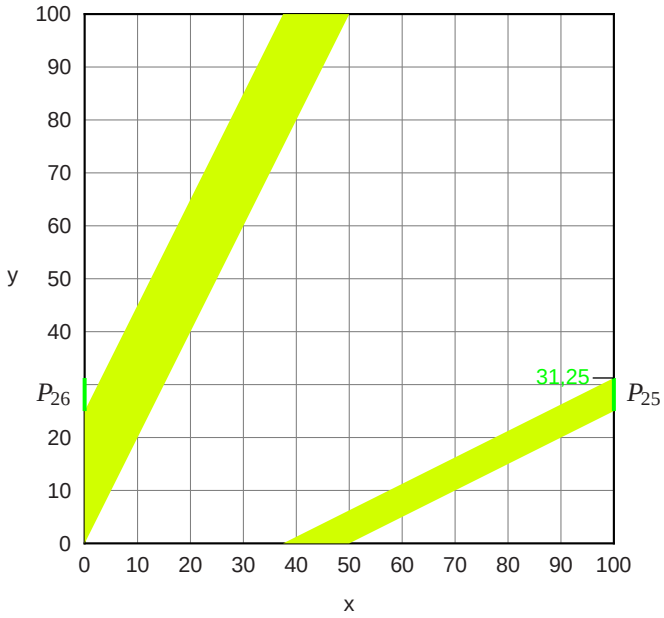
5. Schneide P mit dem Guard von T. Falls leer, nehme nächste Transition.
6. Falls nicht, setze P entsprechend Reset–Ausdruck zurück.

