

Vorlesung „Methoden des Software Engineering“

Block C „Formale Methoden“ Reaktive Systeme

Martin Wirsing

Einheit C.4, 14.12.2004

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Ziele

- Transitionssysteme zur Beschreibung reaktiver Systeme einsetzen lernen
- Den Begriff der "Fairness" verstehen
- Sicherheits- und Lebendigkeitseigenschaften von Transitionssystemen verstehen lernen
- Temporale Logik zur Spezifikation von Eigenschaften von Transitionssystemen einsetzen lernen

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Reaktive Systeme

Bisher standen folgende Aspekte der Beschreibung von Systemen im Vordergrund:

[Datenstrukturen und Funktionen:] Operationen auf Datenstrukturen durch Angabe des Ein-/Ausgabeverhaltens

[zustandsbasierte (interaktive) Systeme:] Wirkungen von Operationen auf Systemzustand

Reaktive Systeme

- kontinuierliche Interaktion mit Umgebung
- Terminierung unerwünscht bzw. unwesentlich

Beispiele: Prozesssteuerungen, eingebettete Systeme, Protokolle

Formal werden reaktive Systeme durch **Transitionssysteme** modelliert.

Dabei interessiert sich man meist mehr für den Kontrollfluss als für den Berechnungsaspekt.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Transitionssysteme und Abläufe

Wiederholung und Erweiterung der Begriffe:

Definition:

Ein markiertes Transitionssystem $\Gamma = (Z, I, \mathcal{A}, \delta)$ mit Anfangszuständen ist gegeben durch

- eine Menge Z von Zuständen,
- eine nichtleere Menge $I \subseteq Z$ von Anfangszuständen,
- eine Menge \mathcal{A} von Aktionen (bzw. Aktionsnamen) und
- eine Zustandsübergangsrelation $\delta \subseteq Z \times \mathcal{A} \times Z$.

Die Aktion A heißt **ausführbar** ("enabled") im Zustand s , wenn ein t existiert mit $(s, A, t) \in \delta$.

Im folgenden setzen wir voraus, dass in jedem Zustand $s \in Z$ mindestens eine Aktion $A \in \mathcal{A}$ ausführbar ist (d.h. δ ist total).

Ein **Ablauf** von Γ ist eine unendliche Folge $\sigma = s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$, so dass gilt:

- $s_0 \in I$ ist ein Anfangszustand von Γ und
- für alle $i \geq 0$ ist $(s_i, A_i, s_{i+1}) \in \delta$.

Ein **endlicher Ablauf** von Γ ist ein endlicher Präfix eines Ablaufs von Γ .

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

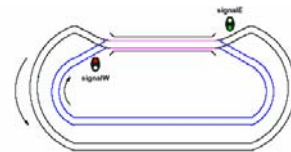
Transitionssysteme

Bemerkungen

- Transitionssysteme haben eine ähnliche Struktur wie endliche Automaten, aber keine Endzustände.
- Definitionen in der Literatur uneinheitlich, z.B.:
 - ein Anfangszustand $s_0 \in Z$ statt Menge $I \subseteq Z$
 - Aktionen oft implizit gelassen
 - Totalität von δ nicht gefordert, gelegentlich auch endliche Abläufe erlaubt
 - explizite "Stotteraktion" τ mit $(s, \tau, t) \in \delta$ gdw. $s = t$
- Die Zustandsmenge Z darf unendlich (sogar überabzählbar) sein.
- Zustandsübergänge modelliert als "atomare" Aktionen ohne zeitliche Dauer.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Beispiel Eisenbahnsteuerung



Zustand der Steuerung beinhaltet Informationen über:

- Signalstellung
- Position, Geschwindigkeit und Beschleunigung der Züge

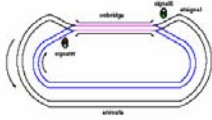
Zustandsübergänge:

- Erfassen neuer Zugdaten durch Sensoren
- Schalten der Signale

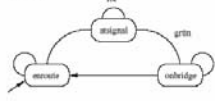
Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Beispiel Eisenbahnsteuerung

