LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Temporale Logik und Zustandssysteme

Kurzskriptum zur Vorlesung

Wintersemester 2004/05

F. Kröger

INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für Programmierung und Softwaretechnik

Inhaltsverzeichnis

Einleitung

Temporale Logik

Anliegen der „üblichen“ (mathematischen) Logik (***klassische Logik***):

- Bereitstellung einer formalen Sprache zur präzisen Formulierung von (mathematischen) ***Aussagen***,
- Untersuchung von Instrumenten, mit denen man feststellen kann, ob eine Aussage „zutrifft“ oder „nicht zutrifft“.

Temporale Logik (TL) bezieht zusätzlich ein:

- Betrachtung zeitabhängiger Aussagen,
- Bereitstellung einer Sprache zur präzisen Formulierung von zeitlichen Beziehungen von Aussagen und die Untersuchung des (ebenfalls zeitabhängigen) Zutreffens solcher Beziehungen.

Zustandssysteme

Beispiel eines „Nicht-Zustandssystems“:

> In drei Behältern liegen rote, blaue und gelbe Kugeln und Würfel. Dabei gilt:
> (1) Falls ein Behälter blaue Gegenstände enthält, so enthält er keine roten Gegenstände.
> (2) In jedem Behälter liegt mindestens ein Würfel.
> (3) In mindestens einem Behälter liegen weder gelbe noch blaue Würfel.

Beispiel eines Zustandssystems:

> Eine Ampel hat die drei Anzeigen rot, gelb und grün. Dabei gilt:
> (1) Die Ampel kann ein- und ausgeschaltet werden.
> (2) Solange die Ampel an ist, schaltet sie fortwährend von rot auf grün, von grün auf gelb, von gelb auf rot usw.
> (3) Die Ampel kann nur ausgeschaltet werden, wenn sie grün ist.
> (4) Wird die Ampel angeschaltet, so wird sie dabei gelb.

Allgemein:
Der (informelle) Sammelbegriff „Zustandssystem“ fasst alle Arten von Systemen zusammen, die „ablaufen“, z.B.: Software-Module, Übertragungsprotokolle, Datenbanksysteme, Schaltwerke, Automaten usw.
Dabei: Ein „Ablauf“ ist eine Folge von „Zuständen“.

Anwendung von TL auf Zustandssysteme:
Versteht man Zustände eines Ablaufs als „Zeitpunkte“, so ermöglicht dies die Formulierung von Aussagen über Systemabläufe und deren (temporal-) logische Behandlung.

Inhalt der Vorlesung

1. Darstellung der Grundlagen temporaler Logik(en).
2. Temporallogische ***Spezifikation*** von Zustandssystemen
   (formale Beschreibung der Wirkungsweise eines Zustandssystems in TL).
3. Anwendung auf die ***Verifikation*** von Zustandssystemen
   (insbesondere: ***parallele Programme*** und ***zustands-endliche Systeme***).

Literaturhinweise

Kröger: *Temporal Logic of Programs.*
Springer 1987.

Manna/Pnueli: *The Temporal Logic of Reactive and Concurrent Systems – Specification (Vol. 1).*
Springer 1992.

Manna/Pnueli: *The Temporal Logic of Reactive and Concurrent Systems – Safety Properties (Vol. 2).*
Springer 1995.

Gabbay/Hodkinson/Reynolds: *Temporal Logic – Mathematical Foundations and Computational Aspects (Vol. 1).*
Clarendon Press 2001.

Hamilton: *Logic for Mathematicians.*
Cambridge University Press 1988 (Revised edition).

Kapitel 1
Logiken und Theorien

1.1 Formale Logiken

Allgemeine Definition von Logiken

Eine ***(formale) Logik*** formalisiert das Schließen über „Aussagen“ eines bestimmten „Diskursbereichs“.
Die Definition einer solchen Logik LOG beinhaltet jeweils die Angabe

- der ***Syntax (Sprache)*** von LOG,
- der ***Semantik*** von LOG.

Darüber hinaus wird häufig eine

- ***Axiomatisierung*** von LOG

angestrebt.

Syntax

Die Syntax einer Logik LOG gibt die Art der LOG zugrundeliegenden (formalen) Sprache(n) $\mathcal{L}_{\mathrm{LOG}}$ über einem (nicht notwendig endlichen) Alphabet von Zeichen an.
Die Elemente von $\mathcal{L}_{\mathrm{LOG}}$ heißen ***Formeln (von*** LOG***)***. Sie repräsentieren die Aussagen des Diskursbereichs.

Semantik

Die Semantik einer Logik LOG gibt an:

- die Klasse der möglichen ***Interpretationen*** $\mathfrak{I}$ der Sprachbestandteile von $\mathcal{L}_{\mathrm{LOG}}$ als deren formal beschriebene „Bedeutungen“;
- die Definition der Relation $\models_{\mathfrak{I}} A$ für Interpretationen $\mathfrak{I}$ und Formeln A von $\mathcal{L}_{\mathrm{LOG}}$; Sprechweise für $\models_{\mathfrak{I}} A$: A ist ***gültig in*** $\mathfrak{I}$.

Damit wird definiert:

Definition. Seien A eine Formel und $\mathcal{F}$ eine Menge von Formeln von $\mathcal{L}_{\mathrm{LOG}}$ für eine Logik LOG. A heißt ***Folgerung von*** $\mathcal{F}$ (auch: A ***folgt aus*** $\mathcal{F}$), geschrieben: $\mathcal{F} \models A$, wenn gilt:

$\models_{\mathfrak{I}} A$ für alle $\mathfrak{I}$ mit der Eigenschaft, dass $\models_{\mathfrak{I}} B$ für alle $B \in \mathcal{F}$.

A heißt ***allgemeingültig***, wenn $\emptyset \models A$ (d.h. $\models_{\mathfrak{I}} A$ für alle $\mathfrak{I}$) gilt.

Schreibweisen: $B_1, \ldots, B_n \models A$, falls $\mathcal{F} = \{B_1, \ldots, B_n\}$,
$\models A$ für $\emptyset \models A$.

Satz 1.1.1 Seien A eine Formel und $\mathcal{F}, \mathcal{F}'$ Mengen von Formeln von $\mathcal{L}_{\mathrm{LOG}}$ für eine Logik LOG. Falls $\mathcal{F} \models B$ für alle $B \in \mathcal{F}'$ und $\mathcal{F}' \models A$, so gilt $\mathcal{F} \models A$.

Axiomatisierung

Eine Axiomatisierung einer Logik LOG geschieht durch ein ***formales System*** Σ_{LOG} mit

- einer Menge von Formeln von $\mathcal{L}_{\mathrm{LOG}}$, genannt ***Axiome***,
- einer Menge von ***(Herleitungs-) Regeln*** der Form $A_1, \ldots, A_n \vdash B \ \ (n \geq 1)$;
  die Formeln $A_1, \ldots, A_n$ heißen ***Prämissen***, die Formel B ***Konklusion*** der Regel.

Induktive Definition der ***Herleitbarkeit*** einer Formel A ***aus*** einer Menge $\mathcal{F}$ von Formeln von $\mathcal{L}_{\mathrm{LOG}}$ im formalen System Σ_{LOG} (geschrieben: $\mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} A$ oder kurz $\mathcal{F} \vdash A$ oder auch $B_1, \ldots, B_n \vdash_{(\Sigma_{\mathrm{LOG}})} A$, falls $\mathcal{F} = \{B_1, \ldots, B_n\}$):

1. $\mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} A$ für jedes $A \in \mathcal{F}$.
2. $\mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} A$ für jedes Axiom A von Σ_{LOG}.
3. Ist $A_1, \ldots, A_n \vdash B$ Regel von Σ_{LOG} und gilt $\mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} A_1, \ldots, \mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} A_n$, so ist auch $\mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} B$.

A heißt ***herleitbar*** (in Σ_{LOG}), wenn $\emptyset \vdash_{\Sigma_{\mathrm{LOG}}} A$ (geschrieben: $\vdash A$) gilt.

Bemerkung: Ist A herleitbar aus $\{B_1, \ldots, B_n\}$, so kann $B_1, \ldots, B_n \vdash A$ als ***abgeleitete Regel*** in anderen Herleitungen verwendet werden.

Korrektheit und Vollständigkeit

Definition. Ein formales System Σ_{LOG} für eine Logik LOG heißt ***korrekt*** (bzgl. der Semantik von LOG), wenn gilt:

$$\mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} A \ \Rightarrow \ \mathcal{F} \models A \qquad \text{(für alle } \mathcal{F} \text{ und alle } A\text{)}.$$

Σ_{LOG} heißt ***vollständig***, wenn gilt:

$$\mathcal{F} \models A \ \Rightarrow \ \mathcal{F} \vdash_{\Sigma_{\mathrm{LOG}}} A \qquad \text{(für alle } \mathcal{F} \text{ und alle } A\text{)}.$$

Bemerkung: Ist Σ_{LOG} korrekt und vollständig, so gilt insbesondere:

$$\vdash_{\Sigma_{\mathrm{LOG}}} A \ \Leftrightarrow \ \models A \qquad \text{(für alle } A\text{)}.$$

Satz 1.1.2 Sei Σ_{LOG} ein formales System für eine Logik LOG. Falls $\models A$ für alle Axiome A von Σ_{LOG} und $A_1, \ldots, A_n \models B$ für alle Regeln $A_1, \ldots, A_n \vdash B$ von Σ_{LOG}, so ist Σ_{LOG} korrekt.

1.2 Klassische Aussagenlogik

Sprache

Erstes Beispiel einer Logik: ***(klassische) Aussagenlogik (PL)***.

Sei $\mathcal{V}$ eine (endliche oder abzählbar unendliche) Menge von ***atomaren Aussagen***. Eine Sprache $\mathcal{L}_{\mathrm{PL}}(\mathcal{V})$ (kurz: $\mathcal{L}_{\mathrm{PL}}$) ist definiert wie folgt:

Alphabet:

- Alle Elemente von $\mathcal{V}$,
- die Zeichen **false** $\mid \rightarrow \mid (\mid)$.

Induktive Definition der Formeln:

1. Jede atomare Aussage $v \in \mathcal{V}$ ist eine Formel.
2. **false** ist eine Formel.
3. Sind A und B Formeln, so ist auch $(A \rightarrow B)$ eine Formel.

Abkürzungen:

$\neg A$	für	$A \rightarrow$ **false**,
$A \vee B$	für	$\neg A \rightarrow B$,
$A \wedge B$	für	$\neg(A \rightarrow \neg B)$,
$A \leftrightarrow B$	für	$(A \rightarrow B) \wedge (B \rightarrow A)$,
true	für	$\neg$**false**

(äußerste Klammern jeweils weggelassen).

Semantik

Interpretationen für $\mathcal{L}_{\mathrm{PL}}$ sind gegeben durch ***Belegungen***.

Es seien tt („true“) und ff („false“) zwei verschiedene ***Wahrheitswerte***. Eine Belegung B für eine Menge $\mathcal{V}$ von atomaren Aussagen ist eine Abbildung

$$\mathsf{B} : \mathcal{V} \rightarrow \{\mathsf{ff}, \mathsf{tt}\}.$$

Jedes B kann induktiv fortgesetzt werden auf die Menge aller Formeln von $\mathcal{L}_{\mathrm{PL}}(\mathcal{V})$:

1. $\mathsf{B}(v)$ für $v \in \mathcal{V}$ ist gegeben.
2. $\mathsf{B}($**false**$) = \mathsf{ff}$.
3. $\mathsf{B}(A \rightarrow B) = \mathsf{tt} \Leftrightarrow \mathsf{B}(A) = \mathsf{ff}$ oder $\mathsf{B}(B) = \mathsf{tt}$.

Für die weiteren Operationen gilt dann:

4. $\mathsf{B}(\neg A) = \mathsf{tt} \Leftrightarrow \mathsf{B}(A) = \mathsf{ff}$.
5. $\mathsf{B}(A \vee B) = \mathsf{tt} \Leftrightarrow \mathsf{B}(A) = \mathsf{tt}$ oder $\mathsf{B}(B) = \mathsf{tt}$.
6. $\mathsf{B}(A \wedge B) = \mathsf{tt} \Leftrightarrow \mathsf{B}(A) = \mathsf{tt}$ und $\mathsf{B}(B) = \mathsf{tt}$.

7. $\mathsf{B}(A \leftrightarrow B) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{B}(A) = \mathsf{B}(B)$.
8. $\mathsf{B}(\mathbf{true}) = \mathsf{tt}$.

Die Gültigkeit $\models_{\mathsf{B}}$ („gültig in B“) ist definiert durch:

$$\models_{\mathsf{B}} A \;\Leftrightarrow\; \mathsf{B}(A) = \mathsf{tt}.$$

Beispiele:

Allgemeingültige Formeln:	$((A \wedge B) \vee C) \leftrightarrow ((A \vee C) \wedge (B \vee C))$, $\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$.
Folgerung:	$A \to B, B \to C \models A \to C$.

Bemerkungen:

1. Eine allgemeingültige Formel von PL heißt auch ***Tautologie***.
2. Es gilt (für Formelmengen $\mathcal{F}$ und Formeln A, B):

   $$\mathcal{F} \cup \{A\} \models B \;\Leftrightarrow\; \mathcal{F} \models A \to B.$$

   Daraus folgt:

   $$A \models B \;\Leftrightarrow\; \models A \to B,$$
   $$A_1, \ldots, A_n \models B \;\Leftrightarrow\; \models A_1 \to (A_2 \to \ldots \to (A_n \to B)\ldots),$$
   $$A_1, \ldots, A_n \models B \;\Leftrightarrow\; \models (A_1 \wedge \ldots \wedge A_n) \to B.$$

   (Beachte: Schreibweise ohne innere Klammern in $(A_1 \wedge \ldots \wedge A_n)$.)

Axiomatisierung

Ein mögliches formales System Σ_{PL} ist gegeben durch:

Axiome (genauer: Axiomenschemata):

- $A \to (B \to A)$,
- $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$,
- $((A \to \mathbf{false}) \to \mathbf{false}) \to A$.

Regel (genauer: Regelschema):

- $A, A \to B \vdash B$ (***modus ponens***).

Bemerkungen:

1. Σ_{PL} ist korrekt und vollständig.
2. Es gilt (***Deduktionstheorem*** und seine Umkehrung):

   $$\mathcal{F} \cup \{A\} \vdash B \;\Leftrightarrow\; \mathcal{F} \vdash A \to B \qquad \text{(für Formelmengen } \mathcal{F} \text{ und Formeln } A, B).$$

   Daraus folgt:

   $$A \vdash B \;\Leftrightarrow\; \vdash A \to B,$$
   $$A_1, \ldots, A_n \vdash B \;\Leftrightarrow\; \vdash A_1 \to (A_2 \to \ldots \to (A_n \to B)\ldots),$$
   $$A_1, \ldots, A_n \vdash B \;\Leftrightarrow\; \vdash (A_1 \wedge \ldots \wedge A_n) \to B.$$

1.3 Theorien

Allgemeine Definition von Theorien

Sei LOG eine Logik. Eine LOG-***Theorie*** $Th = (\mathcal{L}_{\mathrm{LOG}}, \mathcal{A})$ ist gegeben durch

- eine Sprache $\mathcal{L}_{\mathrm{LOG}}$ von LOG,
- eine Menge $\mathcal{A}$ von Formeln von $\mathcal{L}_{\mathrm{LOG}}$ (***nicht-logische Axiome***).

Eine Interpretation $\mathfrak{I}$ für die Sprache $\mathcal{L}_{\mathrm{LOG}}$ mit

$$\models_{\mathfrak{I}} A \text{ für alle } A \in \mathcal{A}$$

heißt ***Modell*** von Th.

Bemerkung: Ist $Th = (\mathcal{L}_{\mathrm{LOG}}, \mathcal{A})$ eine LOG-Theorie und Σ_{LOG} ein korrektes formales System für LOG, so gilt:

$$\mathcal{A} \vdash_{\Sigma_{\mathrm{LOG}}} A \;\Rightarrow\; \mathcal{A} \models A \;\Rightarrow\; \models_{\mathfrak{I}} A \text{ für jedes Model } \mathfrak{I} \text{ von } Th.$$

Beispiele

1. PL-Theorie $Th_{bsp} = (\mathcal{L}_{\mathrm{PL}}(\mathcal{V}_{bsp}), \mathcal{A}_{bsp})$ mit

   $\mathcal{V}_{bsp} = \{v_1, v_2, v_3\}$,
   $\mathcal{A}_{bsp} = \{v_1 \leftrightarrow \neg v_2, v_2 \vee v_3\}$.

   Modelle von Th:

   $\mathsf{B}_1 : \mathcal{V}_{bsp} \to \{\mathsf{ff}, \mathsf{tt}\}$ mit $\mathsf{B}_1(v_1) = \mathsf{ff}, \mathsf{B}_1(v_2) = \mathsf{tt}, \mathsf{B}_1(v_3) = \mathsf{ff}$,
   $\mathsf{B}_2 : \mathcal{V}_{bsp} \to \{\mathsf{ff}, \mathsf{tt}\}$ mit $\mathsf{B}_2(v_1) = \mathsf{tt}, \mathsf{B}_2(v_2) = \mathsf{ff}, \mathsf{B}_2(v_3) = \mathsf{tt}$.

2. Formalisierung des „Behälter-Systems“ (siehe Einleitung) als PL-Theorie:

   $Th_{beh} = (\mathcal{L}_{\mathrm{PL}}(\mathcal{V}_{beh}), \mathcal{A}_{beh})$ mit

   - $\mathcal{V}_{beh} = \{enth_{i,farbe,gestalt} \mid i \in \{1,2,3\}, farbe \in \{r, b, g\}, gestalt \in \{k, w\}\}$

     (Intention: $enth_{2,g,k} \mathrel{\hat{=}}$ „Behälter 2 enthält gelbe Kugel(n)“,
     analog für die anderen Index-Werte),

   - $\mathcal{A}_{beh}$ bestehend aus:
     - $(enth_{i,b,w} \vee enth_{i,b,k}) \to (\neg enth_{i,r,w} \wedge \neg enth_{i,r,k})$ für $i = 1, 2, 3$,
     - $enth_{i,b,w} \vee enth_{i,r,w} \vee enth_{i,g,w}$ für $i = 1, 2, 3$,
     - $(\neg enth_{1,g,k} \wedge \neg enth_{1,b,k}) \vee$
       $(\neg enth_{2,g,k} \wedge \neg enth_{2,b,k}) \vee$
       $(\neg enth_{3,g,k} \wedge \neg enth_{3,b,k})$.

   Es gilt (z.B.):

   $$\mathcal{A}_{beh} \models \neg enth_{1,b,k} \vee \neg enth_{2,b,k} \vee \neg enth_{3,b,k}$$

   („Einer der Behälter enthält keine blauen Kugeln“).

Kapitel 2
Lineare temporale Aussagenlogik

2.1 Sprache und Semantik

Das Grundkonzept

Ziel:
Gegeben Aussagen $A, B, \ldots$;
Formulierung neuer Aussagen über das Zutreffen von $A, B, \ldots$ zu „anderen Zeitpunkten“.

Basisoperatoren hierfür:

- $\circ A$: „A trifft im unmittelbar folgenden Zeitpunkt zu“ (***nexttime***-Operator);
- $\Box A$: „A trifft in allen nachfolgenden Zeitpunkten zu“ (***always***-/***henceforth***-Operator);
- $\Diamond A$: „A trifft in (mindestens) einem nachfolgenden Zeitpunkt zu“ (***sometime***-/***eventuality***-Operator).

Die Basissprache

Sei $\mathcal{V}$ eine (endliche oder abzählbar unendliche) Menge von ***atomaren Aussagen***. Eine (Basis-) Sprache $\mathcal{L}_{\mathrm{LTL}}(\mathcal{V})$ (kurz: $\mathcal{L}_{\mathrm{LTL}}$) der ***(linearen) temporalen Aussagenlogik (LTL)*** ist definiert wie folgt:

Alphabet:

- Alle Elemente von $\mathcal{V}$,
- die Zeichen **false** $\mid \rightarrow \mid \circ \mid \Box \mid (\mid)$.

Induktive Definition der Formeln:

1. Jede atomare Aussage $v \in \mathcal{V}$ ist eine Formel.
2. **false** ist eine Formel.
3. Sind A und B Formeln, so ist auch $(A \rightarrow B)$ eine Formel.
4. Ist A eine Formel, so sind auch $\circ A$ und $\Box A$ Formeln.

Abkürzungen: $\neg, \vee, \wedge, \leftrightarrow$, **true**: wie in PL,
$\Diamond A$ für $\neg\Box\neg A$.

Schreibweisen: Äußerste Klammern werden weggelassen.
Weitere Klammerersparnis durch folgende Operator-Prioritätsregeln:
$\vee, \wedge$ binden stärker als $\rightarrow, \leftrightarrow$,
$\rightarrow$ bindet stärker als $\leftrightarrow$.
($\neg, \circ, \Box, \Diamond$ binden gemäß Definition stärker als $\vee, \wedge, \rightarrow, \leftrightarrow$.)

Beispiel:

Formel von $\mathcal{L}_{\mathrm{LTL}}$ (gemäß Syntaxdefinition):

$$(((\Box A_1 \wedge A_2) \rightarrow A_3) \leftrightarrow (\Diamond(\circ A_4 \vee \neg A_5) \wedge A_6))$$

Schreibweise mit Klammerersparnis:

$$\Box A_1 \wedge A_2 \rightarrow A_3 \;\leftrightarrow\; \Diamond(\circ A_4 \vee \neg A_5) \wedge A_6$$

Semantik

Interpretationen für $\mathcal{L}_{\mathrm{LTL}}$ sind gegeben durch ***temporale Strukturen (Kripke-Strukturen)***.

Eine temporale Struktur für eine Menge $\mathcal{V}$ von atomaren Aussagen ist eine unendliche Folge $\mathsf{K} = (\eta_0, \eta_1, \eta_2, \ldots)$ von Abbildungen

$$\eta_i : \mathcal{V} \;\rightarrow\; \{\mathsf{ff}, \mathsf{tt}\}$$

(ff, tt wie bei PL). Die η_i heißen ***Zustände***, η_0 heißt ***Anfangszustand***.
(Beachte: Jeder Zustand ist eine Belegung im Sinne der klassischen Aussagenlogik.)

Für eine temporale Struktur $\mathsf{K} = (\eta_0, \eta_1, \eta_2, \ldots)$ wird $\mathsf{K}_i(F) \in \{\mathsf{ff}, \mathsf{tt}\}$ für jede Formel F von $\mathcal{L}_{\mathrm{LTL}}$ und jedes $i \in \mathbb{N}$ („Wahrheitswert von F im i-ten Zustand von K“) induktiv definiert wie folgt:

1. $\mathsf{K}_i(v) = \eta_i(v)$ für $v \in \mathcal{V}$.
2. $\mathsf{K}_i(\textbf{false}) = \mathsf{ff}$.
3. $\mathsf{K}_i(A \rightarrow B) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_i(A) = \mathsf{ff}$ oder $\mathsf{K}_i(B) = \mathsf{tt}$.
4. $\mathsf{K}_i(\circ A) = \mathsf{K}_{i+1}(A)$.
5. $\mathsf{K}_i(\Box A) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_j(A) = \mathsf{tt}$ für alle $j \geq i$.