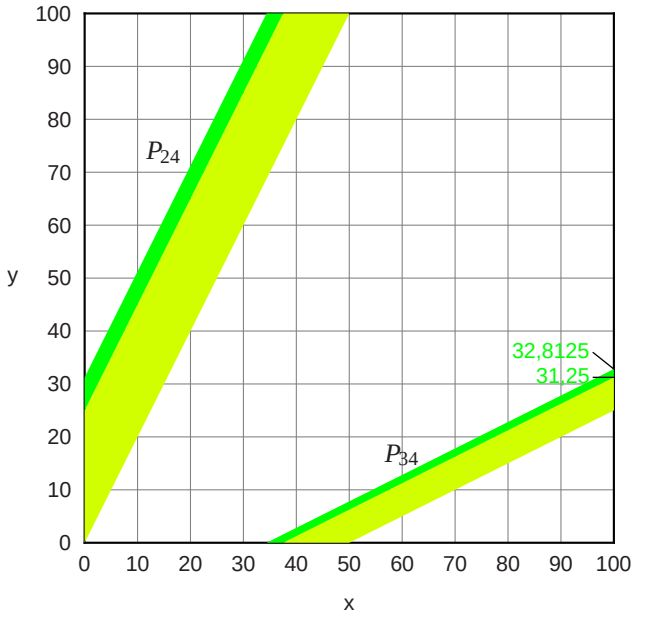


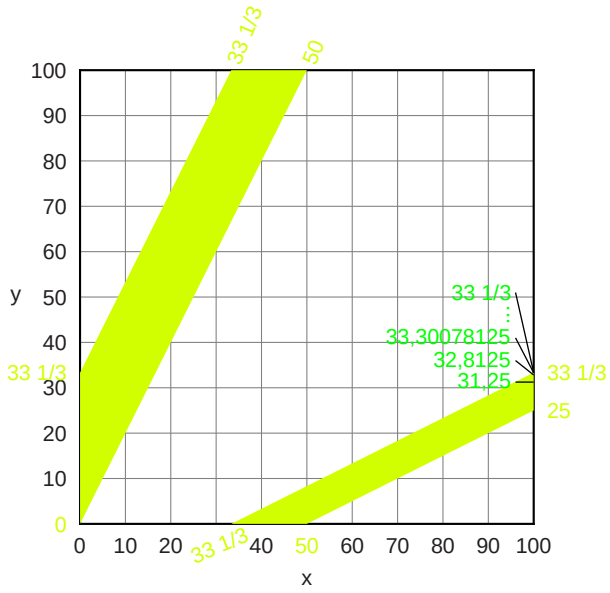


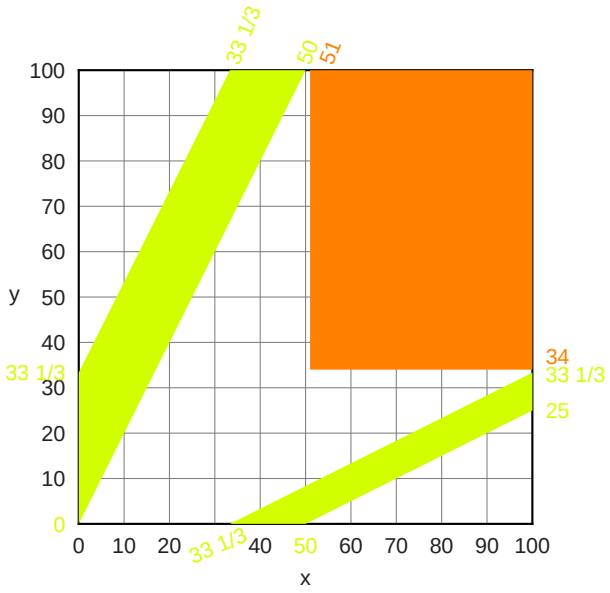
2. Lasse P entsprechend der Aktivität von L wachsen.
3. Schneide P mit der Invariante von L.
5. Schneide P mit dem Guard von T. Falls leer, nehme nächste Transition.
6. Falls nicht, setze P entsprechend Reset–Ausdruck zurück.











Erreichbarkeitsanalyse von hybriden Automaten: Grenzen

Theoretisch:

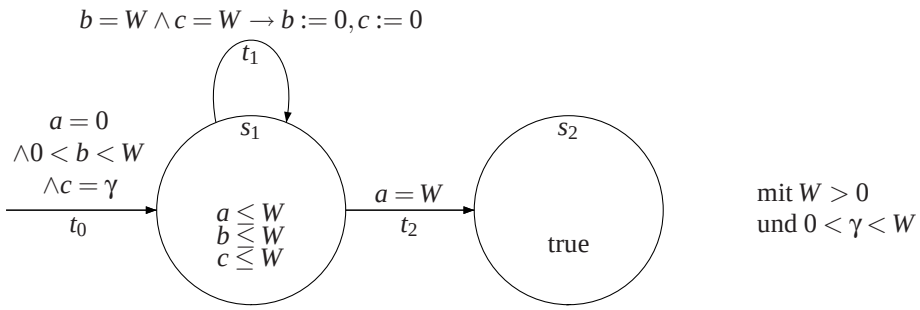
- Erreichbarkeit ist unentscheidbar für allgemeine hybride Automaten.
- Entscheidbarkeit gilt nur für sehr eingeschränkte Klassen von hybriden Automaten (Alur et al., 1995).
- Für rektanguläre Automaten ist die Erreichbarkeit partiell entscheidbar.

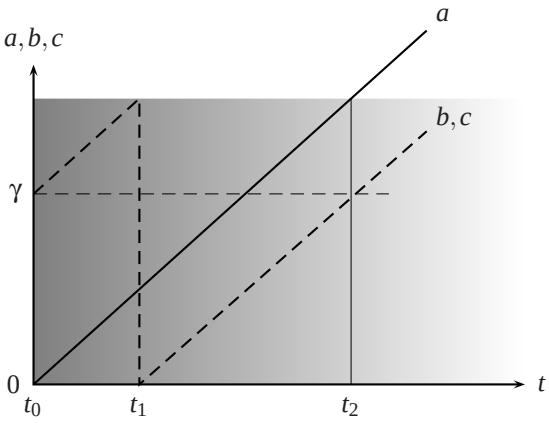
Praktisch:

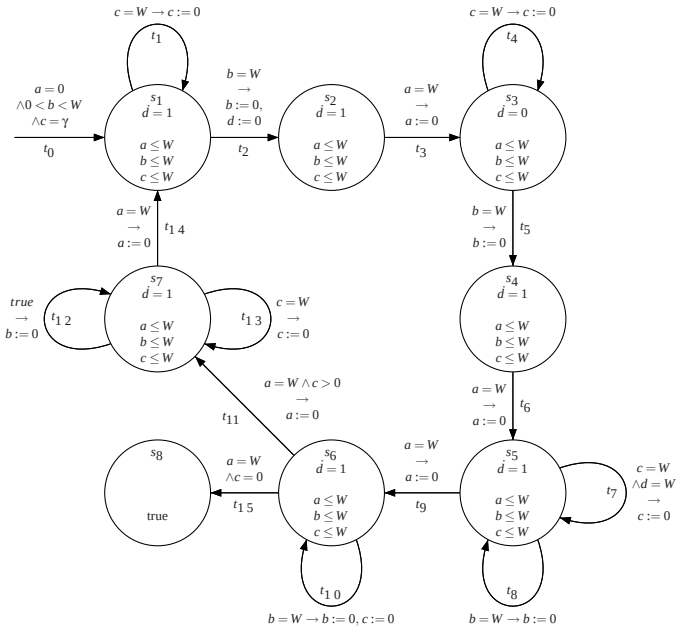
- Hytech verwendet ganzzahlige Arithmetik (kein Runden).
  $\Longrightarrow$ Integer-Überläufe nach wenigen Iterationen.
- Rechenzeit wächst exponentiell mit der Anzahl kontinuierlicher Variablen (zusätzlich zur diskreten Zustandsraumexplosion).