Abstraktere Modellierung führt zu endlichem Zustandsraum, z.B. Einteilung der Strecke in Abschnitte



Zustandübergänge
Züge



Nichtdeterminismus wegen abstrakter Modellierung

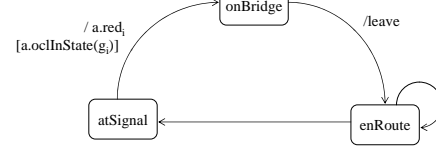
Signale (kombinatorische Schaltung)

trainW	trainE	signalW	signalE
enroute	atsignal	rot	grün
atsignal	enroute	grün	rot
atsignal	atsignal	?	?
sonst		rot	rot

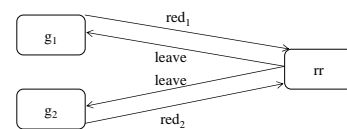
Beispiel Eisenbahnsteuerung: Statechart

Modellierung mit globalen Variablen g1,g2

Train ti, i=1,2



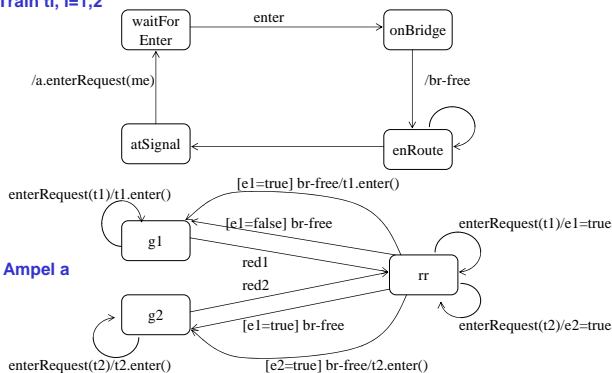
Ampel a



Beispiel Eisenbahnsteuerung: Statechart

Train ti, i=1,2

Modellierung ohne globale Variablen



Beispiel Eisenbahnsteuerung

Interessante Eigenschaften:

- Ist sichergestellt, dass sich immer nur höchstens ein Zug auf der Brücke befindet?
- Verlässt jeder Zug die Brücke wieder?
- Kann jeder wartende Zug irgendwann auf die Brücke?
- Schaltet die Ampel irgendwann auf Grün, wenn ein Zug wartet?

Fairnessbedingungen

Transitionssysteme

- Transitionssysteme erlauben häufig Abläufe, die im modellierten System nicht vorkommen können.
- **Typische Ursachen unfairer Abläufe:**
 - Abstraktion bei Modellierung führt zu Nichtdeterminismus im Modell vgl. Beispiel Eisenbahn: Transitionssystem erlaubt, dass
 - ein Zug immer auf der Brücke bleibt oder dass
 - ein Zug nie auf die Brücke gelassen wird
- **Fairnessbedingungen schränken die erlaubten Abläufe ein, indem „unfaire“ Abläufe ausgeschlossen werden.**

Typische Fairnessbedingungen

Beispiel Eisenbahn

- Wartet ein Zug in einem Ablauf immer wieder am Signal, so fährt er auch immer wieder auf die Brücke.
- Ein Zug, der ständig von der Brücke herunterfahren kann, fährt auch irgendwann hinunter.

Schwache und starke Fairness

Fairnessbedingungen fordern intuitiv, dass Aktionen, die "genügend häufig" ausführbar sind, irgendwann ausgeführt werden. Wir betrachten zwei Fairnessbegriffe:

schwache Fairness (weak fairness, justice): Ein Ablauf $s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$ heißt schwach fair bezüglich der Aktion A , falls gilt:
Existiert ein $n \in \mathbb{N}$, so dass A in allen Zuständen s_m mit $m \geq n$ ausführbar ist, so ist $A_i = A$ für unendlich viele $i \in \mathbb{N}$.
Äquivalente Formulierung: Wird A nur endlich oft ausgeführt, so ist A im Ablauf unendlich oft nicht ausführbar.

starke Fairness (strong fairness, compassion): Ein Ablauf $s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$ heißt stark fair bezüglich der Aktion A , falls gilt:
Existieren unendlich viele $m \in \mathbb{N}$, so dass A im Zustand s_m ausführbar ist, so ist $A_i = A$ für unendlich viele $i \in \mathbb{N}$.
Äquivalente Formulierung: Wird A nur endlich oft ausgeführt, so ist A im Ablauf nur endlich oft ausführbar.