Für die weiteren Operationen gilt dann:

6. $\mathsf{K}_i(\neg A) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_i(A) = \mathsf{ff}$.
7. $\mathsf{K}_i(A \vee B) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_i(A) = \mathsf{tt}$ oder $\mathsf{K}_i(B) = \mathsf{tt}$.
8. $\mathsf{K}_i(A \wedge B) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_i(A) = \mathsf{tt}$ und $\mathsf{K}_i(B) = \mathsf{tt}$.
9. $\mathsf{K}_i(A \leftrightarrow B) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_i(A) = \mathsf{K}_i(B)$.
10. $\mathsf{K}_i(\textbf{true}) = \mathsf{tt}$.
11. $\mathsf{K}_i(\Diamond A) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_j(A) = \mathsf{tt}$ für ein $j \geq i$.

Die Gültigkeit $\models_{\mathsf{K}}$ („gültig in K“) ist definiert durch:

$$\models_{\mathsf{K}} A \;\Leftrightarrow\; \mathsf{K}_i(A) = \mathsf{tt} \text{ für alle } i \in \mathbb{N}.$$

Beispiele

1. Seien $A \equiv \Diamond\neg v_1 \wedge \bigcirc v_1 \rightarrow \Box v_2$, $B \equiv \Diamond v_2$ und K gegeben mit:

	η_0	η_1	η_2	η_3	η_4	...
v_1	ff	ff	tt	tt	ff	...(beliebig)...
v_2	tt	tt	ff	tt	tt	...(unverändert tt)...

   (Die Werte für andere $v \in \mathcal{V}$ sind irrelevant.) Dann gilt:

$$\mathsf{K}_0(\bigcirc v_1) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_0(\Diamond\neg v_1 \wedge \bigcirc v_1) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_0(A) = \mathsf{tt},$$
$$\mathsf{K}_1(\Diamond\neg v_1) = \mathsf{K}_1(\bigcirc v_1) = \mathsf{tt}, \mathsf{K}_1(\Box v_2) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_1(A) = \mathsf{ff},$$
$$\mathsf{K}_2(\Diamond\neg v_1) = \mathsf{K}_2(\bigcirc v_1) = \mathsf{tt}, \mathsf{K}_2(\Box v_2) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_2(A) = \mathsf{ff},$$
$$\mathsf{K}_3(\Box v_2) = \mathsf{K}_4(\Box v_2) = \ldots = \mathsf{tt} \;\Rightarrow\; \mathsf{K}_3(A) = \mathsf{K}_4(A) = \ldots = \mathsf{tt}.$$

   Insbesondere: A ist nicht gültig in K.
   Dagegen: B ist gültig in K.

2. Die Formel

$$\neg\bigcirc A \leftrightarrow \bigcirc\neg A$$

   ist allgemeingültig.

3. Einige Formeln mit ihrer informellen Bedeutung:

$A \rightarrow \bigcirc B$:	„Falls A, dann B im nächsten Zustand“,
$A \rightarrow \Box B$:	„Falls A, dann ab jetzt auch B“,
$A \rightarrow \Diamond B$:	„Falls A, dann irgendwann einmal (jetzt oder zukünftig) B“,
$\Box(A \rightarrow B)$:	„Wann immer ab jetzt A, dann auch B in dem Zustand“,
$\Box\Diamond A$:	„Zu jedem Zustand ab jetzt gibt es einen späteren Zustand, in dem A zutrifft“, d.h.: „A trifft ab jetzt unendlich oft zu“,
$\Diamond\Box A$:	„Irgendwann einmal trifft A permanent zu“, d.h.:, „A trifft von jetzt ab nur endlich mal nicht zu“.

Einige semantische Eigenschaften

Lemma 2.1.1 Sei $\mathsf{K} = (\eta_0, \eta_1, \eta_2, \ldots)$ eine temporale Struktur, $i \in \mathbb{N}$. Falls $\mathsf{K}_i(A) = \mathsf{tt}$ und $\mathsf{K}_i(A \rightarrow B) = \mathsf{tt}$, so $\mathsf{K}_i(B) = \mathsf{tt}$.

Satz 2.1.2 Es gilt: $A, A \rightarrow B \models B$.

Satz 2.1.3 Es gilt: $A \models \circ A$ und $A \models \Box A$.

Satz 2.1.4 Es gilt: $A \to B, A \to \circ A \models A \to \Box B$.

Lemma 2.1.5 Seien $\mathsf{K} = (\eta_0, \eta_1, \eta_2, \ldots)$ und $\mathsf{K}' = (\eta'_0, \eta'_1, \eta'_2, \ldots)$ temporale Strukturen, $i \in \mathbb{N}$ und $\eta'_j = \eta_{i+j}$ für alle $j \in \mathbb{N}$. Dann ist $\mathsf{K}'_j(A) = \mathsf{K}_{i+j}(A)$ für alle $j \in \mathbb{N}$.

Satz 2.1.6 $\mathcal{F} \cup \{A\} \models B$ genau dann, wenn $\mathcal{F} \models \Box A \to B$.

Satz 2.1.7 Falls $\mathcal{F} \models A \to B$, so $\mathcal{F} \cup \{A\} \models B$.

2.2 Logische Gesetze

Klassische Logik in LTL

Definition. Eine Formel von $\mathcal{L}_{\mathrm{LTL}}$ heißt ***aussagenlogisch (tautologisch) gültig***, wenn sie aus einer Tautologie von $\mathcal{L}_{\mathrm{PL}}$ durch (konsistente) Ersetzung der atomaren Aussagen durch Formeln von $\mathcal{L}_{\mathrm{LTL}}$ entsteht.

Satz 2.2.1 Jede aussagenlogisch gültige Formel ist allgemeingültig.

Definition. $A_1, \ldots, A_n, B$ $(n \geq 1)$ seien Formeln von $\mathcal{L}_{\mathrm{LTL}}$. B heißt ***aussagenlogische Folgerung*** von $A_1, \ldots, A_n$, wenn die Formel $A_1 \to (A_2 \to \ldots \to (A_n \to B) \ldots)$ aussagenlogisch gültig ist.

Satz 2.2.2 Ist B aussagenlogische Folgerung von $A_1, \ldots, A_n$, so gilt: $A_1, \ldots, A_n \models B$.
.

Eine Liste allgemeingültiger Formeln

Dualitäts-Gesetze

(T1) $\neg\circ A \leftrightarrow \circ\neg A$
(T2) $\neg\Box A \leftrightarrow \Diamond\neg A$
(T3) $\neg\Diamond A \leftrightarrow \Box\neg A$

Reflexivitäts-Gesetze

(T4) $\Box A \to A$
(T5) $A \to \Diamond A$

Gesetze über die „Stärke“ der Operatoren

(T6) $\Box A \rightarrow \bigcirc A$
(T7) $\bigcirc A \rightarrow \Diamond A$
(T8) $\Box A \rightarrow \Diamond A$
(T9) $\Diamond\Box A \rightarrow \Box\Diamond A$

Idempotenz-Gesetze

(T10) $\Box\Box A \leftrightarrow \Box A$
(T11) $\Diamond\Diamond A \leftrightarrow \Diamond A$

Kommutativ-Gesetze

(T12) $\Box\bigcirc A \leftrightarrow \bigcirc\Box A$
(T13) $\Diamond\bigcirc A \leftrightarrow \bigcirc\Diamond A$

Distributiv-Gesetze

(T14) $\bigcirc(A \rightarrow B) \leftrightarrow \bigcirc A \rightarrow \bigcirc B$
(T15) $\bigcirc(A \wedge B) \leftrightarrow \bigcirc A \wedge \bigcirc B$
(T16) $\bigcirc(A \vee B) \leftrightarrow \bigcirc A \vee \bigcirc B$
(T17) $\Box(A \wedge B) \leftrightarrow \Box A \wedge \Box B$
(T18) $\Diamond(A \vee B) \leftrightarrow \Diamond A \vee \Diamond B$

Schwache Distributiv-Gesetze

(T19) $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$
(T20) $\Box A \vee \Box B \rightarrow \Box(A \vee B)$
(T21) $(\Diamond A \rightarrow \Diamond B) \rightarrow \Diamond(A \rightarrow B)$
(T22) $\Diamond(A \wedge B) \rightarrow \Diamond A \wedge \Diamond B$

Fixpunkt-Charakterisierungen von $\Box$ und $\Diamond$

(T23) $\Box A \leftrightarrow A \wedge \bigcirc\Box A$
(T24) $\Diamond A \leftrightarrow A \vee \bigcirc\Diamond A$

Monotonie-Gesetze

(T25) $\Box(A \rightarrow B) \rightarrow (\bigcirc A \rightarrow \bigcirc B)$
(T26) $\Box(A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B)$

Rahmen-Gesetze

(T27) $\Box A \rightarrow (\bigcirc B \rightarrow \bigcirc(A \wedge B))$
(T28) $\Box A \rightarrow (\Box B \rightarrow \Box(A \wedge B))$
(T29) $\Box A \rightarrow (\Diamond B \rightarrow \Diamond(A \wedge B))$

Temporallogische ***Generalisierungs***- und ***Partikularisierungs***-Gesetze

(T30) $\Box(\Box A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$
(T31) $\Box(A \rightarrow \Diamond B) \rightarrow (\Diamond A \rightarrow \Diamond B)$

Bemerkung

Gesetze der Form

$$\Box A \rightarrow B$$

können auch als Folgerungsbeziehung in der Form

$$A \models B$$

geschrieben werden. Beispiele:

(T19) $A \to B \models \Box A \to \Box B$
(T25) $A \to B \models \circ A \to \circ B$
(T27) $A \models \circ B \to \circ(A \wedge B)$
(T30) $\Box A \to B \models \Box A \to \Box B$
(T31) $A \to \Diamond B \models \Diamond A \to \Diamond B$

2.3 Axiomatisierung

Das formale System Σ_{LTL}

Ein mögliches formales System Σ_{LTL} für LTL ist gegeben durch:

Axiome:
(taut) Alle aussagenlogisch gültigen Formeln,
(ltl1) $\neg\circ A \leftrightarrow \circ\neg A$,
(ltl2) $\circ(A \to B) \to (\circ A \to \circ B)$,
(ltl3) $\Box A \to A \wedge \circ\Box A$.

Regeln:
(mp) $A, A \to B \vdash B$,
(nex) $A \vdash \circ A$,
(ind) $A \to B, A \to \circ A \vdash A \to \Box B$.

Bemerkung:

Statt dem der Einfachheit halber gewählten (taut) könnten übliche Axiome der klassischen Aussagenlogik (vgl. Abschnitt 1.2) verwendet werden.

Satz 2.3.1 (Korrektheitssatz für Σ_{LTL})
Sei A eine Formel und $\mathcal{F}$ eine Menge von Formeln. Falls $\mathcal{F} \vdash A$ dann $\mathcal{F} \models A$.

Satz 2.3.2 Ist die Formel B aussagenlogische Folgerung der Formeln $A_1, \ldots, A_n$, so ist $A_1, \ldots, A_n \vdash B$.

Folgerung aus Satz 2.3.2: In Herleitungen in Σ_{LTL} kann die zusätzliche Regel

(prop) $A_1, \ldots, A_n \vdash B$, falls B aussagenlogische Folgerung von $A_1, \ldots, A_n$

verwendet werden. Beispiel:

$A \to B, B \to C \vdash A \to C$ (***Kettenregel***).

Beispiele für Herleitungen

(ltl2') $(\circ A \to \circ B) \to \circ(A \to B)$ („Umkehrrichtung“ von (ltl2))

(1)	$\neg(A \to B) \to A$	(taut)
(2)	$\circ(\neg(A \to B) \to A)$	(nex),(1)
(3)	$\circ(\neg(A \to B) \to A) \to (\circ\neg(A \to B) \to \circ A)$	(ltl2)
(4)	$\circ\neg(A \to B) \to \circ A$	(mp),(2),(3)
(5)	$\neg\circ(A \to B) \leftrightarrow \circ\neg(A \to B)$	(ltl1)
(6)	$\neg\circ(A \to B) \to \circ A$	(prop),(5),(4)
(7)	$\neg(A \to B) \to \neg B$	(taut)
(8)	$\neg\circ(A \to B) \to \circ\neg B$	aus (7) ebenso wie (6) aus (1)
(9)	$\circ\neg B \to \neg\circ B$	(prop),(ltl1)
(10)	$\neg\circ(A \to B) \to \neg\circ B$	(prop),(8),(9)
(11)	$\neg\circ(A \to B) \to \neg(\circ A \to \circ B)$	(prop),(6),(10)
(12)	$(\circ A \to \circ B) \to \circ(A \to B)$	(prop),(11)

(ltl3') $A \wedge \circ\Box A \to \Box A$ („Umkehrrichtung“ von (ltl3))

(1)	$A \wedge \circ\Box A \to A$	(taut)
(2)	$\Box A \to A \wedge \circ\Box A$	(ltl3)
(3)	$\circ(\Box A \to A \wedge \circ\Box A)$	(nex)(2)
(4)	$\circ\Box A \to \circ(A \wedge \circ\Box A)$	(ltl2),(mp),(3)
(5)	$A \wedge \circ\Box A \to \circ(A \wedge \circ\Box A)$	(prop),(4)
(6)	$A \wedge \circ\Box A \to \Box A$	(ind),(1),(5)

Einige abgeleitete Regeln

(ind1) $A \to \circ A \vdash A \to \Box A$,

(ind2) $A \to B, B \to \circ B \vdash A \to \Box B$,

(alw) $A \vdash \Box A$,

(som) $A \to \circ B \vdash A \to \Diamond B$.

((ind1) und (ind2) sind nützliche Varianten der Regel (ind).)

Das Deduktionstheorem

Satz 2.3.3 (Deduktionstheorem von LTL)
Seien A, B Formeln, $\mathcal{F}$ eine Menge von Formeln. Falls $\mathcal{F} \cup \{A\} \vdash B$, so $\mathcal{F} \vdash \Box A \to B$.

Bemerkungen:

1. Das Deduktionstheorem

   $$\text{Falls } \mathcal{F} \cup \{A\} \vdash B \text{, so } \mathcal{F} \vdash A \to B$$

   der klassischen Aussagenlogik gilt in LTL im Allgemeinen nicht.

2. Spezialfälle des Deduktionstheorems:

   - Falls $A \vdash B$, so $\vdash \Box A \to B$.
   - Falls $A_1, \ldots, A_n \vdash B$, so $\vdash \Box A_1 \to (\Box A_2 \to \ldots \to (\Box A_n \to B) \ldots)$
     und $\vdash \Box A_1 \wedge \ldots \wedge \Box A_n \to B$.

Satz 2.3.4 Seien A, B Formeln, $\mathcal{F}$ eine Menge von Formeln. Falls $\mathcal{F} \vdash \Box A \to B$, so $\mathcal{F} \cup \{A\} \vdash B$.

Bemerkung:

Es gilt auch: Falls $\mathcal{F} \vdash A \to B$, so $\mathcal{F} \cup \{A\} \vdash B$.

Zur Vollständigkeit von Σ_{LTL}

1. LTL ist nicht vollständig axiomatisierbar. Beispiel: Mit

   $$\mathcal{F} = \{A \to B, A \to \bigcirc B, A \to \bigcirc\bigcirc B, A \to \bigcirc\bigcirc\bigcirc B, \ldots\}$$

   gilt

   $$\mathcal{F} \models A \to \Box B, \quad \text{aber (i. Allg.) nicht} \quad \mathcal{F} \models_{\Sigma} A \to \Box B \quad \text{(für beliebiges } \Sigma\text{)}.$$

2. Σ_{LTL} ist ***schwach vollständig*** in folgendem Sinne:

   - $\mathcal{F} \models A \Rightarrow \mathcal{F} \vdash A$, falls A Formel und $\mathcal{F}$ endliche Menge von Formeln.

   Insbesondere gilt für jede Formel A:

   - $\models A \Rightarrow \vdash A$.

   (Die Eigenschaft von Formeln, allgemeingültig zu sein, ist sogar entscheidbar.)

3. Aus der schwachen Vollständigkeit von Σ_{LTL} folgt, dass alle Gesetze (T1), (T2), ... als abgeleitete Formeln bzw. Regeln verwendet werden können.

Weitere Beispiele für Herleitungen

(T32) $\Box\Diamond(A \vee B) \leftrightarrow \Box\Diamond A \vee \Box\Diamond B$

Herleitung der Richtung „→“:

(1)	$\Box\Diamond A \vee \Box\Diamond B \to \Box(\Diamond A \vee \Diamond B)$	(T20)
(2)	$\Diamond A \vee \Diamond B \to \Diamond(A \vee B)$	(T18),(prop)
(3)	$\Box(\Diamond A \vee \Diamond B) \to \Box\Diamond(A \vee B)$	(T19),(2)
(4)	$\Box\Diamond A \vee \Box\Diamond B \to \Box\Diamond(A \vee B)$	(prop),(1),(2)

Für die umgekehrte Richtung genügt es gemäß dem Deduktionstheorem zu zeigen, dass $\Diamond(A \vee B) \vdash \Box\Diamond A \vee \Box\Diamond B$:

(1)	$\Diamond(A \vee B)$	Annahme
(2)	$\Diamond\Box\neg A \rightarrow \Diamond(\Diamond(A \vee B) \wedge \Box\neg A)$	(T29),(1)
(3)	$\Box\neg A \rightarrow (\Diamond(A \vee B) \rightarrow \Diamond(\neg A \wedge (A \vee B)))$	(T29)
(4)	$\Diamond(A \vee B) \wedge \Box\neg A \rightarrow \Diamond(\neg A \wedge (A \vee B))$	(prop),(3)
(5)	$\neg A \wedge (A \vee B) \rightarrow B$	(taut)
(6)	$\Diamond(\Diamond(A \vee B) \wedge \Box\neg A) \rightarrow \Diamond B$	(prop),(T26),(T31),(4),(5)
(7)	$\Diamond\Box\neg A \rightarrow \Diamond B$	(prop),(2),(6)
(8)	$\Box\Diamond\Box\neg A \rightarrow \Box\Diamond B$	(T19),(7)
(9)	$\Box\neg A \rightarrow \Box\Box\neg A$	(prop),(T10)
(10)	$\Diamond\Box\neg A \rightarrow \Diamond\Box\Box\neg A$	(T26),(9)
(11)	$\Diamond\Box\Box\neg A \rightarrow \Box\Diamond\Box\neg A$	(T9)
(12)	$\Diamond\Box\neg A \rightarrow \Box\Diamond B$	(prop),(8),(10),(11)
(13)	$\Box\Diamond A \rightarrow \Box\Diamond B$	(prop),(T2),(T3),(13)

(chain) $A \rightarrow \Diamond B, B \rightarrow \Diamond C \vdash A \rightarrow \Diamond C$

(1)	$A \rightarrow \Diamond B$	Annahme
(2)	$B \rightarrow \Diamond C$	Annahme
(3)	$\Diamond B \rightarrow \Diamond C$	(T31),(2)
(4)	$A \rightarrow \Diamond C$	(prop),(1),(3)

Beispiel einer LTL-Theorie

Sei $Th_{amp} = (\mathcal{L}_{\mathrm{LTL}}(\mathcal{V}_{amp}), \mathcal{A}_{amp})$ mit

$\mathcal{V}_{amp} = \{aus, istrot, istgelb, istgruen\}$,
$\mathcal{A}_{amp}$ bestehend aus:

(A1)	$istgruen \rightarrow \bigcirc(istgelb \vee aus)$
(A2)	$istgelb \rightarrow \bigcirc istrot$
(A3)	$istrot \rightarrow \bigcirc istgruen$
(A4)	$aus \rightarrow \bigcirc istgelb$

Modell von Th_{amp} (z.B.): K mit

	η_0	η_1	η_2	η_3	η_4	η_5	η_6	...
aus	ff	ff	ff	ff	tt	ff	ff	...
$istrot$	ff	ff	tt	ff	ff	ff	tt	...
$istgelb$	ff	tt	ff	ff	ff	tt	ff	...
$istgruen$	tt	ff	ff	tt	ff	ff	ff	...

Intuition: „Zustandsfolge“

grün – gelb – rot – grün – aus – gelb – rot – ...

des Ampelsystems (siehe Einleitung). Formaler Zusammenhang: Siehe Kapitel 5.

2.4 Schwache Vollständigkeit

Beweisziel

Schwache Vollständigkeit von Σ_{LTL}:

$\mathcal{F} \models A \Rightarrow \mathcal{F} \vdash A$ ($\mathcal{F}$ endliche Formelmenge, A Formel).

Es genügt zu zeigen:

(*) $\models A \Rightarrow \vdash A$ (A Formel).

Denn: ($\mathcal{F} = \{A_1, \ldots, A_n\}$)

$$\begin{array}{lll} A_1, \ldots, A_n \models A & \Rightarrow \models \Box A_1 \rightarrow (\Box A_2 \rightarrow \ldots \rightarrow (\Box A_n \rightarrow A) \ldots) & (2.1.6) \\ & \Rightarrow \vdash \Box A_1 \rightarrow (\Box A_2 \rightarrow \ldots \rightarrow (\Box A_n \rightarrow A) \ldots) & (*) \\ & \Rightarrow A_1, \ldots, A_n \vdash A & (2.3.4). \end{array}$$

Definition. Ein ***Positiv-Negativ-Paar (PNP)*** ist ein Paar $\mathcal{P} = (\mathcal{F}^+, \mathcal{F}^-)$ zweier endlicher Mengen $\mathcal{F}^+$ and $\mathcal{F}^-$ von Formeln. Die Menge $\mathcal{F}^+ \cup \mathcal{F}^-$ sei mit $\mathcal{F}_{\mathcal{P}}$ bezeichnet, und die Formel $\hat{\mathcal{P}}$ sei die Abkürzung

$$\hat{\mathcal{P}} \equiv \bigwedge_{A \in \mathcal{F}^+} A \ \wedge \bigwedge_{B \in \mathcal{F}^-} \neg B.$$

(Schreibweise: $\bigwedge_{A \in \mathcal{F}} A \ \equiv \ A_1 \wedge \ldots \wedge A_n$, falls $\mathcal{F} = \{A_1, \ldots, A_n\}$,
für $n = 0$ identifiziert mit der Formel **true**).

Ein PNP $\mathcal{P}$ heißt ***inkonsistent***, falls $\vdash \neg\hat{\mathcal{P}}$. Andernfalls heißt $\mathcal{P}$ ***konsistent***.

Statt (*) wird gezeigt:

(**) Ist $\mathcal{P}$ ein konsistentes PNP, so gibt es eine temporale Struktur K mit $\mathsf{K}_0(\hat{\mathcal{P}}) = \mathsf{tt}$ (***Erfüllbarkeitssatz***).

Aus (**) folgt (*).