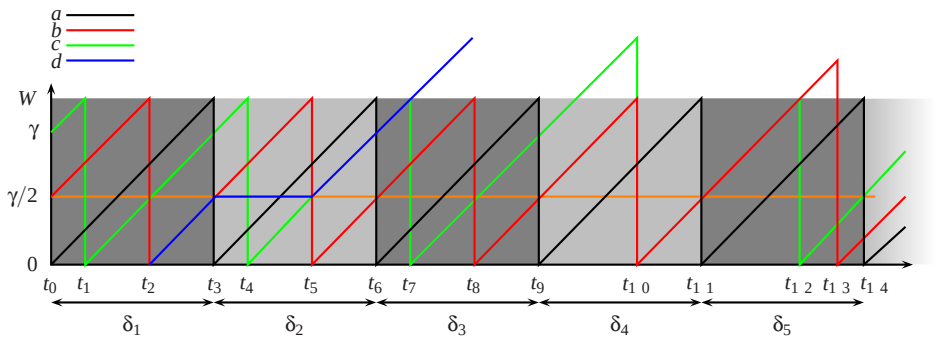
Abhilfe: **Abstraktion**

Beispiel für Nichttermination der Erreichbarkeit wegen Stoppuhren



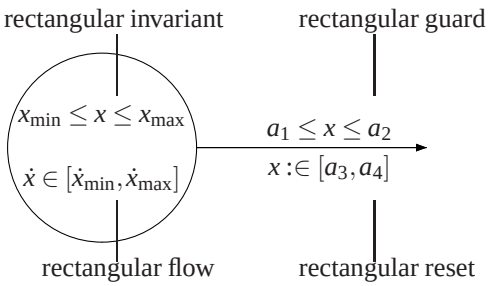






- δ_1: Synchronisiert b und d.
- δ_2: $c(t_0) = 2b(t_0) \Longrightarrow d$ synchronisiert mit c.
- δ_3: Stellt sicher, daß $c(t_6) = d(t_6)$, welches $c(t_3) = 2b(t_3)$ benötigt.
- δ_4: Synchronisiert b und $d \Longrightarrow c(t_{11}) = b(t_9)$. Von t_0 nach t_{11}: $c := c/2$.
- δ_5: Wähle neuen Wert für b. (Es muß $b(t_{14}) = b(t_{11})/2$ gelten, um Deadlock zu vermeiden)

Rektangulärer Automat



Grundidee bei der approximativen Analyse einfacher rektangulärer Automaten

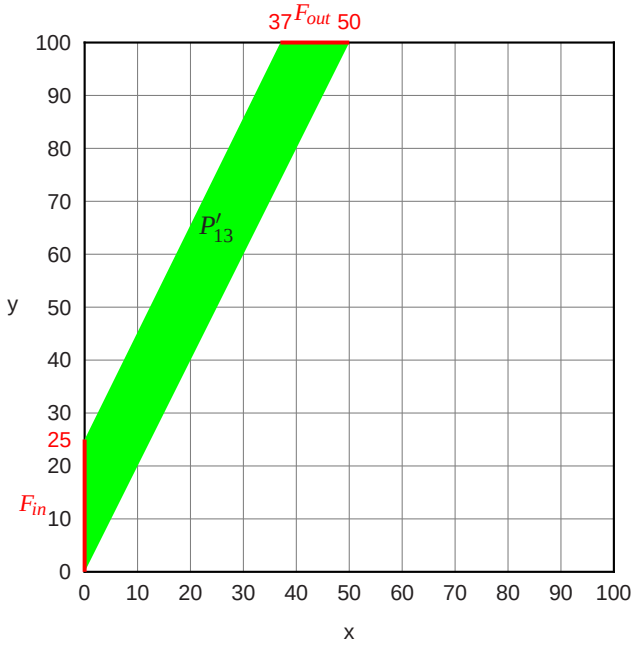
- Implizite Darstellung der erreichbaren Regionen R durch ihre Eingangs– und Ausgangs–Faces F_{in} bzw. F_{out}. Es gilt: $R = \textbf{KonvexeHülle}(F_{in}, F_{out})$