Bemerkung: Starke Fairness impliziert schwache Fairness.
Ist ein Ablauf stark fair bezüglich A , so ist er auch schwach fair bezüglich A .

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Transitionssystem mit Fairness

Definition:

Ein Transitionssystem mit Fairness (fair transition system, FTS) $\Gamma_f = (Z, I, \mathcal{A}, \delta, W, S)$ erweitert ein Transitionssystem $\Gamma = (Z, I, \mathcal{A}, \delta)$ um Mengen $W, S \subseteq \mathcal{A}$.
Die Abläufe von Γ_f sind diejenigen Abläufe von Γ , die schwach fair sind bezüglich der Aktionen $A \in W$ und stark fair bezüglich der Aktionen $A \in S$.

Für die betrachteten Beispiele sind folgende Fairnessbedingungen sinnvoll:

Beispiel Eisenbahn: schwache Fairness für die Aktionen "Brücke verlassen" mit der Vorbedingung "Zug im Abschnitt onbridge" und der Nachbedingung "Zug im Abschnitt enroute".

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Implementierung von Fairnessbedingungen

Fairnessbedingungen können durch geeignete Scheduler implementiert werden. Für schwache Fairness genügt ein "round-robin-Scheduler".

Theorem:

Es sei $\Gamma_f = (Z, I, \mathcal{A}, \delta, \{B_0, \dots, B_{m-1}\}, \emptyset)$ ein FTS ohne starke Fairness, und $s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n$ sei ein endlicher Ablauf von Γ .

Dann ist jede Folge $s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n \xrightarrow{A_n} s_{n+1} \dots$ ein Ablauf von Γ_f , falls für alle $k \geq n$ die folgenden Bedingungen erfüllt sind:

- $A_k = B_{k \bmod m}$, falls die Aktion $B_{k \bmod m}$ im Zustand s_k ausführbar ist.
- $(s_k, A_k, s_{k+1}) \in \delta$.

Die Implementierung von starken Fairnessbedingungen benötigt einen Scheduler mit "dynamischen Prioritäten": Aktionen werden um so stärker priorisiert, je länger sie nicht ausgeführt wurden.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Implementierung von Fairnessbedingungen

Bemerkungen

- Ist Γ_f ein FTS, dessen zu Grunde liegendes Transitionssystem Γ ohne Fairnessbedingungen ausführbar ist (d.h. die Menge der Nachfolgerzustände zu jeder Aktion ist berechenbar), so können auch faire Abläufe von Γ_f systematisch erzeugt werden.
Dabei reicht es sogar, den Scheduler erst ab einem beliebigen Zeitpunkt nach Beginn des Ablaufs zu verwenden.
- Allerdings werden auf diese Weise nicht alle fairen Abläufe von Γ_f generiert (selbst endliche FTSe haben i.a. überabzählbar unendlich viele verschiedene faire Abläufe).
- Der prioritätsbasierte Scheduler kann auch für FTSe verwendet werden, die sowohl starke wie schwache Fairnessbedingungen enthalten, da starke Fairness schwache Fairness impliziert.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Eigenschaften von Abläufen

Bei der Analyse von Transitionssystemen interessiert man sich meist für Aussagen über ihre Abläufe, z.B.:

- Die Züge können zu keinem Zeitpunkt beide im Abschnitt onbridge sein.
- Wartet ein Zug vor dem Signal, so wird er zu einem späteren Zeitpunkt auf der Brücke sein.