Beispiel zur Illustration der Beweisidee

Sei

$$\begin{array}{l} A \ \equiv \ (v_1 \rightarrow v_2 \ ) \ \rightarrow \Box v_3, \\ B \ \equiv \ \bigcirc(v_2 \ \rightarrow v_3) \rightarrow v_1, \\ \mathcal{P} \ = (\{A\}, \{B\}) \qquad (\mathcal{P} \text{ konsistent}, \hat{\mathcal{P}} \ \equiv \ A \wedge \neg B). \end{array}$$

Schritt 1: Gewinne systematischen Überblick darüber, welche „Teilformeln“ (bzgl. $\rightarrow$, $\neg$ und $\Box$) von A and B in einem Zustand „true“ oder „false“ werden sollten, damit in diesem Zustand $\hat{\mathcal{P}}$, d.h. A und $\neg B$, „wahr“ werden, und füge sie zu $\mathcal{F}^+$ bzw. $\mathcal{F}^-$ unter Wahrung der Konsistenz des PNP hinzu:

$$\mathcal{P}_0 = \mathcal{P}^* = (\{A,\ v_1 \rightarrow v_2,\ \Box v_3,\ v_3,\ v_2, \bigcirc(v_2 \rightarrow v_3)\},\ \{B, v_1\})$$

$\mathcal{P}^*$ heißt ***konsistente Vervollständigung*** von $\mathcal{P}$.

Beachte: Im Allgemeinen gibt es mehr als eine konsistente Vervollständigung eines konsistenten PNP.

Schritt 2: Übertrage für alle Formeln mit temporalen Operatoren in $\mathcal{P}^*$ die „Information“ aus $\mathcal{P}^*$ auf ein konsistentes PNP $\sigma(\mathcal{P}^*)$ „für den nächsten Zustand“, wende Schritt 1 auf $\sigma(\mathcal{P}^*)$ an und iteriere diesen Vorgang:

$$\begin{array}{lcll} & \sigma(\mathcal{P}^*) & = & (\{\Box v_3, v_2 \rightarrow v_3\}, \phi), \\ \mathcal{P}_1 & \sigma(\mathcal{P}^*)^* & = & (\{\Box v_3, v_3\}, \{v_2\}), \\ & \sigma(\sigma(\mathcal{P}^*)^*) & = & (\{\Box v_3\}, \phi), \\ \mathcal{P}_2 & \sigma(\sigma(\mathcal{P}^*)^*)^* & = & (\{\Box v_3, v_3\}, \phi), \\ \vdots & & & \\ \text{und so weiter} & & & \\ \vdots & & & \end{array}$$

Grundidee für die temporale Struktur K

Definiere $\mathsf{K} = (\eta_0, \eta_1, \eta_2, \ldots)$ durch

$$\eta_i(v) = \mathsf{tt} \Leftrightarrow v \in \mathcal{F}_i^+ \qquad \text{für alle } v \in \mathcal{V}, i \in \mathbb{N}$$

(wobei $\mathcal{P}_i = (\mathcal{F}_i^+, \mathcal{F}_i^-)$).

Im Beispiel:

	η_0	η_1	η_2	η_3	$\ldots$
v_1	ff	ff	ff	ff	$\ldots$
v_2	tt	ff	ff	ff	$\ldots$
v_3	tt	tt	tt	tt	$\ldots$

Damit gilt: $\mathsf{K}_0(A \wedge \neg B) = \mathsf{tt}$.

Verfeinerung der Konstruktion

Problem ist noch: Die Folge $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ der PNP kann sein wie folgt:

$$
\begin{array}{rcl}
 & & \vdots \\
\mathcal{P}_i & = & (\{\ldots, v, \ldots\}, \{\ldots, \Box v, \ldots\}) \\
 & & \downarrow \sigma \\
 & & (\{\ldots\}, \{\ldots, \Box v, \ldots\}) \\
 & & \downarrow \text{Vervollständigung} \\
\mathcal{P}_{i+1} & = & (\{\ldots, v, \ldots\}, \{\ldots, \Box v, \ldots\}) \\
 & & \downarrow \\
 & & \text{ad infinitum in gleicher Weise } (v \in \mathcal{F}_j^+, \Box v \in \mathcal{F}_j^-)
\end{array}
$$

Dann ist: $\eta_j(v) = \mathsf{tt}$ für $j \geq i$, also $\mathsf{K}_i(\Box v) = \mathsf{tt}$ (Widerspruch).

Lösung:

- Betrachte alle möglichen (Konstruktionen von) Folgen $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ (entsprechend allen möglichen jeweiligen Vervollständigungen).
- Zeige: Es gibt eine Folge, die nicht von obiger Form ist (und die dann für die Definition von K verwendet werden kann).

Skizze der wesentlichen Beweisführung

Angenommen, die obige Situation ergibt sich für alle möglichen Vervollständigungen:

$$
\begin{array}{c}
\mathcal{P}_i = (\{\ldots, A, \ldots\}, \{\ldots, \Box A, \ldots\}) \\
\downarrow \\
\overbrace{\mathcal{P}_{i+1}^1 = (\{\ldots, A, \ldots\}, \{\ldots, \Box A, \ldots\}), \mathcal{P}_{i+1}^2 = (\{\ldots, A, \ldots\}, \{\ldots, \Box A, \ldots\}), \ldots, \mathcal{P}_{i+1}^n = (\{\ldots, A, \ldots\}, \{\ldots, \Box A, \ldots\})} \\
\downarrow \\
\overbrace{\mathcal{P}_{i+2}^1 = (\{\ldots, A, \ldots\}, \{\ldots, \Box A, \ldots\}), \ldots, \mathcal{P}_{i+2}^m = (\{\ldots, A, \ldots\}, \{\ldots, \Box A, \ldots\})} \\
\vdots
\end{array}
$$

Es gilt: Es gibt nur endlich viele verschiedene $\mathcal{P}_j^k$, etwa $\mathcal{P}_1, \ldots, \mathcal{P}_l$. Sei $C \equiv \hat{\mathcal{P}}_1 \vee \ldots \vee \hat{\mathcal{P}}_l$.

Dann ist: $\vdash C \rightarrow \circ C$,
$\vdash C \rightarrow A$.

Mit (ind): $\vdash C \rightarrow \Box A$.

Trivial: $\vdash \hat{\mathcal{P}}_1 \rightarrow C$.

Also: $\vdash \hat{\mathcal{P}}_1 \rightarrow \Box A$ (Widerspruch, da $\hat{\mathcal{P}}_1 \equiv \ldots \wedge \neg\Box A \wedge \ldots$).

Insgesamt: Ist $\mathcal{P}$ ein konsistentes PNP und K definiert wie skizziert, so gilt $\mathsf{K}_0(\hat{\mathcal{P}}) = \mathsf{tt}$.

Kapitel 3
Erweiterungen von LTL

3.1 Binäre Operatoren

Temporale Beziehungen mit binären Operatoren

Die Basissprache von LTL (mit $\bigcirc$, $\Box$, ($\Diamond$) als „Grundausstattung“) kann durch weitere Operatoren erweitert werden, z.B. durch binäre Operatoren wie:

A **until** B:	„Es gibt einen nachfolgenden Zustand, in dem B zutrifft, und A trifft in allen Zuständen bis dahin zu“;
A **unless** B:	„Falls es einen nachfolgenden Zustand gibt, in dem B zutrifft, so trifft A in allen Zuständen bis dahin zu (andernfalls trifft A in allen nachfolgenden Zuständen zu)“;
A **atnext** B:	„A trifft im ersten nachfolgenden Zustand zu, in dem B zutrifft (falls dieser Zustand existiert)“;
A **before** B:	„Falls es einen nachfolgenden Zustand gibt, in dem B zutrifft, so trifft A in einem Zustand davor zu“;

(usw.)

until ist ein ***starker*** Operator (B trifft irgendwann einmal zu); die anderen Operatoren sind ***schwach***.

Formalisierung

Die Semantik von LTL wird wie folgt auf die binären Operatoren erweitert
(K temporale Struktur, $i \in \mathbb{N}$):

- $\mathsf{K}_i(A \textbf{ until } B) = \mathsf{tt} \Leftrightarrow \mathsf{K}_j(B) = \mathsf{tt}$ für ein $j > i$ und $\mathsf{K}_k(A) = \mathsf{tt}$ für alle k, $i < k < j$.
- $\mathsf{K}_i(A \textbf{ unless } B) = \mathsf{tt} \Leftrightarrow \mathsf{K}_j(B) = \mathsf{tt}$ für ein $j > i$ und $\mathsf{K}_k(A) = \mathsf{tt}$ für alle $k, i < k < j$ oder $\mathsf{K}_k(A) = \mathsf{tt}$ für alle $k > i$.
- $\mathsf{K}_i(A \textbf{ atnext } B) = \mathsf{tt} \Leftrightarrow \mathsf{K}_j(B) = \mathsf{ff}$ für alle $j > i$ oder $\mathsf{K}_k(A) = \mathsf{tt}$ für das kleinste $k > i$ mit $\mathsf{K}_k(B) = \mathsf{tt}$.
- $\mathsf{K}_i(A \textbf{ before } B) = \mathsf{tt} \Leftrightarrow$ für jedes $j > i$ mit $\mathsf{K}_j(B) = \mathsf{tt}$ gibt es k, $i < k < j$, mit $\mathsf{K}_k(A) = \mathsf{tt}$.

Beziehungen zwischen den Operatoren

(Notation: Die binären Operatoren binden stärker als die klassischen Operatoren.)

Starke und schwache Operatoren

(T33) A **until** $B \leftrightarrow \bigcirc\Diamond B \wedge A$ **unless** B
(T34) A **unless** $B \leftrightarrow A$ **until** $B \vee \bigcirc\Box A$

Gegenseitige Ausdrückbarkeit (der schwachen Operatoren)

(T35) A **unless** $B \leftrightarrow B$ **atnext** $(A \rightarrow B)$
(T36) A **before** $B \leftrightarrow \neg B$ **atnext** $(A \vee B)$
(T37) A **atnext** $B \leftrightarrow \neg B$ **unless** $(A \wedge B)$
(T38) A **atnext** $B \leftrightarrow B$ **before** $(\neg A \wedge B)$

Weitere allgemeingültige Formeln

Ausdrückbarkeit von $\bigcirc$ und $\Box$

(T39) $\bigcirc A \leftrightarrow A$ **atnext true**
(T40) $\Box A \leftrightarrow A \wedge$ **false atnext** $\neg A$

Fixpunkt-Charakterisierungen

(T41) A **until** $B \leftrightarrow \bigcirc B \vee \bigcirc(A \wedge A$ **until** $B)$
(T42) A **unless** $B \leftrightarrow \bigcirc B \vee \bigcirc(A \wedge A$ **unless** $B)$
(T43) A **atnext** $B \leftrightarrow \bigcirc(B \rightarrow A) \wedge \bigcirc(\neg B \rightarrow A$ **atnext** $B)$
(T44) A **before** $B \leftrightarrow \bigcirc\neg B \wedge \bigcirc(A \vee A$ **before** $B)$

Gesetze für **atnext**

(T45) $\Box A \rightarrow A$ **atnext** B
(T46) $\bigcirc(A$ **atnext** $B) \leftrightarrow \bigcirc A$ **atnext** $\bigcirc B$
(T47) $(A \wedge B)$ **atnext** $C \leftrightarrow A$ **atnext** $C \wedge B$ **atnext** C
(T48) $(A \vee B)$ **atnext** $C \leftrightarrow A$ **atnext** $C \vee B$ **atnext** C
(T49) A **atnext** $(B \vee C) \rightarrow A$ **atnext** $B \vee A$ **atnext** C
(T50) $\Box(A \rightarrow B) \rightarrow (A$ **atnext** $C \rightarrow B$ **atnext** $C)$
(T51) $\Box A \rightarrow (B$ **atnext** $C \rightarrow (A \wedge B)$ **atnext** $(A \wedge C))$

Die Spracherweiterung $\mathcal{L}^{\mathrm{b}}_{\mathrm{LTL}}$

Die Sprache $\mathcal{L}_{\mathrm{LTL}}$ wird durch Hinzunahme binärer Operatoren zur Sprache $\mathcal{L}^{\mathrm{b}}_{\mathrm{LTL}}$ erweitert. Gemäß (T33)–(T38) genügt es, die Sprachdefinition auf einen der binären Operatoren abzustützen.

Sei **op** $\in \{$**until**, **unless**, **atnext**, **before**, $\ldots\}$. Die Sprache $\mathcal{L}^{\mathrm{b}}_{\mathrm{LTL}}$ und ihre Semantik ergibt sich aus $\mathcal{L}_{\mathrm{LTL}}$ durch

- Erweiterung des Alphabets von $\mathcal{L}_{\mathrm{LTL}}$ um **op**;
- Erweiterung der induktiven Formeldefinition um die Regel

  5. Sind A und B Formeln, so ist auch $(A$ **op** $B)$ eine Formel;
- Erweiterung der Semantikdefinition um $\mathsf{K}_i(A$ **op** $B) = \ldots$ (wie oben).

Axiomatisierung

Ein formales System $\Sigma^{\mathrm{b}}_{\mathrm{LTL}}$ für die Logik LTL+b (Lineare Temporale Logik mit binären Operatoren) erhält man durch Erweiterung von Σ_{LTL} um Axiome für den in der Sprachdefinition verwendeten binären Operator **op**.

Die Erweiterung geschieht in jedem Fall durch zwei Axiome:

- stark/schwach-Charakterisierung,
- Fixpoint-Charakterisierung (T41)/.../(T44).

Im Einzelnen sind die Axiome gegeben wie folgt.

atnext als Basisoperator:

(atn1) $\bigcirc\Box\neg B \rightarrow A$ **atnext** B,
(atn2) A **atnext** $B \leftrightarrow \bigcirc(B \rightarrow A) \wedge \bigcirc(\neg B \rightarrow A$ **atnext** $B)$.

until als Basisoperator:

(unt1) A **until** $B \rightarrow \bigcirc\Diamond B$,
(unt2) A **until** $B \leftrightarrow \bigcirc B \vee \bigcirc(A \wedge A$ **until** $B)$.

unless als Basisoperator:

(unl1) $\bigcirc\Box A \rightarrow A$ **unless** B,
(unl2) A **unless** $B \leftrightarrow \bigcirc B \vee \bigcirc(A \wedge A$ **unless** $B)$.

before als Basisoperator:

(bef1) $\bigcirc\Box\neg B \rightarrow A$ **before** B,
(bef2) A **before** $B \leftrightarrow \bigcirc\neg B \wedge \bigcirc(A \vee A$ **before** $B)$.

In jedem Fall gilt: $\Sigma^{\mathrm{b}}_{\mathrm{LTL}}$ ist korrekt und schwach vollständig.

Induktionsregeln für die schwachen binären Operatoren

(indatnext) $A \rightarrow \bigcirc(C \rightarrow B) \wedge \bigcirc(\neg C \rightarrow A) \vdash A \rightarrow B$ **atnext** C
(indunless) $A \rightarrow \bigcirc C \vee \bigcirc(A \wedge B) \vdash A \rightarrow B$ **unless** C
(indbefore) $A \rightarrow \bigcirc\neg C \wedge \bigcirc(A \vee B) \vdash A \rightarrow B$ **before** C

Beispiele für Herleitungen

(*) $\neg(A \textbf{ unless } B) \leftrightarrow \bigcirc\neg B \wedge \bigcirc(\neg A \vee \neg(A \textbf{ unless } B))$

(1)	$A \textbf{ unless } B \leftrightarrow \bigcirc B \vee \bigcirc(A \wedge A \textbf{ unless } B)$	(unl2)
(2)	$\neg(A \textbf{ unless } B) \leftrightarrow \neg\bigcirc B \wedge \neg\bigcirc(A \wedge A \textbf{ unless } B)$	(prop),(1)
(3)	$\neg(A \textbf{ unless } B) \leftrightarrow \bigcirc\neg B \wedge \bigcirc\neg(A \wedge A \textbf{ unless } B)$	(prop),(ltl1),(2)
(4)	$\neg(A \textbf{ unless } B) \leftrightarrow \bigcirc\neg B \wedge \bigcirc(\neg A \vee \neg(A \textbf{ unless } B))$	(prop),(3)

(indunless) $A \rightarrow \bigcirc C \vee \bigcirc(A \wedge B) \vdash A \rightarrow B \textbf{ unless } C$

(1)	$A \rightarrow \bigcirc C \vee \bigcirc(A \wedge B)$	Annahme
(2)	$\neg(B \textbf{ unless } C) \rightarrow \neg\bigcirc C \wedge \bigcirc(\neg B \vee \neg(B \textbf{ unless } C))$	(prop),(*),(ltl1)
(3)	$A \wedge \neg(B \textbf{ unless } C) \rightarrow \bigcirc B$	(prop),(T15),(1),(2)
(4)	$A \wedge \neg(B \textbf{ unless } C) \rightarrow \bigcirc(A \wedge \neg(B \textbf{ unless } C))$	(prop),(T15),(1),(2)
(5)	$A \wedge \neg(B \textbf{ atnext } C) \rightarrow \Box\bigcirc B$	(ind),(3),(4)
(6)	$\bigcirc\Box B \rightarrow B \textbf{ unless } C$	(unl1)
(7)	$A \wedge \neg(B \textbf{ unless } C) \rightarrow B \textbf{ unless } C$	(prop),(T12),(5),(6)
(8)	$A \rightarrow B \textbf{ unless } C$	(prop),(7)

3.2 Past-Operatoren

Aussagen über die Vergangenheit

Erweiterung der Sprache von LTL durch (unäre) ***past***-Operatoren:

- $\ominus A$: „A traf im vorherigen Zustand zu“ (***schwacher previous-Operator***);
- $\boxminus A$: „A traf in allen vorangegangenen Zuständen zu“ (***has-always-been-Operator***).

Die Spracherweiterung $\mathcal{L}^{\mathrm{p}}_{\mathrm{LTL}}$

Die Sprache $\mathcal{L}_{\mathrm{LTL}}$ wird zur Sprache $\mathcal{L}^{\mathrm{p}}_{\mathrm{LTL}}$ erweitert durch

- Erweiterung des Alphabets von $\mathcal{L}_{\mathrm{LTL}}$ um $\ominus$ und $\boxminus$;
- Erweiterung der induktiven Formeldefinition um die Regel

  5. Ist A eine Formel, so sind auch $\ominus A$ und $\boxminus A$ Formeln;

- Erweiterung der Semantikdefinition um (K temporale Struktur, $i \in \mathbb{N}$):
  - $\mathsf{K}_i(\ominus A) = \mathsf{tt} \Leftrightarrow$ falls $i > 0$, so $\mathsf{K}_{i-1}(A) = \mathsf{tt}$.
  - $\mathsf{K}_i(\boxminus A) = \mathsf{tt} \Leftrightarrow \mathsf{K}_j(A) = \mathsf{tt}$ für alle $j \leq i$.

Weitere Operatoren als Abkürzungen:

$$\circledcirc A \equiv \neg\ominus\neg A \qquad (\textbf{\textit{starker previous-Operator}})$$
$$\diamondminus A \equiv \neg\boxminus\neg A \qquad (\textbf{\textit{once-Operator}})$$

Für diese Operatoren gilt:

$$\mathsf{K}_i(\circledcirc A) = \mathsf{tt} \Leftrightarrow i > 0 \text{ und } \mathsf{K}_{i-1}(A) = \mathsf{tt},$$
$$\mathsf{K}_i(\diamondminus A) = \mathsf{tt} \Leftrightarrow \mathsf{K}_j(A) = \mathsf{tt} \text{ für ein } j \leq i.$$

Einige allgemeingültige Formeln

(P1) $\circledcirc A \rightarrow \neg\ominus \textbf{false}$
(P2) $\circledcirc\neg A \rightarrow \neg\circledcirc A$
(P3) $A \rightarrow \ominus\bigcirc A$
(P4) $A \rightarrow \bigcirc\circledcirc A$
(P5) $\ominus(A \rightarrow B) \leftrightarrow \ominus A \rightarrow \ominus B$
(P6) $\ominus(A \wedge B) \leftrightarrow \ominus A \wedge \ominus B$
(P7) $\circledcirc(A \wedge B) \leftrightarrow \circledcirc A \wedge \circledcirc B$

Axiomatisierung

Ein formales System $\Sigma^{\mathrm{p}}_{\mathrm{LTL}}$ für die Logik LTL+p (Lineare Temporale Logik mit past-Operatoren) erhält man durch Erweiterung von Σ_{LTL} um folgende Axiome und Regeln:

(pltl1) $\circledcirc\neg A \rightarrow \neg\circledcirc A$,
(pltl2) $\ominus(A \rightarrow B) \rightarrow (\ominus A \rightarrow \ominus B)$,
(pltl3) $\boxminus A \rightarrow A \wedge \ominus\boxminus A$,
(pltl4) $\diamondminus\ominus \textbf{false}$,
(pltl5) $A \rightarrow \ominus\bigcirc A$,
(pltl6) $A \rightarrow \bigcirc\circledcirc A$.
(prev) $A \vdash \ominus A$,
(indpast) $A \rightarrow B, A \rightarrow \ominus A \vdash A \rightarrow \boxminus B$.

$\Sigma^{\mathrm{p}}_{\mathrm{LTL}}$ ist korrekt und schwach vollständig.

Beispiel für eine Herleitung

(P1) $\boxminus A \rightarrow \neg \ominus \mathbf{false}$ (mit $\boxminus A \equiv \neg \ominus \neg A$)

(1)	$\mathbf{false} \rightarrow \neg A$	(taut)
(2)	$\ominus(\mathbf{false} \rightarrow \neg A)$	(prev),(1)
(3)	$\ominus \mathbf{false} \rightarrow \ominus \neg A$	(pltl2),(prop),(2)
(4)	$\boxminus A \rightarrow \neg \ominus \mathbf{false}$	(prop),(3)

3.3 Auszeichnung des Anfangszustands

Aussagen über den Anfangszustand

Betrachte die Aussage

- „A trifft im Anfangszustand zu“ (*)

(bezogen auf eine gegebene temporale Struktur).

In $\mathcal{L}^{(\mathrm{b})}_{\mathrm{LTL}}$: nicht ausdrückbar.
In $\mathcal{L}^{\mathrm{p}}_{\mathrm{LTL}}$: ausdrückbar durch $\ominus \mathbf{false} \rightarrow A$.

„Geringfügigere“ Erweiterung von LTL, um (*) ausdrücken zu können:

- Ausgezeichnete Formel **init**, die genau im Anfangszustand zutrifft.

(*) wird dann ausgedrückt durch

$\mathbf{init} \rightarrow A$.

Die Spracherweiterung $\mathcal{L}^{\mathrm{i}}_{\mathrm{LTL}}$

Die Sprache $\mathcal{L}_{\mathrm{LTL}}$ wird zur Sprache $\mathcal{L}^{\mathrm{i}}_{\mathrm{LTL}}$ erweitert durch

- Erweiterung des Alphabets von $\mathcal{L}_{\mathrm{LTL}}$ um **init**;
- Erweiterung der induktiven Formeldefinition um die Regel

  5. **init** ist eine Formel;

- Erweiterung der Semantikdefinition um (K temporale Struktur, $i \in \mathbb{N}$):
  - $\mathsf{K}_i(\mathbf{init}) = \mathsf{tt} \;\Leftrightarrow\; i = 0$.

Axiomatisierung

Ein formales System $\Sigma^{\mathrm{i}}_{\mathrm{LTL}}$ für die Logik LTL+i (Lineare Temporale Logik mit Initialisierungs-Formel) erhält man durch Erweiterung von Σ_{LTL} um ein Axiom und eine Regel:

(ltlin) $\bigcirc\neg\mathbf{init}$.