Berechnungsvorschrift (Beispiel):

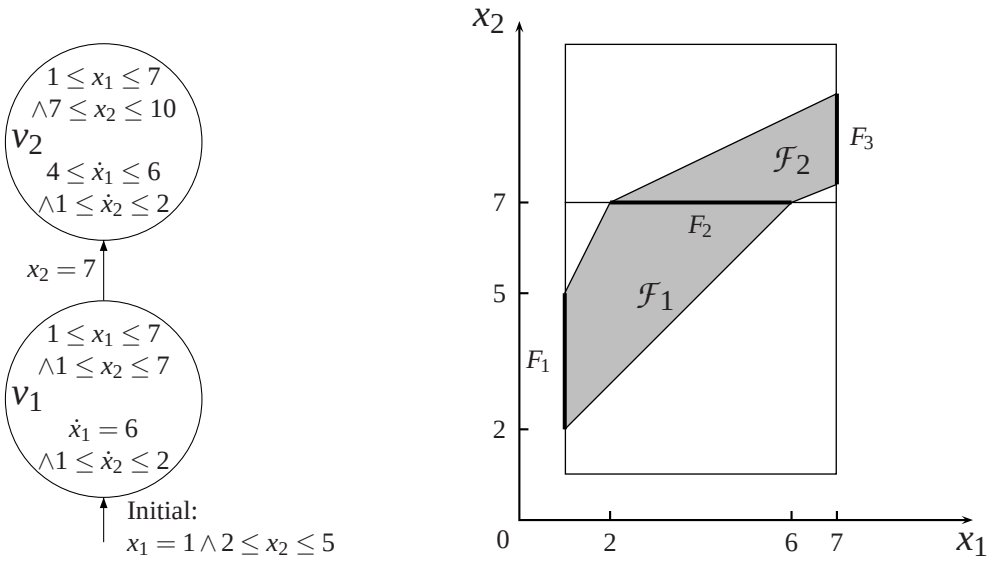
1. Bestimme das Zeitintervall ΔT, in dem die Grenze $y = 100$ von Punkten aus F_{in} erreicht verden kann: $\dot{y} = 2 \Longrightarrow \Delta T = [37.5, 50]$

2. Runde konservativ auf gewünschte Auflösung (= “Dehnen” von ΔT): $\Delta T' = [37, 50]$

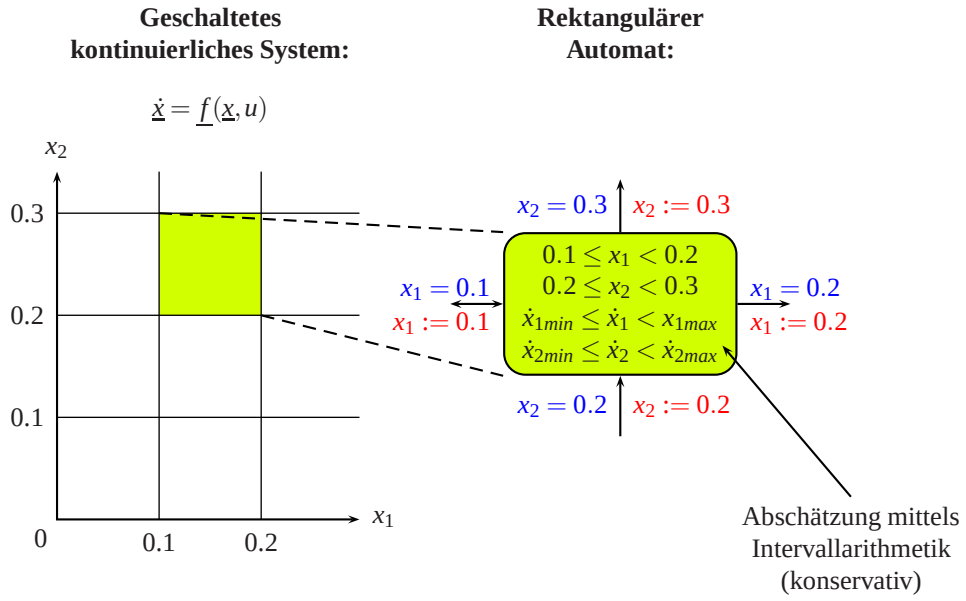
3. Berechne für die andere Dimensionen die ΔT mögliche Auslenkung: $\dot{x} = 1 \Longrightarrow F_{out,x} = [37, 50]$