Außerdem interessiert man sich gelegentlich für Aussagen über die Verzweigungsstruktur von Transitionssystemen wie:

- Aktionen A und B stehen in Konflikt oder sind unabhängig.
- Von jedem Zustand aus kann ein Anfangszustand erreicht werden.
- Zwei Prozesse können kooperieren, um einen dritten Prozess auszusperren.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Eigenschaften von Abläufen

Wir identifizieren Eigenschaften mit der Menge der Zustands-Aktions-Folgen, welche die Eigenschaft erfüllen.

Definition:

Sei $\Sigma = (Z, \mathcal{A})$ eine Menge von Zuständen und Aktionen.

Eine (Σ -)Eigenschaft ist eine Menge von Folgen $\sigma = s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$ mit $s_i \in Z$ und $A_i \in \mathcal{A}$.

Eine "Eigenschaft" P ist also eine Aussage, von der es sinnvoll ist zu fragen, ob sie auf einen Ablauf σ zutrifft ($\sigma \in P$) oder nicht.

Aussagen über die Existenz von Abläufen sind keine "Eigenschaften" im Sinne dieser Definition.

Beispiele:

- Menge der Abläufe eines Transitionssystems Γ
- Menge der Zustands-Aktions-Folgen, die stark fair sind bzgl. Aktion A
- Menge aller Folgen $s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$, so dass $s_n(y) = 1$ gilt für ein $n \in \mathbb{N}$

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

Sicherheits- und Lebendigkeitseigenschaften

Sicherheits- und Lebendigkeitseigenschaften

- grundlegende Klassen von Eigenschaften mit unterschiedlichen Beweisprinzipien
- Verallgemeinerung von partieller Korrektheit und Terminierung bei sequentiellen Programmen

Informelle Charakterisierung (Lampert 1980)

Sicherheitseigenschaft (safety property): *something bad never happens*

- Reihenfolge der empfangenen Nachrichten entspricht Reihenfolge beim Senden

Lebendigkeitseigenschaft (liveness property): *something good eventually happens*

- Züge können Brücke irgendwann befahren

Sicherheits- und Lebendigkeitseigenschaften

Formale Charakterisierung (Alpern, Schneider 1985)

Definition:

Sei $\Sigma = (Z, \mathcal{A})$ gegeben.

P ist eine (Σ -)Sicherheitseigenschaft genau dann, wenn für jede Folge

$\sigma = s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$ gilt:

$\sigma \in P$ gdw. für jeden Präfix $s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n$ von σ gibt es eine Folge $\tau = s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n \xrightarrow{B_n} t_{n+1} \xrightarrow{B_{n+1}} t_{n+2} \dots$ mit $\tau \in P$

P ist eine (Σ -)Lebendigkeitseigenschaft genau dann, wenn sich jede endliche Folge

$s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n$ zu einer Folge $s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n \xrightarrow{A_n} s_{n+1} \dots \in P$ ergänzen lässt.

Zusammenhang zu informeller Charakterisierung

- Eine Folge σ erfüllt eine Sicherheitseigenschaft P genau dann **nicht**, wenn es einen endlichen Präfix von σ gibt, der sich nicht zu einer Folge ergänzen lässt, die P erfüllt.
- Lebendigkeitseigenschaften schränken dagegen die Menge der endlichen Präfixe nicht ein.

Sicherheits- und Lebendigkeitseigenschaften

Vereinfachende Schreibweisen ($\Sigma = (Z, \mathcal{A})$ sei vorausgesetzt)

- Für eine Folge $\sigma = s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$ bezeichne $\sigma[..n]$ den Präfix $s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n$.
- Für Folgen $\rho = s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n$ und $\sigma = s_n \xrightarrow{A_n} s_{n+1} \xrightarrow{A_{n+1}} s_{n+2} \dots$ bezeichnet $\rho \circ \sigma$ die Konkatenation $s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n \xrightarrow{A_n} s_{n+1} \xrightarrow{A_{n+1}} s_{n+2} \dots$.
- Für eine endliche Folge $\rho = s_0 \xrightarrow{A_0} s_1 \dots \xrightarrow{A_{n-1}} s_n$ schreiben wir $\rho \in P$, falls $\rho = \sigma[..n]$ gilt für ein $\sigma \in P$ und ein $n \in \mathbb{N}$.
- Σ^* und Σ^ω bezeichnen die Menge aller endlichen bzw. unendlichen Folgen von Zuständen und Aktionen.