(init) $\mathbf{init} \rightarrow \Box A \vdash A$.

$\Sigma^{\mathrm{i}}_{\mathrm{LTL}}$ ist korrekt und schwach vollständig.

Abgeleitete (Induktions-) Regel:

(indinit) $\mathbf{init} \rightarrow A, A \rightarrow \bigcirc A \vdash A$

Herleitung von (indinit):

(1)	$\mathbf{init} \rightarrow A$	Annahme
(2)	$A \rightarrow \bigcirc A$	Annahme
(3)	$\mathbf{init} \rightarrow \Box A$	(ind2),(1),(2)
(4)	A	(init),(3)

Semantik mit initialer Gültigkeit

Die Möglichkeit, Aussagen auf den Anfangszustand einer temporalen Struktur zu beziehen, kann statt mit den (syntaktischen) Spracherweiterungen „p“ bzw. „i“ auch durch eine andere Definition der Gültigkeit in der Semantik von LTL erreicht werden (***Semantik mit initialer Gültigkeit***; die bisherige Semantik heißt zur Unterscheidung auch ***normale Semantik***).

Die (initiale) Gültigkeit $\models^0_{\mathsf{K}}$ („initial gültig in K“) ist definiert durch:

$$\models^0_{\mathsf{K}} A \;\Leftrightarrow\; \mathsf{K}_0(A) = \mathsf{tt}.$$

Sprechweisen: „initiale Folgerung“ statt „Folgerung“ ($\mathcal{F} \models^0 A$),
„initial allgemeingültig“ statt „allgemeingültig“ ($\models^0 A$).

Bezeichnung: LTL_0 bezeichnet LTL mit der Semantik mit initialer Gültigkeit.

In LTL_0 wird „A trifft im Anfangszustand zu“ ausgedrückt durch die Formel

A.

Normale Semantik und Semantik mit initialer Gültigkeit

$$\begin{array}{lcl}
\models_{\mathsf{K}} A & \Rightarrow & \models^0_{\mathsf{K}} A. \\
\models_{\mathsf{K}} A & \Leftrightarrow & \models^0_{\mathsf{K}} \Box A. \\
\models A & \Leftrightarrow & \models^0 A. \\
\mathcal{F} \models^0 A & \Rightarrow & \mathcal{F} \models A. \\
\mathcal{F} \models A & \Leftrightarrow & \Box\mathcal{F} \models^0 A.
\end{array}$$

(Dabei: $\Box\mathcal{F} = \{\Box B \mid B \in \mathcal{F}\}$.)

3.4 Kombination von Erweiterungen

LTL mit mehreren Erweiterungen

- Die Spracherweiterungen in den vorangegangenen Abschnitten (und andere mögliche, hier nicht behandelte) können auch miteinander kombiniert werden.
- Axiomatisierung von LTL mit mehreren Erweiterungen: Vereinigung der Axiomen- und Regelmengen der einzelnen formalen Systeme.
- In einer solchen Erweiterung sind alle allgemeingültigen Formeln von LTL und den einzelnen Erweiterungen allgemeingültig.

Beispiel:

Erweiterungen „b“ und „i“: Logik LTL+b+i; Sprache $\mathcal{L}_{\mathrm{LTL}}^{\mathrm{bi}}$; formales System (z.B. mit **atnext**):

(taut) Alle aussagenlogisch gültigen Formeln,
(ltl1) $\neg\bigcirc A \leftrightarrow \bigcirc\neg A$,
(ltl2) $\bigcirc(A \rightarrow B) \rightarrow (\bigcirc A \rightarrow \bigcirc B)$,
(ltl3) $\Box A \rightarrow A \wedge \bigcirc\Box A$,
(atn1) $\bigcirc\Box\neg B \rightarrow A$ **atnext** B,
(atn2) A **atnext** $B \leftrightarrow \bigcirc(B \rightarrow A) \wedge \bigcirc(\neg B \rightarrow A$ **atnext** $B)$,
(ltlin) $\bigcirc\neg$**init**.

Regeln:

(mp) $A, A \rightarrow B \vdash B$,
(nex) $A \vdash \bigcirc A$,
(ind) $A \rightarrow B, A \rightarrow \bigcirc A \vdash A \rightarrow \Box B$,
(init) **init** $\rightarrow \Box A \vdash A$.

Binäre past-Operatoren

Bei der Erweiterung LTL+b+p ist es sinnvoll, auch noch weitere binäre Operatoren („in die Vergangenheit“) hinzuzufügen, z.B. (analog zu A **until** B, A **unless** B, A **atnext** B, A **before** B):

A **since** B:	„Es gibt einen vorangegangenen Zustand, in dem B zutraf, und A traf in allen Zuständen seitdem zu“;
A **backto** B:	„Falls es einen vorangegangenen Zustand gibt, in dem B zutraf, so traf A in allen Zuständen seitdem zu (andernfalls traf A in allen vorangegangenen Zuständen zu)“;
A **atlast** B:	„A traf im letzten vorangegangenen Zustand zu, in dem B zutraf (falls dieser Zustand existiert)“;
A **after** B:	„Falls es einen vorangegangenen Zustand gibt, in dem B zutraf, so traf A in einem Zustand danach zu“.

Formalisierung: Analog wie in Abschnitt 3.1.

Kapitel 4
Lineare temporale Prädikatenlogik

4.1 Klassische Prädikatenlogik

Signaturen

Eine ***Signatur*** $SIG = (\mathbf{S}, \mathbf{F}, \mathbf{P})$ ist gegeben durch

- eine Menge $\mathbf{S}$ von ***Sorten***,
- für jedes $\sigma \in \mathbf{S}^*$ und $s \in \mathbf{S}$ eine endliche oder abzählbar unendliche Menge $\mathbf{F}^{(\sigma,s)}$ von ***Funktionszeichen*** (im Fall von $\sigma = \varepsilon$ auch ***Konstanten*** genannt) und $\mathbf{F} = \bigcup_{\sigma\in\mathbf{S}^*, s\in\mathbf{S}} \mathbf{F}^{(\sigma,s)}$,
- für jedes $\sigma \in \mathbf{S}^*$ eine endliche oder abzählbar unendliche Menge $\mathbf{P}^{(\sigma)}$ von ***Prädikatszeichen*** (im Fall von $\sigma = \varepsilon$ auch ***atomare Aussagen*** genannt) und $\mathbf{P} = \bigcup_{\sigma\in\mathbf{S}^*} \mathbf{P}^{(\sigma)}$.

(σ, s) bzw. σ heißt ***Funktionalität*** der jeweiligen Funktions- und Prädikatszeichen.

Schreibweisen: $f^{(\sigma,s)}$ für $f \in \mathbf{F}^{(\sigma,s)}$,
$p^{(\sigma)}$ für $p \in \mathbf{P}^{(\sigma)}$.

Sprache

Sei $SIG = (\mathbf{S}, \mathbf{F}, \mathbf{P})$ eine Signatur. Eine Sprache $\mathcal{L}_{\mathrm{FOL}}(SIG)$ (kurz: $\mathcal{L}_{\mathrm{FOL}}$, Sprache ***erster Stufe***, Sprache der ***(klassischen) Prädikatenlogik (erster Stufe) (FOL)***) ist definiert wie folgt:

Alphabet:

- Alle Zeichen von $\mathbf{F}$ und $\mathbf{P}$,
- für jedes $s \in \mathbf{S}$ eine abzählbar unendliche Menge $\mathcal{X}_s$ von ***(Individuen-) Variablen*** ($\mathcal{X} = \bigcup_{s\in\mathbf{S}} \mathcal{X}_s$),
- das ***Gleichheitszeichen*** $=$,
- die Zeichen **false** $\mid \rightarrow \mid \exists \mid , \mid (\mid)$.

Induktive Definition der ***Terme*** und ***ihrer Sorten***:

1. Jede Variable $x \in \mathcal{X}_s$ ist ein Term der Sorte s.
2. Ist $f \in \mathbf{F}^{(s_1\ldots s_n,s)}$ ein Funktionszeichen und sind t_i Terme der Sorten s_i $(1 \leq i \leq n)$, so ist $f(t_1, \ldots, t_n)$ ein Term der Sorte s.

Eine ***atomare Formel*** ist eine Zeichenreihe einer der beiden Gestalten

- $p(t_1, \ldots, t_n)$, wobei $p \in \mathbf{P}^{(s_1 \ldots s_n)}$ ein Prädikatszeichen und t_i Term der Sorten s_i $(1 \leq i \leq n)$ sind,
- $t_1 = t_2$, wobei t_1 und t_2 Terme gleicher Sorte sind.

Induktive Definition der Formeln:

1. Jede atomare Formel ist eine Formel.
2. **false** ist eine Formel, und sind A und B Formeln, so ist $(A \to B)$ eine Formel.
3. Ist A eine Formel und $x \in \mathcal{X}$, so ist $\exists x A$ eine Formel.

Abkürzungen (zusätzlich zu den Abkürzungen in $\mathcal{L}_{\mathrm{PL}}$):

$\forall x A$ für $\neg \exists \neg A$,
$t_1 \neq t_2$ für $\neg\, t_1 = t_2$.

Schreibweisen: f für Term $f()$, falls f Konstante,
p für atomare Formel $p()$, falls p atomare Aussage,
Infixschreibweisen (u.a.) für Terme und atomare Formeln „wie üblich“.

Das Auftreten einer Variablen x in einer Formel A heißt

- ***gebunden***, wenn es in einem Teil $\exists x B$ von A vorkommt;
- andernfalls heißt es ***frei***.

Eine Formel heißt ***geschlossen***, wenn in ihr keine Variablen frei vorkommen.

Bezeichnung:
$A_x(t)$ bezeichnet die Formel, die sich aus A durch Ersetzen jedes freien Vorkommens der Variablen x durch den Term t ergibt.
(Dabei: x, t von gleicher Sorte, t ohne Variablen, die in A gebunden vorkommen.)

Semantik

Interpretationen für $\mathcal{L}_{\mathrm{FOL}}$ sind gegeben durch ***(prädikatenlogische) Strukturen***.

Eine Struktur S für eine Signatur $SIG = (\mathbf{S}, \mathbf{F}, \mathbf{P})$ ist gegeben durch

- nicht-leere ***Trägermengen*** $|\mathsf{S}|_s$ für alle $s \in \mathbf{S}$ $(|\mathsf{S}| = \bigcup_{s \in \mathbf{S}} |\mathsf{S}|_s)$,
- Abbildungen $f^{\mathsf{S}} : |\mathsf{S}|_{s_1} \times \ldots \times |\mathsf{S}|_{s_n} \to |\mathsf{S}|_s$ für alle Funktionszeichen $f \in \mathbf{F}^{(s_1 \ldots s_n, s)}$,
- Abbildungen $p^{\mathsf{S}} : |\mathsf{S}|_{s_1} \times \ldots \times |\mathsf{S}|_{s_n} \to \{\mathsf{ff}, \mathsf{tt}\}$ für alle Prädikatszeichen $p \in \mathbf{P}^{(s_1 \ldots s_n)}$.

(Beachte: $f^{\mathsf{S}} \in |\mathsf{S}|_s$ für Konstanten $f \in \mathbf{F}^{(\varepsilon, s)}$,
$p^{\mathsf{S}} \in \{\mathsf{ff}, \mathsf{tt}\}$ für atomare Aussagen $p \in \mathbf{P}^{(\varepsilon)}$.)

Eine ***Variablenbelegung*** ξ (bezüglich S) ist eine Zuordnung von je einem Element $\xi(x) \in |\mathsf{S}|_s$ zu jeder Variablen $x \in \mathcal{X}_s$ (für alle $s \in \mathbf{S}$).

Für eine Signatur $SIG = (\mathbf{S}, \mathbf{F}, \mathbf{P})$, eine Struktur S für SIG und eine Variablenbelegung ξ bezüglich S werden $\mathsf{S}^{(\xi)}(t) \in |\mathsf{S}|$ für jeden Term t von $\mathcal{L}_{\mathrm{FOL}}(SIG)$ und $\mathsf{S}^{(\xi)}(A) \in \{\mathsf{ff}, \mathsf{tt}\}$ für jede Formel A von $\mathcal{L}_{\mathrm{FOL}}(SIG)$ definiert wie folgt.

Terme (induktive Definition):

1. $\mathsf{S}^{(\xi)}(x) = \xi(x)$ für $x \in \mathcal{X}$.
2. $\mathsf{S}^{(\xi)}(f(t_1, \ldots, t_n)) = f^{\mathsf{S}}(\mathsf{S}^{(\xi)}(t_1), \ldots, \mathsf{S}^{(\xi)}(t_n))$.

Atomare Formeln:

1. $\mathsf{S}^{(\xi)}(p(t_1, \ldots, t_n)) = p^{\mathsf{S}}(\mathsf{S}^{(\xi)}(t_1), \ldots, \mathsf{S}^{(\xi)}(t_n))$.
2. $\mathsf{S}^{(\xi)}(t_1 = t_2) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{S}^{(\xi)}(t_1)$ und $\mathsf{S}^{(\xi)}(t_2)$ sind gleiche Elemente in $|\mathsf{S}|_s$ (wobei s die Sorte von t_1 und t_2 ist).

Formeln (induktive Definition):

1. $\mathsf{S}^{(\xi)}(A)$ für atomare Formeln ist bereits definiert.
2. $\mathsf{S}^{(\xi)}(\mathbf{false}) = \mathsf{ff}$.
3. $\mathsf{S}^{(\xi)}(A \to B) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{S}^{(\xi)}(A) = \mathsf{ff}$ oder $\mathsf{S}^{(\xi)}(B) = \mathsf{tt}$.
4. $\mathsf{S}^{(\xi)}(\exists x A) = \mathsf{tt} \;\Leftrightarrow$ es gibt Variablenbelegung ξ' mit $\xi \sim_x \xi'$ und $\mathsf{S}^{(\xi')}(A) = \mathsf{tt}$. (Dabei: $\xi \sim_x \xi' \;\Leftrightarrow\; \xi(y) = \xi'(y)$ für alle von x verschiedenen Variablen y.)

Für $\forall x A$ erhält man dann noch:

5. $\mathsf{S}^{(\xi)}(\forall x A) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{S}^{(\xi')}(A) = \mathsf{tt}$ für alle ξ' mit $\xi \sim_x \xi'$.

Bemerkung:

$\mathsf{S}^{(\xi)}(A)$ hängt für geschlossene Formeln A nicht von der Variablenbelegung ξ ab.

Die Gültigkeit $\models_{\mathsf{S}}$ („gültig in S“) ist definiert durch:

$$\models_{\mathsf{S}} A \;\Leftrightarrow\; \mathsf{S}^{(\xi)}(A) = \mathsf{tt} \text{ für jede Variablenbelegung } \xi.$$

Beispiele:

Allgemeingültige Formeln:	$\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$,
	$\forall x A \to A_x(t)$.
Folgerungen:	$A \to B, B \to C \models A \to C$,
	$A \models \forall x A$,
	$t_1 = t_2, t_2 = t_3 \models t_1 = t_3$.

Bemerkung:

Eine Sprache $\mathcal{L}_{\mathrm{PL}}(\mathcal{V})$ der klassischen Aussagenlogik kann aufgefasst werden als Sprache $\mathcal{L}_{\mathrm{FOL}}(SIG)$ mit $SIG = (\emptyset, \emptyset, \mathbf{P})$, $\mathbf{P} = \mathbf{P}^{(\varepsilon)} = \mathcal{V}$; die Gültigkeitsbegriffe sind jeweils gleich.

Axiomatisierung

Ein mögliches (korrektes und vollständiges) formales System Σ_{FOL} ist gegeben durch:

Axiome:

- Alle Axiome von Σ_{PL},
- $A_x(t) \rightarrow \exists x A$,
- $x = x$,
- $x = y \rightarrow (A \rightarrow A_x(y))$.

Regeln:

- $A, A \rightarrow B \vdash B$,
- $A \rightarrow B \vdash \exists x A \rightarrow B$, falls x in B nicht frei vorkommt (***Partikularisierung***).

Bemerkung:

Das Deduktionstheorem von Σ_{PL} gilt für Σ_{FOL} nur in eingeschränkter Form, z.B.:

$$\mathcal{F} \cup \{A\} \vdash B \;\Rightarrow\; \mathcal{F} \vdash A \rightarrow B, \quad \text{falls } A \text{ geschlossen.}$$

Beispiele für FOL-Theorien

1. FOL-Theorien für das „Behälter-System“ (siehe Einleitung):
   (a) Die PL-Theorie Th_{beh} aus Abschnitt 1.3 kann auch als FOL-Theorie aufgefasst werden.
   (b) Allgemeinere Formalisierung (einheitlich für beliebig viele Behälter):

   $Th'_{beh} = (\mathcal{L}_{\mathrm{FOL}}(SIG_{beh}), \mathcal{A}'_{beh})$ mit
   - $SIG_{beh} = (\{GEGENST, BEH, FARBE\}, \mathbf{F}, \mathbf{P})$ und
     $\mathbf{F} = \{rot^{(\varepsilon, FARBE)}, blau^{(\varepsilon, FARBE)}, gelb^{(\varepsilon, FARBE)}, farbe^{(GEGENST, FARBE)}\}$,
     $\mathbf{P} = \{istK^{(GEGENST)}, istW^{(GEGENST)}, enth^{(BEH, GEGENST)}\}$,
   - $\mathcal{A}'_{beh}$ bestehend aus:
     - $enth(x, y) \wedge farbe(y) = blau \rightarrow \neg(enth(x, z) \wedge farbe(z) = rot)$,
     - $\forall x \exists y (enth(x, y) \wedge istW(y))$,
     - $\exists x \neg(enth(x, y) \wedge istW(y) \wedge (farbe(y) = gelb \vee farbe(y) = blau))$.

2. Eine Theorie der natürlichen Zahlen (***Peano-Arithmetik***):

   $Nat = (\mathcal{L}_{\mathrm{FOL}}(SIG_{Nat}), \mathcal{N})$ mit
   - $SIG_{Nat} = (\{NAT\}, \mathbf{F}, \emptyset)$ und
     $\mathbf{F} = \{0^{(\varepsilon, NAT)}, s^{(NAT, NAT)}, +^{(NAT\ NAT, NAT)}, *^{(NAT\ NAT, NAT)}\}$,

- $\mathcal{N}$ bestehend aus:
  - $s(x) = s(y) \rightarrow x = y$,
  - $x + 0 = x$,
  - $x + s(y) = s(x + y)$,
  - $x * 0 = 0$,
  - $x * s(y) = (x * y) + x$,
  - $(A_x(0) \wedge \forall x(A \rightarrow A_x(s(x)))) \rightarrow \forall x A$.

Standardmodell N von Nat: $|\mathsf{N}| = |\mathsf{N}|_{NAT} = \mathbb{N}$,
$0^{\mathsf{N}} =$ „null“,
$s^{\mathsf{N}} =$ „plus eins“,
$+^{\mathsf{N}} =$ „plus“,
$*^{\mathsf{N}} =$ „mal“.

3. Eine Theorie der „Stacks“:

$Stack = (\mathcal{L}_{\mathrm{FOL}}(SIG_{st}), \mathcal{S})$ mit

- $SIG_{st} = (\{OBJ, STACK\}, \mathbf{F}, \emptyset)$ und
  $\mathbf{F} = \{EMPTY^{(\varepsilon, STACK)}, PUSH^{(STACK\ OBJ, STACK)}, POP^{(STACK, STACK)}, TOP^{(STACK, OBJ)}\}$,
- $\mathcal{S}$ bestehend aus:
  - $PUSH(s, x) \neq EMPTY$,
  - $POP(PUSH(s, x)) = s$,
  - $TOP(PUSH(s, x)) = x$.

Modelle S von $Stack$: $|\mathsf{S}|_{OBJ} =$ Menge von Objekten,
$|\mathsf{S}|_{STACK} =$ Keller von Objekten aus $|\mathsf{S}|_{OBJ}$,
$EMPTY^{\mathsf{S}} =$ „leerer Keller“,
$PUSH^{\mathsf{S}} =$ „push-Operation auf Keller“,
$POP^{\mathsf{S}} =$ „pop-Operation auf Keller“,
$TOP^{\mathsf{S}} =$ „oberstes Kellerelement“.

(Informatiksprechweise: $Stack$ ist eine ***(algebraische) Spezifikation*** des ***Datentyps*** „Stack“, d.h. von Modellen der angegebenen Art.)

4.2 Sprache und Semantik der linearen temporalen Prädikatenlogik

Temporale Signaturen

Eine ***temporale Signatur*** $TSIG = (SIG, \mathcal{X}^F, \mathcal{V}^F)$ ist gegeben durch

- eine Signatur $SIG = (\mathbf{S}, \mathbf{F}, \mathbf{P})$,
- für jedes $s \in \mathbf{S}$ eine endliche oder abzählbar unendliche Menge $\mathcal{X}_s^F$ von ***flexiblen Individuensymbolen*** und $\mathcal{X}^F = \bigcup_{s \in \mathbf{S}} \mathcal{X}_s^F$,
- eine endliche oder abzählbar unendliche Menge $\mathcal{V}^F$ von ***flexiblen Aussagensymbolen***.

Für $TSIG = (SIG, \mathcal{X}^F, \mathcal{V}^F)$ sei SIG^+ die Signatur, die aus $SIG = (\mathbf{S}, \mathbf{F}, \mathbf{P})$ entsteht, wenn man $\mathbf{F}^{(\varepsilon,s)}$ durch $\mathbf{F}^{(\varepsilon,s)} \cup \mathcal{X}_s^F$ (für jedes $s \in \mathbf{S}$) und $\mathbf{P}^{(\varepsilon)}$ durch $\mathbf{P}^{(\varepsilon)} \cup \mathcal{V}^F$ ersetzt.

Bemerkung:

Flexible (Individuen- bzw. Aussagen-) Symbole

- spielen technisch die Rolle von Konstanten bzw. atomaren Aussagen;
- werden oft auch ***flexible (...-) Variablen*** genannt.

Sprache

Sei $TSIG = (SIG, \mathcal{X}^F, \mathcal{V}^F)$ eine temporale Signatur und $\mathcal{L}_{\mathrm{FOL}}(SIG^+)$ eine Sprache erster Stufe (SIG^+ definiert wie oben) mit der Variablenmenge $\mathcal{X} = \bigcup_{s \in \mathbf{S}} \mathcal{X}_s$. Eine (Basis-) Sprache $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$ (kurz: $\mathcal{L}_{\mathrm{FOLTL}}$) der ***(linearen) temporalen Prädikatenlogik (erster Stufe) (FOLTL)*** ist definiert wie folgt:

Alphabet:

- Alle Zeichen von $\mathcal{L}_{\mathrm{FOL}}(SIG^+)$,
- die Zeichen $\circ$ und $\Box$.

Terme (mit ***ihren Sorten***) und ***atomare Formeln*** von $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$ sind die Terme und atomaren Formeln von $\mathcal{L}_{\mathrm{FOL}}(SIG^+)$.
Insbesondere: Jedes $a \in \mathcal{X}^F$ ist ein Term, und jedes $v \in \mathcal{V}^F$ ist eine atomare Formel.