Approximative Erreichbarkeitsanalyse mit “Faces”

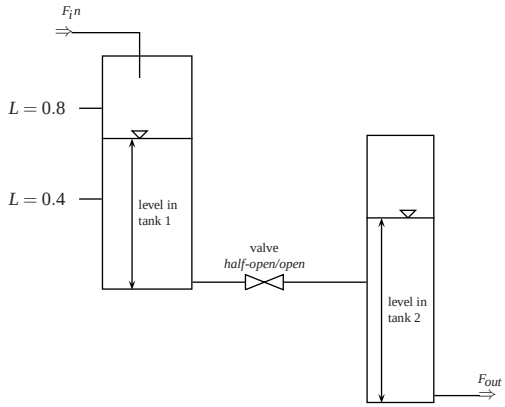


Abstratktion von geschalteten kontinuierlichen Systemen durch rektanguläre Automaten



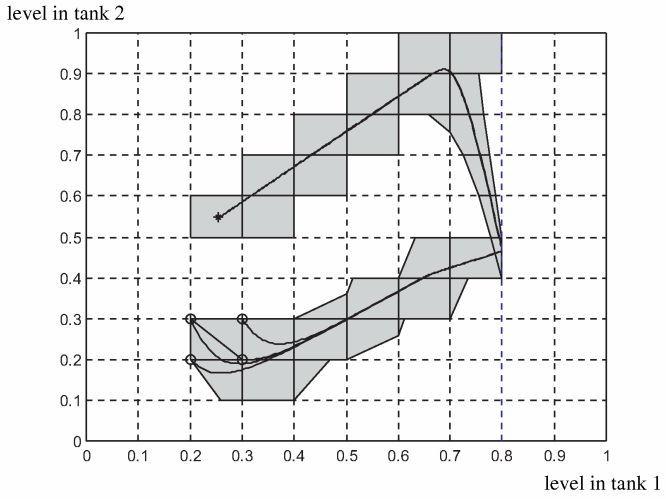
Doppel-Tank-Beispiel

- konstanter Eingangsfluß F_{in}
- zu Beginn ist das Ventil halb-offen $\Longrightarrow$ der Pegel in Tank 1 steigt
- dann wird das Ventil komplett geöffnet $\Longrightarrow$ der Pegel in Tank 1 sinkt und
  der Pegel in Tank 2 kann steigen
  bis ein Gleichgewicht $F_{in} = F_{out}$ erreicht ist.

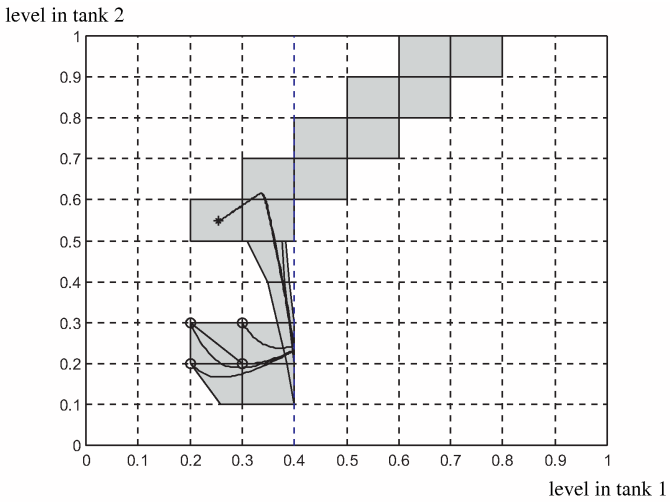


Prüfungsaufgabe:
Wenn in Tank 1 ein bestimmter **Pegel** L **erreicht** ist, **öffnet** der **Kontroller** das Ventil vollständig(*open*), um einen Überlaufen des Tanks 1 zu vermeiden.
Kann das zu einem Überlauf in Tank 2 führen?

Analyse-Ergebnis für $L = 0.8$:



(Unerwartetes) Analyse-Ergebnis für $L = 0.4$:



Index