Damit gilt:

P ist Sicherheitseigenschaft genau dann, wenn für alle Folgen $\sigma \in \Sigma^\omega$ gilt:

Falls $\sigma[..n] \in P$ für alle $n \in \mathbb{N}$, so gilt auch $\sigma \in P$.

P ist Lebendigkeitseigenschaft genau dann, wenn $\sigma[..n] \in P$ gilt für alle Folgen $\sigma \in \Sigma^\omega$ und alle $n \in \mathbb{N}$.

Sicherheits- und Lebendigkeitseigenschaften

Beobachtung: Die Menge der Abläufe eines Transitionssystems $\Gamma = (Z, I, \mathcal{A}, \delta)$ ohne Fairnessbedingungen ist eine Sicherheitseigenschaft.

Denn: Sei $\sigma = s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$

σ Ablauf von Γ

gdw. $s_0 \in I$ und für alle $i \in \mathbb{N}$ gilt $(s_i, s_{i+1}) \in \delta$ [siehe Definition]
 gdw. für alle $n \in \mathbb{N}$ gilt $s_0 \in I$ und für alle $i < n$ gilt $(s_i, s_{i+1}) \in \delta$ [Arithmetik]
 gdw. für alle $n \in \mathbb{N}$ gibt es $\tau \in \Sigma^\omega$, so dass $\sigma[..n] \circ \tau$ Ablauf von Γ [wegen Totalität von δ]

Dagegen sind Fairnessbedingungen Lebendigkeitseigenschaften:

Wie in den vorangegangenen Sätzen bewiesen, kann jede Folge $\rho \in \Sigma^*$ zu einer Folge $\sigma \in \Sigma^\omega$ ergänzt werden, welche die Fairnessseigenschaft erfüllt.

Sicherheits- und Lebendigkeitseigenschaften

Beispiel Eisenbahn

- Die Eigenschaft „Es sind nie gleichzeitig zwei Züge auf der Brücke“ ist eine Sicherheitseigenschaft.

Denn trifft sie auf eine Folge $\sigma = s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$ nicht zu,

so gibt es einen kleinsten Index n mit zwei Zügen auf der Brücke im Zustand

s_n . Also verletzt schon $\sigma = s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots \xrightarrow{A_n} s_n$ die

Eigenschaft.

Logiken für Eigenschaften von Transitionssystemen

• Temporale Logik

- ist eine Erweiterung der mathematischen Logik um Aussagen, deren Wahrheitswert zu verschiedenen Zeitpunkten unterschiedlich sein kann;
- wird eingesetzt zur formalen Beschreibung von Eigenschaften reaktiver Systeme;
- findet insbesondere Anwendung bei der interaktiven und automatischen Verifikation reaktiver Systeme.

• Model Checking (Modellprüfung)

- bezeichnet ein Klasse von Verfahren zur automatischen Überprüfung von Eigenschaften, bei der systematisch der gesamte Zustandsraum untersucht wird;
- verwendet temporale Logik zur Beschreibung der Eigenschaften;
- wird erfolgreich zur Überprüfung des dynamischen Verhalten von Hardware und (neuerdings) Software eingesetzt.