Induktive Definition der Formeln:

1. Jede atomare Formel ist eine Formel.
2. **false** ist eine Formel, und sind A und B Formeln, so sind $(A \rightarrow B)$, $\circ A$ und $\Box A$ Formeln.
3. Ist A eine Formel und $x \in \mathcal{X}$, so ist $\exists x A$ eine Formel.

Freie und gebundene Variablen, geschlossene Formeln, Abkürzungen, Schreibweisen: wie in $\mathcal{L}_{\mathrm{LTL}}$ und $\mathcal{L}_{\mathrm{FOL}}$. (Außerdem: zusätzliche Klammern zur besseren Lesbarkeit, z.B. $\circ(a < b)$ statt $\circ\, a < b$.)

$\mathcal{L}_{\mathrm{FOL}}(SIG^+)$ heißt ***Kern*** von $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$ und wird auch mit $ker(\mathcal{L}_{\mathrm{FOLTL}})$ bezeichnet. (Jede Formel von $ker(\mathcal{L}_{\mathrm{FOLTL}})$ ist auch Formel von $\mathcal{L}_{\mathrm{FOLTL}}$.)

Semantik

Interpretationen für $\mathcal{L}_{\mathrm{FOLTL}}$ sind (wie bei $\mathcal{L}_{\mathrm{LTL}}$) gegeben durch ***temporale Strukturen*** mit folgender neuer Definition.

Eine temporale Struktur $\mathsf{K} = (\mathsf{S}, \mathsf{W})$ für eine temporale Signatur $TSIG = (SIG, \mathcal{X}^F, \mathcal{V}^F)$ ist gegeben durch

- eine Struktur S für SIG,
- eine unendliche Folge $\mathsf{W} = (\eta_0, \eta_1, \eta_2, \ldots)$ von Abbildungen

  $$\eta_i : \mathcal{X}^F \cup \mathcal{V}^F \;\rightarrow\; |\mathsf{S}| \cup \{\mathsf{ff}, \mathsf{tt}\}$$

  mit

  $$\eta_i(a) \in |\mathsf{S}|_s \text{ für } a \in \mathcal{X}^F_s,\ s \in \mathsf{S},$$
  $$\eta_i(v) \in \{\mathsf{ff}, \mathsf{tt}\} \text{ für } v \in \mathcal{V}^F$$

  für alle $i \in \mathbb{N}$. (Die η_i heißen wieder ***Zustände***, η_0 heißt ***Anfangszustand***.)

Für eine temporale Signatur $TSIG = (SIG, \mathcal{X}^F, \mathcal{V}^F)$, eine temporale Struktur $\mathsf{K} = (\mathsf{S}, \mathsf{W})$ für $TSIG$, eine Variablenbelegung ξ bezüglich S und jedes $\eta_i \in \mathsf{W}$ werden $\mathsf{S}^{(\xi,\eta_i)}(t) \in |\mathsf{S}|$ für jeden Term t von $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$ und $\mathsf{S}^{(\xi,\eta_i)}(A) \in \{\mathsf{ff}, \mathsf{tt}\}$ für jede atomare Formel A von $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$ definiert analog wie in FOL, z.B.:

- $\mathsf{S}^{(\xi,\eta_i)}(x) = \xi(x)$ für $x \in \mathcal{X}$.
- $\mathsf{S}^{(\xi,\eta_i)}(a) = \eta_i(a)$ für $a \in \mathcal{X}^F$.
- $\mathsf{S}^{(\xi,\eta_i)}(v) = \eta_i(v)$ für $v \in \mathcal{V}^F$.

usw.

Induktive Definition von $\mathsf{K}_i^{(\xi)}(F) \in \{\mathsf{ff}, \mathsf{tt}\}$ für jede Formel F von $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$ und jedes $i \in \mathbb{N}$:

1. $\mathsf{K}_i^{(\xi)}(A) = \mathsf{S}^{(\xi,\eta_i)}(A)$ für atomare Formeln A.
2. $\mathsf{K}_i^{(\xi)}(\mathbf{false}) = \mathsf{ff}$.
3. $\mathsf{K}_i^{(\xi)}(A \rightarrow B) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_i^{(\xi)}(A) = \mathsf{ff}$ oder $\mathsf{K}_i^{(\xi)}(B) = \mathsf{tt}$.
4. $\mathsf{K}_i^{(\xi)}(\circ A) = \mathsf{K}_{i+1}^{(\xi)}(A)$.
5. $\mathsf{K}_i^{(\xi)}(\Box A) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_j^{(\xi)}(A) = \mathsf{tt}$ für alle $j \geq i$.
6. $\mathsf{K}_i^{(\xi)}(\exists x A) = \mathsf{tt} \;\Leftrightarrow$ es gibt Variablenbelegung ξ' mit $\xi \sim_x \xi'$ und $\mathsf{K}_i^{(\xi')}(A) = \mathsf{tt}$.

Bemerkungen:

1. Die Abbildung $\mathsf{S}^{(\xi,\eta_i)}$ kann (wie in FOL) auch für sich allein auf alle Formeln von $ker(\mathcal{L}_{\mathrm{FOLTL}})$ (d.h. $\mathcal{L}_{\mathrm{FOL}}(SIG^+)$) erweitert werden. Gemäß der Definition von $\mathsf{K}_i^{(\xi)}(A)$ gilt dann:

   $\mathsf{K}_i^{(\xi)}(A) = \mathsf{S}^{(\xi,\eta_i)}(A)$ für jede Formel A von $ker(\mathcal{L}_{\mathrm{FOLTL}})$.

2. Für die weiteren (als Abkürzungen eingeführten) Formeln A ergibt sich $\mathsf{K}_i^{(\xi)}(A)$ wie bisher.

Die Gültigkeit $\models_{\mathsf{K}}$ („gültig in K“) ist definiert durch:

$$\models_{\mathsf{K}} A \;\Leftrightarrow\; \mathsf{K}_i^{(\xi)}(A) = \mathsf{tt} \text{ für alle } i \in \mathbb{N} \text{ und alle Variablenbelegungen } \xi.$$

Beispiel

Seien $TSIG = (SIG_{Nat}, \{a, b\}, \{v\})$ temporale Signatur (mit einer geeigneten Signatur SIG_{Nat} für natürliche Zahlen) und x, y Variablen (der Sorte NAT). Dann ist

$$A \;\equiv\; \exists x(a = x + y) \wedge \circ v \rightarrow \Box(b \leq 7)$$

eine Formel von $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$. $\mathsf{K} = (\mathsf{N}, \mathsf{W})$ sei eine temporale Struktur für TSIG mit dem Standardmodell N der natürlichen Zahlen und W mit:

	η_0	η_1	η_2	η_3	η_4	...
a	2	8	5	7	3	... (beliebig) ...
b	4	7	9	5	5	... (unverändert 5) ...
v	tt	tt	ff	tt	tt	... (beliebig) ...

ξ sei eine Variablenbelegung mit $\xi(y) = 3$. Dann ist

$$\mathsf{K}_i^{(\xi)}(\exists x(a = x + y)) = \mathsf{tt} \;\Leftrightarrow\; \text{es gibt } \xi' \text{ mit } \xi \sim_x \xi' \text{ und } \eta_i(a) = \xi'(x) + 3,$$

also $\mathsf{K}_0^{(\xi)}(\exists x(a = x + y)) = \mathsf{ff}$ und $\mathsf{K}_2^{(\xi)}(\exists x(a = x + y)) = \mathsf{tt}$ (mit $\xi'(x) = 2$). Somit:

$\mathsf{K}_0^{(\xi)}(A) = \mathsf{tt}$,
$\mathsf{K}_1^{(\xi)}(\circ v) = \eta_2(v) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_1^{(\xi)}(A) = \mathsf{tt}$,
$\mathsf{K}_2^{(\xi)}(\circ v) = \eta_3(v) = \mathsf{tt}, \mathsf{K}_2^{(\xi)}(b \leq 7) = \mathsf{ff}, \mathsf{K}_2^{(\xi)}(\Box(b \leq 7)) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_2^{(\xi)}(A) = \mathsf{ff}$,
$\mathsf{K}_i^{(\xi)}(\Box(b \leq 7)) = \mathsf{tt} \;\Rightarrow\; \mathsf{K}_i^{(\xi)}(A) = \mathsf{tt}$ für $i \geq 3$.

Insbesondere: A ist nicht gültig in K.

Weitere temporallogische Gesetze

(T52) $\exists x \circ A \leftrightarrow \circ \exists x A$
(T53) $\forall x \circ A \leftrightarrow \circ \forall x A$
(T54) $\exists x \Diamond A \leftrightarrow \Diamond \exists x A$
(T55) $\forall x \Box A \leftrightarrow \Box \forall x A$

Einsetzbarkeit von Termen

Definition. Sei A eine Formel von $\mathcal{L}_{\mathrm{FOLTL}}$. Ein Term t heißt ***einsetzbar für*** x ***in*** A, wenn $A_x(t)$ gegenüber A keine neuen Vorkommen flexibler Symbole im „Wirkungsbereich“ eines temporalen Operators hat.

Beispiel:

Der Term $b \in \mathcal{X}^F$ ist einsetzbar für x in

$$x = a \wedge \bigcirc(y \neq a),$$

nicht jedoch in

$$x = a \wedge \bigcirc(x \neq a).$$

Lemma 4.2.1 Ist t einsetzbar für x in A, so ist die Formel $A_x(t) \rightarrow \exists x A$ allgemeingültig.

4.3 Axiomatisierung

Das formale System Σ_{FOLTL}

Ein mögliches formales System Σ_{FOLTL} für FOLTL ist gegeben durch:

Axiome: Alle Axiome von Σ_{LTL} sowie zusätzlich

(ltl4) $A_x(t) \rightarrow \exists x A$, falls t für x in A einsetzbar ist,

(ltl5) $\bigcirc\exists x A \rightarrow \exists x \bigcirc A$,

(ltl6) $A \rightarrow \bigcirc A$, falls A keine flexiblen Symbole enthält,

(eq1) $x = x$,

(eq2) $x = y \rightarrow (A \rightarrow A_x(y))$, falls A Formel von $ker(\mathcal{L}_{\mathrm{FOLTL}})$ ist.

Regeln: Alle Regeln von Σ_{LTL} sowie zusätzlich

(par) $A \rightarrow B \vdash \exists x A \rightarrow B$, falls x in B nicht frei vorkommt.

Bemerkungen:

1. Σ_{FOLTL} ist korrekt.
2. Abgeleitete Regeln von Σ_{FOL} können auch in Herleitungen in Σ_{FOLTL} verwendet werden (Kennzeichnung: (pred)), z.B.:

   $A \vdash \forall x A$,
   $A \rightarrow B \vdash A \rightarrow \forall x B$, falls x in A nicht frei vorkommt,
   $t_1 = t_2, t_2 = t_3 \vdash t_1 = t_3$.

3. Es gilt:

$$\mathcal{F} \cup \{A\} \vdash B \Leftrightarrow \mathcal{F} \vdash \Box A \rightarrow B, \quad \text{falls } A \text{ geschlossen.}$$

4. FOLTL kann nicht einmal schwach vollständig axiomatisiert werden.

Beispiel für eine Herleitung

(T54) $\forall x \Box A \leftrightarrow \Box \forall x A$

(1)	$\neg \Box A \rightarrow \exists x \neg \Box A$	(ltl4)
(2)	$\Box A \rightarrow \bigcirc \Box A$	(prop),(ltl3)
(3)	$\forall x \Box A \rightarrow \bigcirc \Box A$	(prop),(1),(2)
(4)	$\forall x \Box A \rightarrow \forall x \bigcirc \Box A$	(pred),(3)
(5)	$\bigcirc \exists x \neg \Box A \rightarrow \exists x \bigcirc \neg \Box A$	(ltl5)
(6)	$\bigcirc \exists x \neg \Box A \rightarrow \exists x \neg \bigcirc \Box A$	(pred),(ltl1),(5)
(7)	$\forall x \bigcirc \Box A \rightarrow \bigcirc \forall x \Box A$	(prop),(ltl1),(6)
(8)	$\forall x \Box A \rightarrow \bigcirc \forall x \Box A$	(prop),(4),(7)
(9)	$\forall x \Box A \rightarrow A$	(prop),(ltl3),(1)
(10)	$\forall x \Box A \rightarrow \forall x A$	(pred),(9)
(11)	$\forall x \Box A \rightarrow \Box \forall x A$	(ind),(8),(10)
(12)	$\neg A \rightarrow \exists x \neg A$	(ltl4)
(13)	$\Box \forall x A \rightarrow \Box A$	(prop),(T19),(12)
(14)	$\Box \forall x A \rightarrow \forall x \Box A$	(pred),(13)
(15)	$\forall x \Box A \leftrightarrow \Box \forall x A$	(prop),(11),(14)

4.4 Erweiterungen

Aussagenlogische Erweiterungen

Aussagenlogische Erweiterungen wie in LTL („b“, „p“, „i“) sind in FOLTL ebenso möglich und analog zu handhaben, z.B.:

Die Sprache $\mathcal{L}^{\mathrm{b}}_{\mathrm{FOLTL}}$ für die Logik FOLTL+b enthält Formeln wie

$\exists x (A$ **atnext** $B)$,
A **unless** $(\forall x B)$
usw.

(Axiomatisierung: mit den betreffenden Axiomen aus Abschnitt 3.1, z.B. (atn1) und (atn2).)

Weitere Gesetze:

(T56) $\exists x (A$ **atnext** $B) \leftrightarrow (\exists x A)$ **atnext** B, falls x in B nicht frei vorkommt
(T57) $\forall x (A$ **atnext** $B) \leftrightarrow (\forall x A)$ **atnext** B, falls x in B nicht frei vorkommt
usw.

Gestrichene Individuensymbole

Idee: Für $a \in \mathcal{X}^F$ wird die Aussage

„Der Wert von a im nächsten Zustand ist um 1 erhöht“

formal ausgedrückt durch

$$\exists x(\circ(a = x) \wedge x = a + 1).$$

„Bequemer“ wäre:

$$a' = a + 1$$

mit der informellen Bedeutung von a': „a im nächsten Zustand“
(analog zu $\circ v$ für $v \in \mathcal{V}^F$).

Formal: Die Sprache $\mathcal{L}_{\mathrm{FOLTL}}$ wird erweitert zu $\mathcal{L}_{\mathrm{FOLTL}'}$ (für die Logik FOLTL$'$) durch

- Erweiterung des Alphabets von $\mathcal{L}_{\mathrm{FOLTL}}$ um das Zeichen $'$;
- Erweiterung der induktiven Termdefinition um die Regel

  3. Für jedes $a \in \mathcal{X}^F$ ist a' ein Term;

- Erweiterung der Semantikdefinition um $\mathsf{S}^{(\xi,\eta_i)}(a') = \eta_{i+1}(a)$.

Satz 4.4.1 Zu jeder Formel A von $\mathcal{L}_{\mathrm{FOLTL}'}$ gibt es eine Formel A^* von $\mathcal{L}_{\mathrm{FOLTL}}$ mit $\vDash A \leftrightarrow A^*$.

Folgerungen:

1. FOLTL$'$ ist keine „echte“ Erweiterung von FOLTL, lediglich „bequemer“.
2. Die Formel $A \leftrightarrow A^*$ kann zu Σ_{FOLTL} für eine Axiomatisierung von FOLTL$'$ hinzugenommen werden.

Verwendung im Folgenden:

- FOLTL$'$ wird statt FOLTL verwendet, wo immer dies „bequem“ ist.
- Gemeinsame Bezeichnung: FOLTL (ohne Unterscheidung):
- Zur Vereinheitlichung wird für $v \in \mathcal{V}^F$ auch v' statt $\circ v$ geschrieben.

Die Logik FOLTL+w

Definition. Eine binäre Relation $\prec$ auf einer Menge M heißt ***fundiert***, falls es keine unendliche Folge $e_0, e_1, e_2, \ldots$ von Elementen von M gibt mit $e_{i+1} \prec e_i$ für alle $i \in \mathbb{N}$.

Beispiel: Die „kleiner“-Relation auf $\mathbb{N}$ ist fundiert (nicht jedoch auf $\mathbb{Z}$ oder $\mathbb{R}$).

Eine Sprache $\mathcal{L}^{\mathrm{w}}_{\mathrm{FOLTL}}(TSIG)$, $TSIG = (SIG, \mathcal{X}^F, \mathcal{V}^F)$, für die Logik FOLTL+w ist eine Sprache $\mathcal{L}_{\mathrm{FOLTL}}(TSIG)$ mit:

- SIG enthält eine ausgezeichnete Sorte WF sowie ein Prädikatszeichen $\prec^{(WF\ WF)}$;
- In jeder temporalen Struktur $\mathsf{K} = (\mathsf{S}, \mathsf{W})$ für $TSIG$ ist $\prec^{\mathsf{S}}$ eine fundierte Relation auf $|\mathsf{S}|_{WF}$.

Ein formales System $\Sigma^{\mathrm{w}}_{\mathrm{FOLTL}}$ für FOLTL+w ergibt sich durch Erweiterung von Σ_{FOLTL} um das Axiom ($z, \bar{z} \in \mathcal{X}_{WF}$)

(wfr) $\quad A \to \Diamond(B \vee \exists \bar{z}(\bar{z} \prec z \wedge A_z(\bar{z}))) \vdash \exists z A \to \Diamond B, \quad$ falls z nicht in B vorkommt.

Beispiel einer Theorie

Sei $Th_{pot2} = (\mathcal{L}^{\mathrm{i}}_{\mathrm{FOLTL}}(TSIG), \mathcal{A}_{pot2})$ mit $TSIG = (SIG_{NAT}, \mathcal{X}^F, \emptyset)$ mit

$\mathcal{X}^F = \mathcal{X}^F_{NAT} = \{a\}$,
$\mathcal{A}_{pot2}$ bestehend aus:

- Axiomen „für $\mathbb{N}$“,
- **init** $\to a = 1$,
- $\Box(a' = 2 * a)$.

($\mathcal{L}^{\mathrm{i}}_{\mathrm{FOLTL}}$ ist Sprache der Logik FOLTL+i (genauer: FOLTL′+i); Erweiterungen können wie bei LTL kombiniert werden.)

Modell von Th_{pot2}: $\mathsf{K} = (\mathsf{N}, \mathsf{W} = (\eta_0, \eta_1, \eta_2, \ldots))$ mit „passendem N“ und

$\eta_j(a) = 2^j$ für alle $j \in \mathbb{N}$.

Kapitel 5
Spezifikation und Verifikation von Zustandssystemen

5.1 Transitionssysteme

Formalisierung von Zustandssystemen

Der informelle Begriff der Zustandssysteme (siehe Einleitung) wird formalisiert durch den Begriff der Transitionssysteme.

Definition. Sei $SIG = (\mathbf{S}, \mathbf{F}, \mathbf{P})$ eine Signatur und S eine Struktur für SIG. Ein ***Transitionssystem (Zustandsübergangssystem) 1. Stufe*** (kurz: ***STS***) $\Gamma = (X, V, Z, T)$ über SIG und S ist gegeben durch:

- eine endliche oder abzählbar unendliche Menge X_s für jedes $s \in \mathbf{S}$ und $X = \bigcup_{s \in \mathbf{S}} X_s$,
- eine endliche oder abzählbar unendliche Menge V,
- eine Menge Z von ***(System-) Zuständen*** $\eta : X \cup V \to |\mathsf{S}| \cup \{\mathsf{ff}, \mathsf{tt}\}$ mit $\eta(a) \in |\mathsf{S}|_s$ für $a \in X_s, s \in \mathbf{S}$ und $\eta(v) \in \{\mathsf{ff}, \mathsf{tt}\}$ für $v \in V$,
- eine totale binäre Relation $T \subseteq Z \times Z$ (***Transitionsrelation***).

Elemente von $X \cup V$ heißen ***Systemvariablen (Zustandsvariablen)***. Ein ***Ablauf*** von Γ ist eine unendliche Folge $\mathsf{W} = (\eta_0, \eta_1, \eta_2, \ldots)$ von Systemzuständen mit $(\eta_i, \eta_{i+1}) \in T$ für alle $i \in \mathbb{N}$.

Bemerkungen:

- Eine binäre Relation R über einer Menge M heißt ***total***, wenn es zu jedem $e_1 \in M$ ein $e_2 \in M$ gibt mit $(e_1, e_2) \in R$.
- Ein STS mit $X = \emptyset$ heißt ***propositional***. (In diesem Fall sind SIG und S irrelevant und können in der Definition weggelassen werden.)

Schreibweisen: $\Gamma(SIG, \mathsf{S})$ für ein STS Γ über SIG und S,
$SIG_\Gamma, \mathbf{S}_\Gamma, \mathbf{F}_\Gamma, \mathbf{P}_\Gamma, \mathsf{S}_\Gamma, X_\Gamma, V_\Gamma, Z_\Gamma, T_\Gamma, \mathsf{W}_\Gamma$
für die einzelnen „Bestandteile“ eines gegebenen $\Gamma(SIG, \mathsf{S})$,
$\eta_0 \longrightarrow \eta_1 \longrightarrow \eta_2 \longrightarrow \ldots$ für Ablauf $(\eta_0, \eta_1, \eta_2, \ldots)$.

Beispiele

1. Zähler, der ein- und ausgeschaltet werden kann: $\Gamma_{count}(SIG_{Nat}, \mathsf{N}) = (X, V, Z, T)$ mit:

$$
\begin{aligned}
&X = X_{NAT} = \{c\},\\
&V = \{ein\},\\
&Z = \{\eta : X \cup V \to \mathbb{N} \cup \{\mathsf{ff}, \mathsf{tt}\} \mid \eta(c) \in \mathbb{N}, \eta(ein) \in \{\mathsf{ff}, \mathsf{tt}\}\},\\
&\qquad\qquad \text{(Notation für } \eta \in Z\text{: } [\eta(ein), \eta(c)]\text{.)}\\
&T = \{([\mathsf{tt}, n], [\mathsf{tt}, n+1]), ([\mathsf{tt}, n], [\mathsf{ff}, n]), ([\mathsf{ff}, n], [\mathsf{ff}, n]), ([\mathsf{ff}, n], [\mathsf{tt}, 0]) \mid n \in \mathbb{N}\}.
\end{aligned}
$$

   (Ein möglicher) Ablauf von Γ_{count}:

$$[\mathsf{ff}, 7] \longrightarrow [\mathsf{tt}, 0] \longrightarrow [\mathsf{tt}, 1] \longrightarrow [\mathsf{tt}, 2] \longrightarrow [\mathsf{tt}, 3] \longrightarrow [\mathsf{ff}, 3] \longrightarrow [\mathsf{ff}, 3] \longrightarrow \ldots$$

2. Ampel (vgl. Einleitung): $\Gamma_{amp} = (\emptyset, V, Z, T)$ (propositionales STS) mit:

$$
\begin{aligned}
&V = \{aus, istrot, istgelb, istgruen\},\\
&Z = \{\eta : V \to \{\mathsf{ff}, \mathsf{tt}\} \mid \eta(v) = \mathsf{tt} \text{ für genau ein } v \in V\},\\
&\qquad\qquad \text{(Notation für } \eta \in Z\text{: dasjenige } v \in V \text{ mit } \eta(v) = \mathsf{tt}\text{.)}\\
&T = \{(istgruen, istgelb), (istgruen, aus),\\
&\qquad (istgelb, istrot), (istrot, istgruen), (aus, istgelb)\}.
\end{aligned}
$$

   Ablauf von Γ_{amp}:

$$
\begin{aligned}
istgruen \longrightarrow istgelb \longrightarrow istrot \longrightarrow\\
istgruen \longrightarrow aus \longrightarrow istgelb \longrightarrow istrot \longrightarrow \ldots
\end{aligned}
$$

Spezifikationen

Schreibweise: $\mathcal{L}_{\mathrm{T}}$ für jegliche Sprache $\mathcal{L}_{\mathrm{FOLTL}}$ mit eventuellen Erweiterungen.

Definition. Sei $\Gamma = (X, V, Z, T)$ ein STS über SIG und S. Eine Sprache $\mathcal{L}_{\mathrm{T}}(TSIG_{\Gamma})$ mit $TSIG_{\Gamma} = (SIG, X, V)$ heißt ***Sprache der temporalen Logik von*** Γ (Kurzbezeichnung: $\mathcal{L}_{\mathrm{T}\Gamma}$). Eine Formel von $ker(\mathcal{L}_{\mathrm{T}\Gamma})$ heißt ***Zustandsformel*** von Γ.

(Beachte: Für jeden Ablauf W von Γ ist $\mathsf{K} = (\mathsf{S}, \mathsf{W})$ eine temporale Struktur für $TSIG_{\Gamma}$.)

Definition. Sei Γ ein STS über SIG und S. Eine Theorie $Th = (\mathcal{L}_{\mathrm{T}\Gamma}, \mathcal{A}_{\Gamma})$ mit der Eigenschaft, dass jede temporale Struktur

$$\mathsf{K} = (\mathsf{S}, \mathsf{W}), \quad \mathsf{W} \text{ Ablauf von } \Gamma,$$

Modell von Th ist, heißt ***(temporallogische) Spezifikation*** von Γ.

Eine Formel A von $\mathcal{L}_{\mathrm{T}\Gamma}$, die in jeder derartigen temporalen Struktur gültig ist, heißt ***Γ-gültig*** (geschrieben: $\models_{\Gamma} A$).

Bemerkungen:

- Diese Definitionen (und die nachfolgenden Untersuchungen) übertragen sich analog auf propositionale STS (mit Sprachen $\mathcal{L}_{\mathrm{T}\Gamma}$ der temporalen Aussagenlogik.)
- Die Aufgabe, für in gegebenes STS Γ eine temporallogische Spezifikation zu finden, besteht im Wesentlichen darin, eine passende Axiomenmenge $\mathcal{A}_\Gamma$ zu finden. Die Formeln von $\mathcal{A}_\Gamma$ müssen Γ-gültig sein.

  $\mathcal{A}_\Gamma$ enthält:

  1. Axiome für die Struktur S_Γ.
     Triviale Lösung: $\mathcal{A}_\Gamma$ enthält alle in S_Γ gültigen Formeln (aus $ker(\mathcal{L}_{\mathrm{T}\Gamma})$). M.a.W.: Jede FOLTL-Spezifikation enthält die Axiome

     (data_Γ) Alle Zustandsformeln A mit $\models_{\mathsf{S}_\Gamma} A$.
  2. „Möglichst vollständige“ Axiome für W_Γ. Diese ***temporalen Axiome*** sind bestimmt durch die Zustände und die Transitionsrelation von Γ.

Beispiele

1. Axiome einer Spezifikation von Γ_{count} (aus obigem Beispiel):
   - (data_Γ),
   - $ein \rightarrow (ein' \wedge c' = c + 1) \vee (\neg ein' \wedge c' = c)$,
   - $\neg ein \rightarrow (\neg ein' \wedge c' = c) \vee (ein' \wedge c' = 0)$.
2. Axiome einer Spezifikation von Γ_{amp} (aus obigem Beispiel):
   - $aus \vee istrot \vee istgelb \vee istgruen$,
   - $v_1 \rightarrow \neg v_2$ für alle $v_1, v_2 \in \{aus, istrot, istgelb, istgruen\}$, $v_1 \neq v_2$,
   - $istgruen \rightarrow \bigcirc (istgelb \vee aus)$,
   - $istgelb \rightarrow \bigcirc istrot$,
   - $istrot \rightarrow \bigcirc istgruen$,
   - $aus \rightarrow \bigcirc istgelb$.

   (Beachte: Die letzten 4 Axiome dieser Spezifikation sind die Axiome der Theorie Th_{amp} aus Abschnitt 2.3.)

5.2 Spezielle Klassen von Transitionssystemen

Verwurzelte Transitionssysteme

Definition. Ein ***verwurzeltes Transitionssystem*** (kurz: rSTS) $\Gamma = (X, V, Z, T, start)$ (über SIG und S) ist ein STS $\Gamma'(SIG, \mathsf{S}) = (X, V, Z, T)$ zusammen mit einer geschlossenen Formel $start$ von $ker(\mathcal{L}_{T\Gamma'})$ (***Anfangsbedingung***). Ein ***Ablauf*** von Γ ist ein Ablauf $(\eta_0, \eta_1, \eta_2, \ldots)$ von Γ' mit $\mathsf{S}^{(\eta_0)}(start) = \mathsf{tt}$.

(Beachte: Schreibweise $\mathsf{S}^{(\eta_i)}(A)$ statt $\mathsf{S}^{(\xi,\eta_i)}(A)$, da dieser Wert nicht von ξ abhängt.)

Beispiele:

1. Zähler aus Abschnitt 5.1: Zusätzliche Anfangsbedingung

   $$start \equiv \neg ein \wedge c = 0.$$

   Im damit definierten rSTS: Jeder Ablauf beginnt mit Anfangszustand $[\mathsf{ff}, 0]$.

2. rSTS $\Gamma_{pot2}(SIG_{NAT}, \mathsf{N}) = (X, \emptyset, Z, T, start)$ mit (vgl. Beispiel in Abschnitt 4.4):

   $$\begin{aligned}
   &X = X_{NAT} = \{a\},\\
   &Z = \{\eta : X \to \mathbb{N}\},\\
   &T = \{(\eta, \eta') \in Z \times Z \mid \eta'(a) = 2 * \eta(a)\},\\
   &start \equiv a = 1.
   \end{aligned}$$

   (Einziger) Ablauf von Γ_{pot2}: $(\eta_0, \eta_1, \eta_2, \ldots)$ mit $\eta_i(a) = 2^i$, $i = 0, 1, 2, \ldots$

3. Türme von Hanoi:
   Zugrundegelegt sei:

   $$\begin{aligned}
   SIG = (&\{STONE, PILE\},\\
   &\{TOWER, EMPTY, PUSH, POP, TOP\},\\
   &\{<, DECR\}),\\
   \mathsf{S} \text{ mit: } &|\mathsf{S}|_{STONE} = \{1, \ldots, n\},\\
   &|\mathsf{S}|_{PILE} = \{1, \ldots, n\}^*,\\
   &TOWER^{\mathsf{S}} = (n, n-1, \ldots, 2, 1),\\
   &EMPTY^{\mathsf{S}} = \varepsilon,\\
   &PUSH^{\mathsf{S}} = push,\\
   &POP^{\mathsf{S}} = pop,\\
   &TOP^{\mathsf{S}} = top,\\
   &<^{\mathsf{S}}(i, j) = \mathsf{tt} \Leftrightarrow i < j,\\
   &DECR^{\mathsf{S}}(i_1, \ldots, i_m) = \mathsf{tt} \Leftrightarrow i_m < i_{m-1} < \ldots < i_1.
   \end{aligned}$$

   ($push, pop, top$: „wie üblich“.)

   rSTS $\Gamma_{TvH}(SIG, \mathsf{S}) = (X, \emptyset, Z, T, start)$ mit:

   $$\begin{aligned}
   &X = X_{PILE} = \{p_1, p_2, p_3\},\\
   &Z = \{\eta \mid \eta : X_{PILE} \to \{1, \ldots, n\}^*\},\\
   &T' = \{(\eta, \eta') \in Z \times Z \mid\\
   &\qquad \eta(p_i) \neq \varepsilon, top(\eta(p_i)) < top(\eta(p_j)), \text{ falls } \eta(p_j) \neq \varepsilon,\\
   &\qquad \eta'(p_i) = pop(\eta(p_i)),\\
   &\qquad \eta'(p_j) = push(\eta(p_j), top(\eta(p_i))),\\
   &\qquad \eta'(p_k) = \eta(p_k),\\
   &\qquad\qquad i, j, k \in \{1, 2, 3\} \qquad \text{(paarweise verschieden)}\}
   \end{aligned}$$

$$T = T' \cup \{(\eta, \eta) \in Z \times Z \mid \text{es gibt kein } \eta' \in Z \text{ mit } (\eta, \eta') \in T'\}.$$

Notation: $T = tot(T')$ (***totaler Abschluss*** von T').

$$start \equiv p_1 = TOWER \wedge p_2 = EMPTY \wedge p_3 = EMPTY.$$

(Ein möglicher) Ablauf von Γ_{TvH} (mit $n = 3$):

$$[321, \varepsilon, \varepsilon] \longrightarrow [32, 1, \varepsilon] \longrightarrow [3, 1, 2] \longrightarrow [3, \varepsilon, 21] \longrightarrow [\varepsilon, 3, 21] \longrightarrow \ldots$$

(Bezeichnung (z.B.): $[3, \varepsilon, 21]$ bezeichnet η mit $\eta(p_1) = (3)$, $\eta(p_2) = \varepsilon$, $\eta(p_3) = (2, 1)$.)

Axiome für rSTS

Satz 5.2.1 Für jedes rSTS Γ ist die Formel **init** $\rightarrow start_\Gamma$ Γ-gültig.

Folgerung: Das Axiom

(root$_\Gamma$) **init** $\rightarrow start_\Gamma$

kann für jedes rSTS zu den temporalen Axiomen einer Spezifikation von Γ hinzugenommen werden.

Beispiele:

1. Zähler als rSTS (s.o.): Spezifikation enthält zusätzlich zu den Axiomen von Γ_{count} (Abschnitt 5.1) das Axiom

   init $\rightarrow \neg ein \wedge c = 0$.

2. Temporale Axiome zur Spezifikation von Γ_{pot2} (s.o.;vgl. Beispiel in Abschnitt 4.4):

   init $\rightarrow a = 1$,
   $\Box(a' = 2 * a)$.

Markierte Transitionssysteme

Idee: Ein Ablauf

$$\eta_0 \longrightarrow \eta_1 \longrightarrow \eta_2 \longrightarrow \ldots$$

eines (r)STS „protokolliert“ die Zustände, die ein System durchläuft.
In vielen Anwendungen: Übergänge $\eta_i \rightsquigarrow \eta_{i+1}$ werden verursacht durch „Aktionen“, deren „Protokollierung“ ebenfalls wünschenswert ist, als Bild:

$$\eta_0 \xrightarrow{\lambda_0} \eta_1 \xrightarrow{\lambda_1} \eta_2 \xrightarrow{\lambda_2} \ldots \qquad (\lambda_i \text{ „Aktion, die von } \eta_i \text{ zu } \eta_{i+1} \text{ führt“}).$$

Beispiel Türme von Hanoi (s.o.): Die „Aktionen“ λ_{ij} in einem Ablauf sind Verlegungen eines Steins von p_i nach p_j ($i \in \{1, 2, 3\}$), z.B.:

$$[321, \varepsilon, \varepsilon] \xrightarrow{\lambda_{12}} [32, 1, \varepsilon] \xrightarrow{\lambda_{13}} [3, 1, 2] \xrightarrow{\lambda_{23}} [3, \varepsilon, 21] \xrightarrow{\lambda_{12}} \ldots$$

Außerdem: Die „Aktionen“ sind oft mit Bedingungen an ihre Ausführbarkeit in einem Zustand verknüpft.

Im Beispiel: Ein Stein kann nur dann von p_i nach p_j verlegt werden, wenn gilt:

- Auf p_i liegt mindestens ein Stein,
- der oberste Stein von p_i ist kleiner als der oberste Stein von p_j (falls vorhanden).

Definition. Ein ***markiertes Transitionssystem*** (kurz: lSTS) $\Gamma = (X, V, Z, T, Act, \mathcal{E})$ (über SIG und S) ist gegeben durch

- ein STS $\Gamma'(SIG, \mathsf{S}) = (X, V, Z, T)$,
- eine endliche Menge Act von ***Aktionen***,
- eine Menge $\mathcal{E} = \{enabled_\lambda \mid \lambda \in Act\}$ geschlossener Formeln von $ker(\mathcal{L}_{T\Gamma'})$ (***Ausführbarkeitsbedingungen***)

mit den Eigenschaften:

- V enthält $\{exec\,\lambda \mid \lambda \in Act\}$ als Teilmenge,
- für alle $\eta \in Z$ ist $\mathsf{S}^{(\eta)}(exec\,\lambda \to enabled_\lambda) = \mathsf{tt}$ (kurz: η ist ***zulässig***),
- falls $(\eta, \eta') \in T$ und $\eta(exec\,\lambda) = \mathsf{ff}$ für alle $\lambda \in Act$, so ist $\eta' = \eta$.

Ein ***Ablauf*** von Γ ist ein Ablauf von Γ'.

Bemerkungen:

- Intuitive Bedeutung von $exec\,\lambda$: „λ wird (als nächstes) ausgeführt“.
- Intuitive Bedeutung der Zulässigkeit von Zuständen: „Nur ausführbare λ können ausgeführt werden“,
- Beachte Verallgemeinerung: In einem Zustand können mehrere Aktionen „gleichzeitig“ ausgeführt werden ($\eta(exec\,\lambda) = \mathsf{tt}$ für mehrere λ).
- Verwurzelte markierte Transitionssysteme (rlSTS) sind analog definiert.

Beispiel: Türme von Hanoi (s.o.) als rlSTS $\Gamma^l_{TvH}(SIG, \mathsf{S}) = (X, V, Z, T, start, Act, \mathcal{E})$:

$SIG, \mathsf{S}, X, start$: wie in Γ_{TvH},

$Act = \{\lambda_{12}, \lambda_{13}, \lambda_{21}, \lambda_{23}, \lambda_{31}, \lambda_{32}\}$,

$\mathcal{E} = \{enabled_\lambda \mid \lambda \in Act\}$ mit

$\quad enabled_{\lambda_{ij}} \equiv p_i \neq EMPTY \wedge (p_j \neq EMPTY \to TOP(p_i) < TOP(p_j))$

$V = \{exec\,\lambda \mid \lambda \in Act\}$,

$Z = \{\eta : X \cup V \to \{1, \ldots, n\}^* \cup \{\mathsf{ff}, \mathsf{tt}\} \mid \eta$ zulässig,
$\eta(p_i) \in \{1, \ldots, n\}^*$ für $p_i \in X$,
$\eta(exec\,\lambda_{ij}) \in \{\mathsf{ff}, \mathsf{tt}\}$ für $\lambda_{ij} \in Act$,
$\eta(exec\,\lambda) = \mathsf{tt}$ für höchstens ein $\lambda \in Act\}$,

$T' = \{(\eta, \eta') \in Z \times Z \mid \eta(exec\,\lambda_{ij}) = \mathsf{tt}$ für ein $\lambda_{ij} \in Act$ und
$\eta'(p_i) = pop(\eta(p_i)), \eta'(p_j) = push(\eta(p_j), top(\eta(p_i)))$,
$\eta'(p_k) = \eta(p_k)$ für $k \neq i, k \neq j\}$,

$T = tot(T')$.

Ablauf (vgl. Beispiel von oben):

$$[321, \varepsilon, \varepsilon, \lambda_{12}] \longrightarrow [32, 1, \varepsilon, \lambda_{13}] \longrightarrow [3, 1, 2, \lambda_{23}] \longrightarrow [3, \varepsilon, 21, \lambda_{12}] \longrightarrow \ldots$$

(Bezeichnung: $[\ldots, \lambda_{ij}]$ bezeichnet η mit $\eta(exec\,\lambda_{ij}) = \mathsf{tt}$.)

Axiome für lSTS

Abkürzung (für lSTS Γ mit $Act = \{\lambda_1, \ldots, \lambda_n\}$): nil_Γ für $\neg exec\,\lambda_1 \wedge \ldots \wedge \neg exec\,\lambda_n$.

Satz 5.2.2 Für jedes lSTS Γ sind die Formeln $nil_\Gamma \wedge A \rightarrow \circ A$ (wobei A Zustandsformel von Γ ist) und $exec\,\lambda \rightarrow enabled_\lambda$ (für beliebige $\lambda \in Act$) Γ-gültig.

Folgerung: Die Axiome

(nil_Γ) $nil_\Gamma \wedge A \rightarrow \circ A$, falls A Zustandsformel von Γ,

(action_Γ) $exec\,\lambda \rightarrow enabled_\lambda$ für alle $\lambda \in Act_\Gamma$

können für jedes (r)lSTS Γ zu den temporalen Axiomen einer Spezifikation von Γ hinzugenommen werden.

Beispiel: Temporale Axiome einer Spezifikation der Türme von Hanoi (als rlSTS, s.o.):

$$\begin{aligned}
&\textbf{init} \rightarrow p_1 = TOWER \wedge p_2 = EMPTY \wedge p_3 = EMPTY,\\
&nil_{\Gamma^l_{TvH}} \wedge A \rightarrow \circ A \quad \text{falls } A \text{ Zustandsformel von } \Gamma^l_{TvH},\\
&exec\,\lambda_{ij} \rightarrow p_i \neq EMPTY \wedge (p_j \neq EMPTY \rightarrow TOP(p_i) < TOP(p_j))\\
&\qquad\qquad \text{für alle } \lambda_{ij} \in Act_{\Gamma^l_{TvH}},\\
&exec\,\lambda_{ij} \rightarrow \neg exec\,\lambda_{kl} \quad \text{für alle } \lambda_{ij}, \lambda_{kl} \in Act_{\Gamma^l_{TvH}}, \lambda_{ij} \neq \lambda_{kl},\\
&exec\,\lambda_{ij} \rightarrow p_i' = POP(p_i) \wedge p_j' = PUSH(p_j, TOP(p_i)) \wedge p_k' = p_k\\
&\qquad\qquad \text{für alle } \lambda_{ij} \in Act_{\Gamma^l_{TvH}}, k \neq i, k \neq j.
\end{aligned}$$

Eine abgeleitete Regel

In jedem (r)lSTS Γ kann folgende abgeleitete Regel verwendet werden:

(trans_Γ) $exec\,\lambda \wedge A \rightarrow \circ B$ für alle $\lambda \in Act_\Gamma$,
$nil_\Gamma \wedge A \rightarrow B$
$\vdash A \rightarrow \circ B$ falls A und B Zustandsformeln von Γ.

Herleitung von (trans_Γ): (Es sei $Act_\Gamma = \{\lambda_1, \ldots, \lambda_n\}$.)

(1)	$exec\,\lambda \wedge A \rightarrow \circ B$ für alle $\lambda \in Act_\Gamma$	Annahme
(2)	$nil_\Gamma \wedge A \rightarrow B$	Annahme
(3)	$nil_\Gamma \vee exec\,\lambda_1 \vee \ldots \vee exec\,\lambda_n$	(taut)
(4)	$nil_\Gamma \wedge A \rightarrow nil_\Gamma \wedge B$	(prop),(2)
(5)	$nil_\Gamma \wedge B \rightarrow \circ B$	(nil_Γ)
(6)	$nil_\Gamma \wedge A \rightarrow \circ B$	(prop),(4),(5)
(7)	$A \rightarrow \circ B$	(prop),(1),(3),(6)

Faire Transitionssysteme

Idee: Transitionssysteme sind im Allgemeinen nichtdeterministisch.

In vielen Anwendungen: Gewisse Einschränkungen der freien Auswahl des nächsten Schrittes wünschenswert.

Fairness: Eine spezielle Art von derartigen Einschränkungen.

Beispiel: Das propositionale lSTS $(\emptyset, V, Z, T, Act, \mathcal{E})$ mit

$$
\begin{aligned}
&Act = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma\},\\
&V = \{exec\,\lambda \mid \lambda \in Act\} \cup \{r_1, r_2\},\\
&enabled_{\alpha_1} \equiv \neg r_1,\ enabled_{\alpha_2} \equiv \neg r_2,\ enabled_{\beta_1} \equiv r_1,\ enabled_{\beta_2} \equiv r_2,\\
&enabled_{\gamma} \equiv \textbf{true},\\
&Z = \{\eta : V \to \{\mathsf{ff}, \mathsf{tt}\} \mid \eta \text{ zulässig und } \eta(exec\,\lambda) = \mathsf{tt} \text{ für genau ein } \lambda \in Act\},\\
&T = \{(\eta, \eta') \in Z \times Z \mid \text{falls } \eta(exec\,\alpha_1) = \mathsf{tt}, \text{so } \eta'(r_1) = \mathsf{tt}, \eta'(r_2) = \eta(r_2),\\
&\qquad\qquad \text{falls } \eta(exec\,\alpha_2) = \mathsf{tt}, \text{so } \eta'(r_1) = \eta(r_1), \eta'(r_2) = \mathsf{tt},\\
&\qquad\qquad \text{falls } \eta(exec\,\beta_1) = \mathsf{tt}, \text{so } \eta'(r_1) = \mathsf{ff}, \eta'(r_2) = \eta(r_2),\\
&\qquad\qquad \text{falls } \eta(exec\,\beta_2) = \mathsf{tt}, \text{so } \eta'(r_1) = \eta(r_1), \eta'(r_2) = \mathsf{ff},\\
&\qquad\qquad \text{falls } \eta(exec\,\gamma) = \mathsf{tt}, \text{so } \eta'(r_1) = \eta(r_1), \eta'(r_2) = \eta(r_2)\}.
\end{aligned}
$$

formalisiert einen Drucker, der fortwährend Druckaufträge von zwei Prozessen P_1 und P_2 abarbeitet. ($\alpha_i \mathrel{\hat{=}}$ „Drucker wird von P_i angefordert“, $\beta_i \mathrel{\hat{=}}$ „Drucker druckt für P_i“, $\gamma \mathrel{\hat{=}}$ „Drucker druckt nicht“, $r_i \mathrel{\hat{=}}$ „Druckauftrag von P_i ist vorhanden“.)

Mögliche Abläufe (η bezeichnet durch $\{r_i \mid \eta(r_i) = \mathsf{tt}\}$, Aktionen als Pfeilmarkierungen):

$$\emptyset \xrightarrow{\gamma} \emptyset \xrightarrow{\alpha_1} \{r_1\} \xrightarrow{\alpha_2} \{r_1, r_2\} \xrightarrow{\beta_1} \{r_2\} \xrightarrow{\alpha_1} \{r_1, r_2\} \xrightarrow{\beta_2} \{r_1\} \xrightarrow{\beta_1} \dots$$

$$\emptyset \xrightarrow{\alpha_1} \{r_1\} \xrightarrow{\alpha_2} \{r_1, r_2\} \xrightarrow{\beta_1} \{r_2\} \xrightarrow{\alpha_1} \{r_1, r_2\} \xrightarrow{\beta_1} \{r_2\} \xrightarrow{\alpha_1} \{r_1, r_2\} \xrightarrow{\beta_1} \dots$$

Der zweite Ablauf (Auftrag von P_2 ist vorhanden, wird aber nie erledigt) ist „unfair“ und sollte (ebenso wie andere von analoger Art) ausgeschlossen werden.