Logiken für Eigenschaften von Transitionssystemen

- Typische temporal-logische Aussagen für eine Formel A sind
 - $\square A$ A gilt zu jedem Zeitpunkt („immer“, „always“)
 - $\diamond A$ A gilt irgendwann jetzt oder später („eventually“, „sometime“)
 - $\bigcirc A$ A gilt im nächsten Zeitpunkt („nexttime“)
- Man unterscheidet Temporale Logiken an ihrem Zeitmodell.
 - **LTL** **Lineare TL** (diskrete lineare Zeit)
 - **CTL** **Computational Tree Logic** (diskrete Zeit, mit Bäumen zur Darstellung der möglichen Entwicklungen der Zukunft)
 - Weitere Logiken mit **kontinuierlichem Zeitmodell**

Lineare Temporallogik (LTL)

Weitere Operatoren von LTL sind

- A until B** B gilt zu einem (echt) späteren Zeitpunkt und A gilt bis zu diesem Zeitpunkt
- A unless B** Wenn B zu einem (echt) späteren Zeitpunkt gilt, dann gilt A bis zu diesem Zeitpunkt. Sonst gilt A immer.

Beispiele

Eisenbahn

- Zwei Züge sind niemals gleichzeitig auf der Brücke:
 - $\square \neg(t1.oclInState(onBridge) \wedge t2.oclInState(onBridge))$
- Der Zug t1 ist immer wieder nicht auf der Brücke:
 - $\square \diamond \neg t1.oclInState(onBridge)$
- Immer wenn t1 am Signal wartet, wird t1 irgendwann später auf der Brücke sein:
 - $\square(t1.oclInState(onBridge) \Rightarrow \diamond t1.oclInState(onBridge))$
- Wenn t1 unendlich oft auf die Brücke fahren kann, so tut t1 dies auch unendlich oft:
 - $\square \diamond enabled(t1.enterBridge()) \Rightarrow \square \diamond t1.enterBridge()$

Typische Beispiele

Invarianten

 $\square P$ $\square \neg(crit_1 \wedge crit_2)$

gegenseitiger Ausschluss

 $\square(enab_1 \vee \dots \vee enab_n)$

Deadlock-Freiheit

Response, recurrence

 $\square(P \Rightarrow \diamond Q)$ $\square(try_1 \Rightarrow \diamond crit_1)$

Lebendigkeit: Zugang zum kritischen Abschnitt

 $\square \diamond \neg crit_1$

keine Blockierung im kritischen Abschnitt

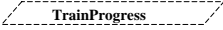
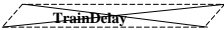
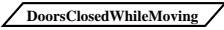


Reactivity, Streett

 $\square \diamond P \Rightarrow \square \diamond Q$ $\square \diamond (try_1 \wedge \neg crit_2) \Rightarrow \square \diamond crit_1$

(starke) Fairness

Modellierung von Zielen in KAOS: Typen von Zielen

Typ definiert Klasse mit vorgeschriebenem oder bevorzugtem Verhalten

- Achieve Goal: Zu erreichendes Ziel
($P \Rightarrow \diamond Q$)
Z.B. Achieve[TrainProgress] 
- Cease Goal: Unerwünschte Eigenschaft, die abgestellt werden soll
($P \Rightarrow \diamond \neg Q$) 
- Maintain Goal: Zu bewahrende Eigenschaft
($P \Rightarrow Q$) oder ($P \Rightarrow Q \wedge R$),
Z.B. Maintain[DoorsClosedWhileMoving] 
- Avoid Goal: Zu verhindernde Eigenschaft
($P \Rightarrow \neg Q$)
Z.B. Avoid[TrainsOnSameBlock] 
- Softgoal: Weiches Ziel (hilft Präferenzen zu setzen, wenn Alternativen ausgewählt werden)
Keine Formalisierung durch Logik
Z.B. Safe Transportation 

Beispielziel in KAOS

Goal[AchieveConvenientMeetingHeld]

Definition

Each requested meeting is eventually held with the presence of all intended participants

FormalDef

$\forall m:Meeting.$

($m.requested \Rightarrow$

$\langle \diamond (m.holds \wedge$

$\forall p:Participant. p.intended(m) \Rightarrow p.participates(m))$)

Zusammenfassung

- Reaktive Systeme werden formal durch **Transitionssysteme** modelliert.

- **Abläufe** eines Transitionssystems Γ sind Folgen $s_0 \xrightarrow{A_0} s_1 \xrightarrow{A_1} s_2 \dots$ von **Zuständen und Aktionen**, so dass s_0 ein Anfangszustand von Γ ist und alle Übergänge $s_i \xrightarrow{A_i} s_{i+1}$ der Zustandsübergangsrelation von Γ entsprechen.

- Zur Systemspezifikation werden **Abläufe** von Transitionssystemen meist weiter **eingeschränkt**.

Insbesondere verwendet man Fairnessbedingungen, um unfaire Abläufe auszuschließen. Fairnessbedingungen besagen intuitiv, dass Aktionen, die genügend oft ausführbar sind, immer wieder ausgeführt werden.

- Bei der Analyse von Transitionssystemen ist man hauptsächlich an Aussagen über ihre Abläufe interessiert und identifiziert daher Eigenschaften mit Mengen von Abläufen.

Methoden des Software Engineering (c) 2004, Koch, Störkle, Wirsing, LMU München

Zusammenfassung

- Zwei grundlegende Klassen von Eigenschaften sind Sicherheitseigenschaften ("something bad never happens") und Lebendigkeitseigenschaften ("something good eventually happens").

Eine Sicherheitseigenschaft trifft auf σ genau dann zu, wenn sie auf alle endlichen Anfangsstücke

$\sigma[..n]$ von σ zutrifft. Sicherheitseigenschaften sind also durch endliche Abläufe vollständig bestimmt.

Eine Lebendigkeitseigenschaft trifft auf alle endlichen Abläufe zu. Endliche Anfangsstücke geben also keinerlei Information auf das Zutreffen der Eigenschaft auf den Gesamtlauf.

- LTL und CTL sind Logiken zur formalen Beschreibung von Eigenschaften von Transitionssystemen.
- Die grundlegenden Operatoren von LTL sind „next“, „always“, „irgendwann“ und „until“.

Methoden des Software Engineering (c) 2004, Koch, Störkle, Wirsing, LMU München

Literatur

- **Kröger: Temporale Logik**
- **Wirsing: Grundlagen der Systementwicklung, Kap. 8**
- **Knapp: Model Checking Praktikum, LTL**

Methoden des Software Engineering (c) 2004, Koch, Störkle, Wirsing, LMU München

Appendix: Semantik von LTL

Formeln interpretiert über unendlichen Zustandsfolgen $\sigma = s_0 s_1 s_2 \dots$

σ_j bezeichnet Teilfolge $s_j s_{j+1} \dots$

$\sigma \models p$	gdw. $s_0 \models p$
$\sigma \models \neg \varphi$	gdw. $\sigma \not\models \varphi$
$\sigma \models \varphi \wedge \psi$	gdw. $\sigma \models \varphi$ und $\sigma \models \psi$
$\sigma \models \circ \varphi$	gdw. $\sigma_1 \models \varphi$
$\sigma \models \diamond \varphi$	gdw. $\sigma_j \models \varphi$ für ein $i \geq 0$
$\sigma \models \square \varphi$	gdw. $\sigma_j \models \varphi$ für alle $i \geq 0$
$\sigma \models \varphi \text{ until } \psi$	gdw. ex. $i \geq 0$ mit $\sigma_j \models \psi$ und für alle $i \leq j < k$ gilt $\sigma_j \models \varphi$
$\sigma \models \varphi \text{ unless } \psi$	gdw. $\sigma \models \varphi \text{ until } \psi$ oder $\sigma \models \square \varphi$

rekursive Charakterisierungen $\square \varphi \equiv \varphi \wedge \square \varphi$

Grundoperatoren \circ, until $\diamond \varphi \equiv \text{true until } \varphi, \square \varphi \equiv \neg \diamond \neg \varphi, \dots$

chen