Definition. Sei Γ ein lSTS. Ein Ablauf $\mathsf{W}_\Gamma = (\eta_0, \eta_1, \eta_2, \dots)$ von Γ heißt ***fair***, wenn für alle $\lambda \in Act_\Gamma$ gilt: Falls $\mathsf{S}_\Gamma^{(\eta_j)}(enabled_\lambda) = \mathsf{tt}$ für unendlich viele j, so ist $\eta_k(exec\,\lambda) = \mathsf{tt}$ für unendlich viele k. Ein ***faires Transitionssystem*** (kurz: fSTS) ist ein lSTS Γ mit der Eigenschaft, dass die Abläufe auf die fairen Abläufe von Γ beschränkt werden.

Bemerkungen:

1. Intuitive Bedeutung der Einschränkung: „Unendlich oft ausführbare Aktionen werden auch unendlich oft ausgeführt“.
2. Verwurzelte faire Transitionssysteme (rfSTS) sind analog definiert.

Axiome für fSTS

Satz 5.2.3 Für jedes fSTS Γ ist die Formel $\Box\Diamond enabled_\lambda \to \Diamond exec\,\lambda$ (für alle $\lambda \in Act_\Gamma$) Γ-gültig.

Folgerung: Das Axiom

(fair$_\Gamma$) $\quad \Box\Diamond enabled_\lambda \rightarrow \Diamond exec\,\lambda \quad$ für alle $\lambda \in Act_\Gamma$

kann für jedes fSTS zu den temporalen Axiomen einer Spezifikation von Γ hinzugenommen werden.

Beispiel: Temporale Axiome einer Spezifikation des Druckersystems (als fSTS, s.o.):

$nil_\Gamma \wedge A \rightarrow \circ A$, falls A Zustandsformel des Systems,
$exec\,\alpha_1 \rightarrow \neg r_1$,
$exec\,\alpha_2 \rightarrow \neg r_2$,
$exec\,\beta_1 \rightarrow r_1$,
$exec\,\beta_2 \rightarrow r_2$,
$\Box\Diamond enabled_\lambda \rightarrow \Diamond exec\,\lambda$ für $\lambda \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma\}$,
$exec\,\alpha_1 \vee exec\,\alpha_2 \vee exec\,\beta_1 \vee exec\,\beta_2 \vee exec\,\gamma$,
$exec\,\lambda \rightarrow \neg exec\,\overline{\lambda}$ für $\lambda, \overline{\lambda} \in \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma\}$, $\lambda \neq \overline{\lambda}$,
$exec\,\alpha_1 \rightarrow r_1' \wedge (r_2' \leftrightarrow r_2)$,
$exec\,\alpha_2 \rightarrow (r_1' \leftrightarrow r_1) \wedge r_2'$,
$exec\,\beta_1 \rightarrow \neg r_1' \wedge (r_2' \leftrightarrow r_2)$,
$exec\,\beta_2 \rightarrow (r_1' \leftrightarrow r_1) \wedge \neg r_2'$,
$exec\,\gamma \rightarrow (r_1' \leftrightarrow r_1) \wedge (r_2' \leftrightarrow r_2)$.

Eine abgeleitete Regel

In jedem (r)fSTS Γ kann folgende abgeleitete Regel verwendet werden:

(progress$_\Gamma$) $\quad enabled_\lambda \rightarrow \neg nil_\Gamma \quad$ für alle $\lambda \in Act_\Gamma$.

Herleitung von (progress$_\Gamma$):

(1)	$nil_\Gamma \wedge enabled_\lambda \rightarrow \circ(nil_\Gamma \wedge enabled_\lambda)$	(prop),(nil$_\Gamma$)
(2)	$nil_\Gamma \wedge enabled_\lambda \rightarrow \Box(nil_\Gamma \wedge enabled_\lambda)$	(ind1),(1)
(3)	$\Box(nil_\Gamma \wedge enabled_\lambda) \rightarrow \Box nil_\Gamma \wedge \Box enabled_\lambda$	(prop),(T17)
(4)	$\Box enabled_\lambda \rightarrow \Box\Diamond enabled_\lambda$	(T5),(T19)
(5)	$\Box enabled_\lambda \rightarrow \Diamond exec\,\lambda$	(prop),(fair$_\Gamma$),(4)
(6)	$nil_\Gamma \wedge enabled_\lambda \rightarrow \Box nil_\Gamma \wedge \Diamond enabled_\lambda$	(prop),(2),(3),(5)
(7)	$\Box nil_\Gamma \wedge \Diamond exec\,\lambda \rightarrow \Diamond(nil_\Gamma \wedge exec\,\lambda)$	(prop),(T29)
(8)	$\Box\neg(nil_\Gamma \wedge exec\,\lambda) \rightarrow (enabled_\lambda \rightarrow \neg nil_\Gamma)$	(prop),(T2),(6),(7)
(9)	$\neg(nil_\Gamma \wedge exec\,\lambda)$	(taut)
(10)	$\Box\neg(nil_\Gamma \wedge exec\,\lambda)$	(alw),(9)
(11)	$enabled_\lambda \rightarrow \neg nil_\Gamma$	(prop),(8),(10)

5.3 Verifikation

Eigenschaften von Zustandssystemen

- Spezifikation eines Zustandssystems (d.h. formal: eines (r)(l)(f)STS) Γ: $(\mathcal{L}_{\mathrm{T}\Gamma}, \mathcal{A}_\Gamma)$.
- Formel F von $\mathcal{L}_{\mathrm{T}\Gamma}$: beschreibt eine ***Eigenschaft*** von Γ.
- $\models_\Gamma F$ bedeutet: „F gilt in jedem Ablauf von Γ“,
  „Γ besitzt die durch F beschriebene Eigenschaft“
- (Deduktive) ***Verifikation*** (von F): Formaler Nachweis von $\models_\Gamma F$ durch Herleitung von $\mathcal{A}_\Gamma \vdash F$.

Bemerkung:

Falls Eigenschaften eines Systems verifiziert werden sollen („Verifikation von Zustandssystemen“), so hängt die Auswahl einer geeigneten Sprache $\mathcal{L}_\mathrm{T}$ nicht nur von der Art des STS (r,l,f) ab, sondern auch davon, welche Eigenschaften dies sind.

Beispiele:

1. Türme von Hanoi (Γ_{TvH} oder Γ^l_{TvH}, Abschnitt 5.2)
   Eigenschaft: „Zu keiner Zeit liegt auf einem Platz ein kleinerer unter einem größeren Stein.“
   Als Formel:
   $$start \rightarrow \Box(DECR(p_1) \wedge DECR(p_2) \wedge DECR(p_3)).$$
2. Zähler (Γ_{count}, Abschnitt 5.1)
   Eigenschaft: „Wenn der Zähler ausgeschaltet ist, so beginnt die Zählung beim nächsten Einschalten bei 0.“
   Als Formel:
   $$\neg ein \rightarrow c = 0 \ \textbf{atnext} \ ein.$$
3. Türme von Hanoi (Γ_{TvH} oder Γ^l_{TvH}, Abschnitt 5.2)
   Eigenschaft: „Ausgehend von der Startsituation steht der Turm irgendwann auf Platz 2.“
   Als Formel:
   $$start \rightarrow \Diamond(p_1 = EMPTY \wedge p_2 = TOWER \wedge p_3 = EMPTY).$$

(Beachte: In 1. und 2.: System besitzt die Eigenschaft.
In 3.: System besitzt die Eigenschaft nicht.)

Klassifikation von Eigenschaften

Systemeigenschaften können (zur Systematisierung) nach verschiedenen Kriterien in Klassen eingeteilt werden.

Einfache Klassifizierung gemäß der Gestalt der beschreibenden temporallogischen Formel (nicht erschöpfend, aber für viele Anwendungen ausreichend, vgl. obige Beispiele):

Invarianzeigenschaften:	$A \to \Box B$
Präzedenzeigenschaften:	$A \to B$ **atnext** C
	$A \to B$ **unless** C
	$\vdots$
Eintrittseigenschaften:	$A \to \Diamond B$

Dabei jeweils: A, B, C Zustandsformeln des jeweiligen Transitionssystems.

Bemerkungen:

- Eine Invarianzeigenschaft der Form $start \to \Box B$ (für ein verwurzeltes STS) kann auch durch $\Box B$ oder auch einfach durch B beschrieben werden.
- Für die einzelnen Klassen von Eigenschaften gibt es grundlegende Verifikationstechniken, ausgedrückt durch charakteristische Herleitungsregeln (im Folgenden nur für (r)lSTS bzw. (r)fSTS behandelt).

Regeln für Invarianzeigenschaften

Schreibweise: A **invof** Act_Γ für $(exec\,\lambda_1 \wedge A \to \circ A) \wedge \ldots \wedge (exec\,\lambda_n \wedge A \to \circ A)$ (wobei $Act_\Gamma = \{\lambda_1, \ldots, \lambda_n\}$).

Abgeleitete Regeln (***Invarianzregeln***), die bei der Verifikation jedes (r)lSTS Γ verwendet werden können:

(inv_Γ) $A \to B, A$ **invof** $Act_\Gamma \vdash A \to \Box B$, falls A Zustandsformel von Γ ist,

(inv1_Γ) A **invof** $Act_\Gamma \vdash A \to \Box A$, falls A Zustandsformel von Γ ist,

(invstart_Γ) $start_\Gamma \to A, A$ **invof** $Act_\Gamma \vdash \Box A$, falls A Zustandsformel von Γ ist.

Bemerkung:

In Anwendungen wird eine Formel $A \to \Box B$ meist in der Form

$$A \to A', A' \textbf{ invof } Act_\Gamma, A' \to B \vdash A \to \Box B$$

hergeleitet.

Herleitung von (inv_Γ):

(1)	$A \to B$	Annahme
(2)	A **invof** Act_Γ	Annahme
(3)	$exec\,\lambda \wedge A \to \circ A$ für alle $\lambda \in Act_\Gamma$	(prop),(2)
(4)	$nil_\Gamma \wedge A \to A$	(taut)
(5)	$A \to \circ A$	(trans_Γ),(3),(4)
(6)	$A \to \Box B$	(ind),(1),(5)

Herleitung von (inv1$_\Gamma$): Analog mit (ind1).

Herleitung von (invstart$_\Gamma$):

(1)	$start_\Gamma \rightarrow A$	Annahme
(2)	A **invof** Act_Γ	Annahme
(3)	$A \rightarrow \circ A$	wie oben
(4)	**init** $\rightarrow start_\Gamma$	(root$_\Gamma$)
(5)	**init** $\rightarrow A$	(prop),(1),(4)
(6)	A	(indinit),(3),(5)
(7)	$\Box A$	(alw),(6)

Regeln für Präzedenzeigenschaften

Abgeleitete Regeln, die bei der Verifikation jedes (r)lSTS Γ verwendet werden können:

(invatnext$_\Gamma$) $exec\lambda \wedge A \rightarrow \circ(C \rightarrow B) \wedge \circ(\neg C \rightarrow A)$ für alle $\lambda \in Act_\Gamma$,
$nil_\Gamma \wedge A \rightarrow (C \rightarrow B)$
$\vdash A \rightarrow B$ **atnext** C, falls A, B und C Zustandsformeln von Γ sind.

(invunless$_\Gamma$) $exec\lambda \wedge A \rightarrow \circ C \vee \circ(A \wedge B)$ für alle $\lambda \in Act_\Gamma$,
$nil_\Gamma \wedge A \rightarrow B \vee C$
$\vdash A \rightarrow B$ **unless** C, falls A, B und C Zustandsformeln von Γ sind.

(invbefore$_\Gamma$) $exec\lambda \wedge A \rightarrow \circ\neg C \wedge \circ(A \vee B)$ für alle $\lambda \in Act_\Gamma$,
$nil_\Gamma \wedge A \rightarrow \neg C$
$\vdash A \rightarrow B$ **before** C, falls A, B und C Zustandsformeln von Γ sind.

Herleitung von (invatnext$_\Gamma$):

(1)	$exec\lambda \wedge A \rightarrow \circ(C \rightarrow B) \wedge \circ(\neg C \rightarrow A)$	für alle $\lambda \in Act_\Gamma$	Annahme
(2)	$nil_\Gamma \wedge A \rightarrow (C \rightarrow B)$		Annahme
(3)	$nil_\Gamma \wedge A \rightarrow (C \rightarrow B) \wedge (\neg C \rightarrow A)$		(prop),(2)
(4)	$A \rightarrow \circ(C \rightarrow B) \wedge \circ(\neg C \rightarrow A)$		(trans$_\Gamma$),(T15),(1),(3)
(5)	$A \rightarrow B$ **atnext** C		(indatnext),(4)

Herleitung von (invunless$_\Gamma$) und (invbefore$_\Gamma$): Analog mit (indunless) und (indbefore).

Regeln für Eintrittseigenschaften

Für Eintrittseigenschaften gibt es eine Reihe einfacher, jedoch nicht immer anwendbarer Regeln. Eine „universelle“ Beweismethode erfordert die Zugrundelegung einer temporalen Prädikatenlogik mit der Erweiterung „w“ und basiert auf der Regel

(wfr) $A \rightarrow \Diamond(B \vee \exists\bar{z}(\bar{z} \prec z \wedge A_z(\bar{z}))) \vdash \exists z A \rightarrow \Diamond B$, falls z nicht in B vorkommt

(Abschnitt 4.4). Die folgende Ausprägung der Methode setzt außerdem die Fairness des zu verifizierenden STS voraus.

Abgeleitete Regel, die bei der Verifikation jedes (r)fSTS Γ verwendet werden kann:

(fairwfr$_\Gamma$) $exec\,\lambda \wedge H_\lambda \wedge A \rightarrow \bigcirc(B \vee \exists \bar{z}(\bar{z} \prec z \wedge A_z(\bar{z})))$ für alle $\lambda \in Act_\Gamma$,
$exec\,\lambda \wedge \neg H_\lambda \wedge A \rightarrow \bigcirc(B \vee \exists \bar{z}(\bar{z} \preceq z \wedge A_z(\bar{z})))$ für alle $\lambda \in Act_\Gamma$,
$\Box A \rightarrow \Diamond(B \vee E_\Gamma)$
$\vdash \exists z A \rightarrow \Diamond B$, falls A und B Zustandsformeln von Γ sind.

Dabei: $Act_\Gamma = \{\lambda_1, \ldots, \lambda_m\}$,
$H_{\lambda_1}, \ldots, H_{\lambda_m}$ Formeln von $\mathcal{L}_{T\Gamma}$ ohne flexible Symbole,
$E_\Gamma \equiv (H_{\lambda_1} \wedge enabled_{\lambda_1}) \vee \ldots \vee (H_{\lambda_m} \wedge enabled_{\lambda_m})$,
$z, \bar{z}$ Variablen der Sorte WF, $\prec^{\mathsf{S}_\Gamma}$ fundierte Relation auf $|\mathsf{S}_\Gamma|_{WF}$,
$\bar{z} \preceq z$ Abkürzung für $\bar{z} \prec z \vee \bar{z} = z$.

Herleitung von (fairwfr$_\Gamma$):

Sei $C_1 \equiv \exists \bar{z}(\bar{z} \prec z \wedge A_z(\bar{z}))$, $C_2 \equiv \exists \bar{z}(\bar{z} \preceq z \wedge A_z(\bar{z}))$, $D \equiv \bigvee_{\lambda \in Act_\Gamma}(H_\lambda \wedge exec\,\lambda)$.

(1)	$exec\,\lambda \wedge H_\lambda \wedge A \rightarrow \bigcirc(B \vee C_1)$ für alle $\lambda \in Act_\Gamma$	Annahme
(2)	$exec\,\lambda \wedge \neg H_\lambda \wedge A \rightarrow \bigcirc(B \vee C_2)$ für alle $\lambda \in Act_\Gamma$	Annahme
(3)	$\Box A \rightarrow \Diamond(B \vee E_\Gamma)$	Annahme
(4)	$C_2 \wedge \Box\neg B \rightarrow \Box(C_2 \wedge \Box\neg B)$	(inv1$_\Gamma$),(1),(2),(ltl3),(T15)
(5)	$A \wedge \Box\neg B \wedge \Box\neg C_1 \rightarrow \Box A \wedge \Box\Diamond E_\Gamma$	(pred),(3),(4),(ltl3), (T17),(T30),(T32)
(6)	$\Box\Diamond(H_\lambda \wedge enabled_\lambda) \rightarrow \Box\Diamond H_\lambda \wedge \Box\Diamond enabled_\lambda$ für alle $\lambda \in Act_\Gamma$	(T17),(T19),(T22)
(7)	$\Box\Diamond H_\lambda \rightarrow \Box H_\lambda$ für alle $\lambda \in Act_\Gamma$	(ind1),(ind2),(ltl3),(ltl6)
(8)	$\Box\Diamond(H_\lambda \wedge enabled_\lambda) \rightarrow \Box H_\lambda \wedge \Diamond exec\,\lambda$ für alle $\lambda \in Act_\Gamma$	(fair$_\Gamma$),(6),(7)
(9)	$A \wedge \Box\neg B \wedge \Box\neg C_1 \rightarrow \Diamond(D \wedge A \wedge \Box\neg B)$	(5),(8),(T10),(T17), (T19),(T29),(T32)
(10)	$exec\,\lambda \wedge H_\lambda \wedge A \wedge \Box\neg B \rightarrow \Diamond C_1$ für alle $\lambda \in Act_\Gamma$	(som),(T27),(1)
(11)	$A \wedge \Box\neg B \wedge \Box\neg C_1 \rightarrow \Diamond C_1$	(chain),(9),(10)
(12)	$A \rightarrow \Diamond(B \vee C_1)$	(prop),(27)
(13)	$\exists z A \rightarrow \Diamond B$	(wfr),(12)

Zusammenfassung: Sprachen für Spezifikation und Verifikation

adäquate Sprache: FOLTL (LTL) + ...	Spezifikation	Verifikation von Invarianz-Eigenschaften	Präzedenz-Eigenschaften	Eintritts-Eigenschaften
(f)(l)STS	–	–	„b“	(„f“)
(f)(l)rSTS	„i“	„i“	„i“+„b“	„i“ (+„f“)

(Beachte: Für die Formulierung und Verifikation komplexerer Eigenschaften sind eventuell weitere Sprachvarianten angemessen.)

5.4 Beispiele

Türme von Hanoi

Komplette Spezifikation als rlSTS Γ^l_{TvH}:

(data) Alle in $\mathsf{S}_{\Gamma^l_{TvH}}$ gültigen Zustandsformeln,
(root) **init** $\rightarrow p_1 = TOWER \wedge p_2 = EMPTY \wedge p_3 = EMPTY$,
(nil) $nil_{\Gamma^l_{TvH}} \wedge A \rightarrow \circ A$ falls A Zustandsformel von Γ^l_{TvH},
(action) $exec\,\lambda_{ij} \rightarrow p_i \neq EMPTY \wedge (p_j \neq EMPTY \rightarrow TOP(p_i) < TOP(p_j))$
für alle $\lambda_{ij} \in Act_{\Gamma^l_{TvH}}$,
(TvH1) $exec\,\lambda_{ij} \rightarrow \neg exec\,\lambda_{kl}$ für alle $\lambda_{ij}, \lambda_{kl} \in Act_{\Gamma^l_{TvH}}$, $\lambda_{ij} \neq \lambda_{kl}$,
(TvH2) $exec\,\lambda_{ij} \rightarrow p'_i = POP(p_i) \wedge p'_j = PUSH(p_j, TOP(p_i)) \wedge p'_k = p_k$
für alle $\lambda_{ij} \in Act_{\Gamma^l_{TvH}}, k \neq i, k \neq j$.

Behauptung: $start \rightarrow \Box B$,
wobei $start \equiv p_1 = TOWER \wedge p_2 = EMPTY \wedge p_3 = EMPTY$,
$B \equiv DECR(p_1) \wedge DECR(p_2) \wedge DECR(p_3)$.

Herleitung:

(1)	$start \rightarrow B$	(data)
(2)	$exec\,\lambda_{12} \wedge B \rightarrow \circ B$	(data),(action),(TvH2)
(3)	$exec\,\lambda_{13} \wedge B \rightarrow \circ B$	(data),(action),(TvH2)
⋮		
(7)	$exec\,\lambda_{32} \wedge B \rightarrow \circ B$	(data),(action),(TvH2)
(8)	B **invof** $Act_{\Gamma^l_{TvH}}$	(prop),(2)–(7)
(9)	$B \rightarrow \Box B$	(inv1),(8)
(10)	$start \rightarrow \Box B$	(prop),(1),(9)

„Gleichwertige“ Behauptung: $\Box B$.

Herleitung:

(1)	$start \rightarrow B$	(data)
(2)	B **invof** $Act_{\Gamma^l_{TvH}}$	wie oben
(3)	$\Box B$	(invstart),(1),(2)

Zähler

Zähler aus Abschnitt 5.1, jetzt als lSTS Γ'_c mit

$Act_{\Gamma'_c} = \{\lambda_{ein}, \lambda_{aus}, \lambda_z, \lambda_p\}$,
$enabled_{\lambda_{ein}} \equiv \neg ein$, $enabled_{\lambda_{aus}} \equiv ein$, $enabled_{\lambda_z} \equiv ein$, $enabled_{\lambda_p} \equiv \neg ein$.

Spezifikation von Γ'_c:

(data) Alle in $\mathsf{S}_{\Gamma'_c}$ gültigen Zustandsformeln,
(nil) $nil_{\Gamma'_c} \wedge A \rightarrow \circ A$ falls A Zustandsformel von Γ'_c,
(Z1) $exec\,\lambda_{ein} \vee exec\,\lambda_{aus} \vee exec\,\lambda_z \vee exec\,\lambda_p$,
(Z2) $exec\,\lambda_1 \rightarrow \neg exec\,\lambda_2$ für $\lambda_1, \lambda_2 \in Act_{\Gamma'_c}, \lambda_1 \neq \lambda_2$,
(Z3) $exec\,\lambda_{ein} \rightarrow \neg ein \wedge ein' \wedge c' = 0$,
(Z4) $exec\,\lambda_{aus} \rightarrow ein \wedge \neg ein' \wedge c' = c$,
(Z5) $exec\,\lambda_z \rightarrow ein \wedge ein' \wedge c' = c + 1$,
(Z6) $exec\,\lambda_p \rightarrow \neg ein \wedge \neg ein' \wedge c' = c$.

(In (Z3)–(Z6) enthalten: (action)-Axiome.)

Behauptung: $\neg ein \rightarrow c = 0$ **atnext** ein.

Herleitung:

(1)	$exec\,\lambda_{ein} \wedge \neg ein \rightarrow \circ(ein \rightarrow c = 0) \wedge \circ(\neg ein \rightarrow \neg ein)$	(Z3),(prop)
(2)	$exec\,\lambda_{aus} \wedge \neg ein \rightarrow \circ(ein \rightarrow c = 0) \wedge \circ(\neg ein \rightarrow \neg ein)$	(Z4),(prop)
(3)	$exec\,\lambda_z \wedge \neg ein \rightarrow \circ(ein \rightarrow c = 0) \wedge \circ(\neg ein \rightarrow \neg ein)$	(Z5),(prop)
(4)	$exec\,\lambda_p \wedge \neg ein \rightarrow \circ(ein \rightarrow c = 0) \wedge \circ(\neg ein \rightarrow \neg ein)$	(Z6),(prop)
(5)	$nil_{\Gamma'_c} \wedge \neg ein \rightarrow (ein \rightarrow c = 0)$	(taut)
(6)	$\neg ein \rightarrow c = 0$ **atnext** ein	(invatnext),(1)-(5)

Fairer Zähler ohne Zurücksetzung

Modifizierter Zähler: Beim Anschalten setzt der Zähler nicht auf 0 zurück, sondern zählt beim zuletzt erreichten Wert weiter.

Spezifikation als fSTS Γ''_c: wie Γ'_c, mit (Z3′) statt (Z3) und zusätzlichem Fairness-Axiom:

(fair) $\Box\Diamond enabled_\lambda \rightarrow \Diamond exec\,\lambda$ für alle $\lambda \in Act_{\Gamma''_c}$,
(Z3′) $exec\,\lambda_{ein} \rightarrow \neg ein \wedge ein' \wedge c' = c$.

($Act_{\Gamma''_c} = Act_{\Gamma'_c}$.)

Behauptung: $c = 0 \rightarrow \Diamond(c = x)$ (x Variable der Sorte NAT).

Zur Herleitung mit der Regel (fairwfr$_{\Gamma''_c}$) sei WF die Sorte NAT mit $<$ als $\prec$ sowie

$$H_{\lambda_z} \equiv \textbf{true}, \qquad H_\lambda \equiv \textbf{false} \quad \text{für } \lambda \in \{\lambda_{ein}, \lambda_{aus}, \lambda_p\}.$$

(Dann ist $E_{\Gamma''_c} \equiv (\textbf{true} \wedge ein) \vee (\textbf{false} \wedge \neg ein) \vee (\textbf{false} \wedge ein) \vee (\textbf{false} \wedge \neg ein)$.)

Herleitung: (Sei $A \equiv z + c = x \wedge c < x$.)

(1)	$exec\,\lambda_z \wedge A \wedge c < x - 1 \rightarrow \circ A_z(z-1)$	(Z5),(data),(pred)
(2)	$exec\,\lambda_z \wedge A \wedge c = x - 1 \rightarrow \circ(c = x)$	(Z5),(data),(pred)
(3)	$exec\,\lambda_z \wedge H_{\lambda_z} \wedge A \rightarrow \circ(c = x \vee \exists \bar{z}(\bar{z} < z \wedge A_z(\bar{z})))$	(pred),(1),(2)
(4)	$exec\,\lambda_z \wedge \neg H_{\lambda_z} \wedge A \rightarrow \circ(c = x \vee \exists \bar{z}(\bar{z} \leq z \wedge A_z(\bar{z})))$	(taut)
(5)	$exec\,\lambda \wedge H_\lambda \wedge A \rightarrow \circ(c = x \vee \exists \bar{z}(\bar{z} < z \wedge A_z(\bar{z})))$ für $\lambda \in \{\lambda_{ein}, \lambda_{aus}, \lambda_p\}$	(taut)
(6)	$exec\,\lambda \wedge \neg H_\lambda \wedge A \rightarrow \circ(c = x \vee \exists \bar{z}(\bar{z} \leq z \wedge A_z(\bar{z})))$ für $\lambda \in \{\lambda_{ein}, \lambda_{aus}, \lambda_p\}$	(Z3′),(Z4),(Z6), (data),(pred)
(7)	$\Box\neg ein \rightarrow \Box\Diamond\neg ein$	(T8),(T30)
(8)	$\Box\neg ein \rightarrow \Diamond exec\,\lambda_{ein}$	(fair),(7),(prop)
(9)	$exec\,\lambda_{ein} \rightarrow \circ ein$	(Z3′),(prop)
(10)	$exec\,\lambda_{ein} \rightarrow \Diamond ein$	(som),(9)
(11)	$\Box\neg ein \rightarrow \Diamond ein$	(chain),(8),(10)
(12)	$\Box A \rightarrow \Diamond ein$	(prop),(11)
(13)	$\Box A \rightarrow \Diamond(c = x \vee E_{\Gamma_c''})$	(T26),(prop),(12)
(14)	$\exists z A \rightarrow \Diamond(c = x)$	(fairwfr),(3)–(6),(13)
(15)	$c = 0 \rightarrow \Diamond(c = x)$	(T5),(data),(pred),(14)

Das alternating bit protocol

Nachrichtenübertragung zwischen einem „Sender“ und einem „Empfänger“: Der Sender erhält fortwährend Nachrichten als Eingabe und überträgt diese zum Empfänger, der sie jeweils ausgibt und eine Bestätigung an den Sender zurückschickt. Übertragungen können (in beiden Richtungen) „zerstört“ werden. Trotzdem soll der Empfänger alle beim Sender eingegangenen Nachrichten in der gleichen Reihenfolge ausgeben.

Lösung durch das ***alternating bit protocol***: Die Nachrichten werden zusammen mit einem Kontrollbit versendet, dessen Wert als Bestätigung zurückgeschickt und für die jeweils nächste Nachricht alterniert wird.

Darstellung als rfSTS Γ_{ABP} mit

$Act_{\Gamma_{ABP}} = \{\alpha_0, \alpha_1, \beta_0, \beta_1\}$,
$X = X_{NAT} \cup X_{MESSAGE}$,
$X_{NAT} = \{sb, rb, wb, ack\}$,
$X_{MESSAGE} = \{sm, rm\}$,
$V = \{exec\,\lambda \mid \lambda \in Act_{\Gamma_{ABP}}\}$,
$enabled_\lambda \equiv$ **true** für alle $\lambda \in Act_{\Gamma_{ABP}}$,
$start_{\Gamma_{ABP}} \equiv sb = 0 \wedge rb = 1 \wedge wb = 0 \wedge ack = 1 \wedge sm = FIRSTMESSAGE$.

Dabei zugrundegelegt:

- Sorte NAT für die Bitwerte 0 und 1 mit der zusätzlichen Konstanten ERR (zerstörter Wert) und Funktionszeichen $\oplus$ (Addition modulo 2),
- Sorte $MESSAGE$ für die Nachrichten mit den Konstanten ERR (zerstörter Wert) und $FIRSTMESSAGE$ (erste zu sendende Nachricht) sowie Funktionszeichen $NEXTMESSAGE^{(MESSAGE\ MESSAGE)}$ (jeweils nächste zu sendende Nachricht).

Intuitive Bedeutungen:

α_0: Sender sendet Nachricht sm und Bit sb;
der Empfänger erhält diese als rm und rb;
β_0: Empfänger gibt rm aus und verändert den jeweils erwarteten Bitwert wb,
falls rb und wb gleich sind;
β_1: Empfänger sendet den betreffenden Bitwert (neu oder alt) zurück;
der Sender erhält diesen als ack;
α_1: Sender bestimmt die als nächstes zu sendenden Werte, falls ack und sb gleich sind.

Spezifikation von Γ_{ABP} (die trivialen (action)-Axiome sind weggelassen):

(data) Alle in $\mathsf{S}_{\Gamma_{ABP}}$ gültigen Zustandsformeln,

(root) $\mathbf{init} \rightarrow start_{\Gamma_{ABP}}$,

(nil) $nil_{\Gamma_{ABP}} \wedge A \rightarrow \circ A$ falls A Zustandsformel von Γ_{ABP},

(ABP1) $exec\,\alpha_0 \rightarrow ((rm' = sm \wedge rb' = sb) \vee (rm' = ERR \wedge rb' = ERR)) \wedge unchanged(sb, wb, ack, sm)$,

(ABP2) $exec\,\alpha_1 \wedge sb = ack \rightarrow sm' = NEXTMESSAGE(sm) \wedge sb' = sb \oplus 1 \wedge unchanged(rb, wb, ack, rm)$,

(ABP3) $exec\,\alpha_1 \wedge sb \neq ack \rightarrow unchanged(sb, rb, wb, ack, sm, rm)$,

(ABP4) $exec\,\beta_0 \wedge rb = wb \rightarrow wb' = wb \oplus 1 \wedge unchanged(sb, rb, ack, sm, rm)$,

(ABP5) $exec\,\beta_0 \wedge rb \neq wb \rightarrow unchanged(sb, rb, wb, ack, sm, rm)$,

(ABP6) $exec\,\beta_1 \rightarrow (ack' = wb \oplus 1 \vee ack' = ERR) \wedge unchanged(sb, rb, wb, sm, rm)$.

Dabei verwendete Abkürzung:

$$unchanged(a_1, \ldots, a_n) \quad \text{für} \quad \bigwedge_{i=1}^{n} a_i' = a_i \quad (a_1, \ldots, a_n \text{ Systemvariablen}).$$

Beachte: Ausgabe bei β_0 ist nicht explizit formuliert. Die Formel

$$output \;\equiv\; exec\,\beta_0 \wedge rb = wb$$

repräsentiert die Tatsache, dass der Empfänger die Nachricht rm ausgibt.

Behauptung: a) $start_{\Gamma_{ABP}} \rightarrow rm = FIRSTMESSAGE$ **atnext** $output$.
b) $output \wedge rm = x \rightarrow rm = NEXTMESSAGE(x)$ **atnext** $output$.

Beweisidee:
Beide Teile a) und b) der Behauptung sind von der Form $A' \rightarrow B$ **atnext** $output$;
zu finden: Formel A (Invariante) mit

i) $A' \rightarrow A$,
ii) $exec\,\lambda \wedge A \rightarrow \circ(output \rightarrow B) \wedge \circ(\neg output \rightarrow A)$ für alle $\lambda \in Act_{\Gamma_{ABP}}$,
iii) $nil_{\Gamma_{ABP}} \wedge A \rightarrow (output \rightarrow B)$.

Aus ii) und iii) folgt dann $A \rightarrow B$ **atnext** $output$ mit (invatnext), und mit i) folgt daraus $A' \rightarrow B$ **atnext** $output$.

Geeignetes A:

Für a): $wb = sb \wedge sm = FIRSTMESSAGE \wedge \neg output.$

Für b): $(output \rightarrow rm = x) \wedge$
$(\neg output \wedge wb \neq sb \rightarrow sm = x) \wedge$
$(\neg output \wedge wb = sb \rightarrow sm = NEXTMESSAGE(x)).$

Zur jeweiligen Herleitung von ii) wird maßgeblich die mit (invstart) herleitbare Formel

$$\Box((rb = wb \rightarrow wb = sb \wedge rm = sm) \wedge (sb = ack \rightarrow wb \neq sb))$$

benutzt („immer geltender“ Zusammenhang zwischen den Systemvariablen).

Kapitel 6
Verifikation paralleler Programme

6.1 Programme und Transitionssysteme

Temporale Semantik von Programmen

- (Imperative) Programme beschreiben „Berechnungsabläufe“, können also als Zustandssysteme aufgefasst werden.
- Formalere Sprechweise: Programme beschreiben Transitionssysteme in einer „ausführbaren“, ***algorithmischen*** Form. Das jeweils beschriebene Transitionssystem ist die formal gegebene ***operationale Semantik*** eines Programms.
- Damit: Temporale Logik anwendbar
  - zur (temporallogischen, axiomatischen, abstrakten) Spezifikation (Darstellung) der von Programmen beschriebenen Transitionssysteme (***temporale Semantik*** von Programmen),
  - (darauf basierend:) zur formalen „Verifikation von Programmen“ (d.h.: Verifikation von Eigenschaften des durch ein Programm beschriebenen Transitionssystems, kurz: ***Programmeigenschaften***).
- Besonders erfolgreiche Anwendungen: bei ***parallelen Programmen***.

Parallele Programme

Grundstruktur paralleler Programme:

- Gebildet mit der ***Parallelanweisung***

  cobegin $\Pi_1 \parallel \ldots \parallel \Pi_p$ **coend**

  mit Programmen (***Prozessen***) $\Pi_1, \ldots, \Pi_p$, $p \geq 1$.

  Im Folgenden zur Vereinfachung angenommen: Prozesse sind ***sequentiell*** (d.h.: sie enthalten selbst keine Parallelanweisung).

Interleaving-Modell
(grundlegendes Prinzip der operationalen Semantik paralleler Programme):

- Alle Prozesse einer Parallelanweisung sind zusammengesetzt aus ausgezeichneten ***atomaren Schritten***.

- Eine Ausführung der Parallelanweisung bedeutet eine sequentielle Ausführung der einzelnen atomaren Schritte derart, dass
  - Schritte aus verschiedenen Prozessen in beliebiger Reihenfolge ausgeführt werden,
  - die (relative) Reihenfolge von Schritten, die zum gleichen Prozess gehören, durch diesen bestimmt ist.

Beispiel

Sei Π das parallele Programm

cobegin $\alpha; \beta \parallel \gamma; \delta$ **coend**

mit: $\alpha: b := b * 2,\ \beta: V(a),\ \gamma: b := b + 1,\ \delta: P(a)$,
zu Programmbeginn: $a = 0$.
(P und V seien die üblichen Semaphor-Operationen.)

Π beschreibt folgendes rfSTS (ebenfalls mit Π bezeichnet) über SIG_{NAT} und N:

$Act_\Pi = \{\alpha, \beta, \gamma, \delta\}$,
$X_\Pi = \{a, b\}$,
$V_\Pi = \{exec\,\alpha, exec\,\beta, exec\,\gamma, exec\,\delta, at\,\alpha, at\,\beta, at\,\gamma, at\,\delta\}$
($at\,\lambda$: „λ ist einer der als nächstes möglichen atomaren Schritte“),
$start_\Pi \equiv at\,\alpha \wedge at\,\gamma \wedge a = 0$,
$enabled_\alpha \equiv at\,\alpha$, $enabled_\beta \equiv at\,\beta$, $enabled_\gamma \equiv at\,\gamma$, $enabled_\delta \equiv at\,\delta \wedge a > 0$,
$Z_\Pi = \{\eta : X_\Pi \cup V_\Pi \to \mathbb{N} \cup \{\mathsf{ff}, \mathsf{tt}\} \mid \eta(a), \eta(b) \in \mathbb{N},\ \eta(v) \in \{\mathsf{ff}, \mathsf{tt}\}$ für $v \in V_\Pi$,
η zulässig,
$\eta(exec\,\lambda) = \mathsf{tt}$ für höchstens ein $\lambda \in Act_\Pi$,
nicht $\eta(at\,\alpha) = \eta(at\,\beta) = \mathsf{tt}$,
nicht $\eta(at\,\gamma) = \eta(at\,\delta) = \mathsf{tt}\}$,
$T' = \{(\eta, \eta') \in Z \times Z \mid$
falls $\eta(exec\,\alpha) = \mathsf{tt}$, so $\eta'(a) = \eta(a), \eta'(b) = \eta(b) * 2$,
$\eta'(at\,\alpha) = \mathsf{ff}, \eta'(at\,\beta) = \mathsf{tt}, \eta'(at\,\gamma) = \eta(at\,\gamma), \eta'(at\,\delta) = \eta(at\,\delta)$,
falls $\eta(exec\,\beta) = \mathsf{tt}$, so $\eta'(a) = \eta(a) + 1, \eta'(b) = \eta(b)$,
$\eta'(at\,\beta) = \mathsf{ff}, \eta'(at\,\lambda) = \eta(at\,\lambda)$ für $\lambda \neq \beta$,
falls $\eta(exec\,\gamma) = \mathsf{tt}$, so $\eta'(a) = \eta(a), \eta'(b) = \eta(b) + 1$,
$\eta'(at\,\gamma) = \mathsf{ff}, \eta'(at\,\delta) = \mathsf{tt}, \eta'(at\,\alpha) = \eta(at\,\alpha), \eta'(at\,\beta) = \eta(at\,\beta)$,
falls $\eta(exec\,\delta) = \mathsf{tt}$, so $\eta'(a) = \eta(a) - 1, \eta'(b) = \eta(b)$,
$\eta'(at\,\delta) = \mathsf{ff}, \eta'(at\,\lambda) = \eta(at\,\lambda)$ für $\lambda \neq \delta\}$,
$T_\Pi = tot(T')$.

Möglicher Ablauf:

$[0, 4, at\,\alpha, at\,\gamma, exec\,\alpha] \longrightarrow [0, 8, at\,\beta, at\,\gamma, exec\,\gamma] \longrightarrow$
$[0, 9, at\,\beta, at\,\delta, exec\,\beta] \longrightarrow [1, 9, at\,\delta, exec\,\delta] \longrightarrow [0, 9] \longrightarrow [0, 9] \longrightarrow \ldots$

(η bezeichnet durch $[\eta(a), \eta(b), v_1, v_2, \ldots \in V_\Pi$ mit $\eta(v_i) = \mathsf{tt}]$.)

Der Zustand $[0, 9]$ ist ein ***Fangzustand*** (es werden keine Aktionen mehr ausgeführt) und repräsentiert die Terminierung des Programms.

Allgemein:

- Typische Form der Ausführbarkeitsbedingungen $enabled_\lambda$:

  $at\,\lambda \wedge C_\lambda$ mit Zustandsformel C_λ.

- Damit i. Allg. zwei verschiedene Arten von Fangzuständen (η_i mit $\eta_i(exec\,\lambda) = \mathsf{ff}$ für alle $\lambda \in Act$):
  - $\eta_i(at\,\lambda) = \mathsf{ff}$ für alle $\lambda \in Act$
    („normale“ Terminierung, Programm ist am „gewollten“ Ende),
  - nicht $\eta_i(at\,\lambda) = \mathsf{ff}$ für alle $\lambda \in Act$, aber $\mathsf{K}_i^{(\xi)}(C_\lambda) = \mathsf{ff}$ für die $\lambda \in Act$ mit $\eta_i(at\,\lambda) = \mathsf{tt}$
    (***Verklemmung***, Programm ist nicht am gewollten Ende, kann aber nicht fortfahren).

6.2 Eine einfache Programmiersprache für reaktive Systeme

Reaktive Systeme

- Im Folgenden betrachtet: Parallele Programme einer Programmiersprache PAR, die nicht terminieren (sollen). Die dadurch implementierten Systeme heißen ***reaktive Systeme***.

  Außerdem: Programme mit „Initialisierungsbedingung“ (im Beispiel: $a = 0$).

  (Grundkonzept in PAR: „Kommunikation über gemeinsame Variablen“;
  analog behandelbar: Programme mit „Kommunikation durch (synchronen) Nachrichtenaustausch“.)

- Bei der Formalisierung von reaktiven Systemen als STS wird die „echte“ Parallelität durch das Interleaving-Modell „verfälscht“. Dies wird „repariert“ durch die Hinzunahme von Fairness.

- Zusammen also: Programme von PAR beschreiben rfSTS.

PAR-Programme

Für ein Programm Π der Sprache PAR (kurz: PAR-Programm) sei eine Signatur SIG_Π und eine Struktur S_Π für SIG_Π gegeben.

Π hat die Gestalt

var Δ
start R
cobegin $\Pi_1 \parallel \ldots \parallel \Pi_p$ **coend**

mit $p \geq 1$. Dabei ist

- Δ eine Folge $\Delta_1; \ldots; \Delta_q$ von ***Variablendeklarationen*** der Form

  $a_1, \ldots, a_r : s$

  mit einer Sorte s von SIG_Π. $a_1, \ldots, a_r$ sind die ***Programmvariablen*** der Sorte s von Π;

- R eine Formel (***Initialisierungsbedingung***) von $\mathcal{L}_{\mathrm{FOL}}(SIG^*_\Pi)$ (kurz: $\mathcal{L}_\Pi$) mit der Signatur SIG^*_Π, die aus SIG_Π entsteht, wenn man die Programmvariablen (jeder Sorte s) von Π zu den Konstanten (gleicher Sorte) von SIG_Π hinzunimmt;

- jedes Π_i, $i = 1, \ldots, p$, ein *Prozess* gemäß folgender BNF-Syntax:

  $$
  \begin{aligned}
  \mathit{Prozess} \;::=\; & \textbf{loop}\ \mathit{Anweisungsfolge}\ \textbf{end} \\
  \mathit{Anweisungsfolge} \;::=\; & \mathit{markierte\ Anweisung} \mid \\
  & \mathit{markierte\ Anweisung}; \mathit{Anweisungsfolge} \\
  \mathit{markierte\ Anweisung} \;::=\; & \mathit{Marke} : \mathit{Anweisung} \\
  \mathit{Anweisung} \;::=\; & \mathit{Zuweisung} \mid \\
  & \textbf{if}\ \mathit{Bedingung}\ \textbf{then}\ \mathit{Anweisungsfolge} \\
  & \qquad \{\textbf{else}\ \mathit{Anweisungsfolge}\}^1_0\ \textbf{endif} \mid \\
  & \textbf{await}\ \mathit{Bedingung}\ \{\textbf{then}\ \mathit{Zuweisung}\}^1_0 \\
  \mathit{Zuweisung} \;::=\; & \mathit{Programmvariable} := \mathit{Term}
  \end{aligned}
  $$

  und mit den zusätzlichen Festlegungen:

  - Eine *Marke* ist ein Element einer gegebenen Menge von Marken. (Bezeichnungen: $\lambda, \alpha_0, \alpha_1, \alpha_2, \ldots, \beta_0, \beta_1, \beta_2, \ldots$ u.ä.)
  - Alle in Π auftretenden Marken sind paarweise verschieden und identifizieren somit eindeutig alle (Vorkommen von) Anweisungen.
  - Eine *Bedingung* ist eine geschlossene Formel von $\mathcal{L}_\Pi$, ein *Term* ist ein Term von $\mathcal{L}_\Pi$.
  - Programmvariable und Term in einer *Zuweisung* haben gleiche Sorte.

Beispiel:

Programm Π_{bsp} mit $SIG_{\Pi_{bsp}} = SIG_{NAT}$ und $\mathsf{S}_{\Pi_{bsp}} = \mathsf{N}$:

Π_{bsp} $\equiv$ **var** $a, b : NAT$
start $a = 0 \wedge b = 1$
cobegin loop $\alpha_0 :\ b := b * b;$
$\alpha_1 :$ **await** $a \neq 0$ **then** $a := 1;$
$\alpha_2 :$ **if** $a < b$ **then** $\alpha_3 :\ b := b - a$ **endif**;
$\alpha_4 :\ b := 2 * b$
endloop
$\|$
loop $\beta_0 :\ a := a + 2;$
$\beta_1 :\ b := 2 * b$
endloop
coend

Informelle Semantik von PAR-Programmen

- **loop** AF **end**: fortwährende (endlose) Wiederholung der Anweisungsfolge AF,
- $A_1; A_2$: Hintereinanderausführen der Anweisungen A_1 und A_2 (wie üblich),
- **if** ... **endif**: bedingte Anweisung (wie üblich),
- **await** B **then** ZU: kann nur ausgeführt werden, wenn B wahr ist; Wirkung gemäß der Zuweisung ZU (falls vorhanden, sonst keine Wirkung) (***Synchronisationsanweisung***),
- atomare Schritte beim interleaving: Alle Vorkommen von
  - Zuweisungen außerhalb von Synchronisationsanweisungen,
  - Synchronisationsanweisungen,
  - Test der Bedingung B in **if** B **then** ... **endif** zusammen mit dem Sprung an die entsprechende Fortsetzungsstelle.

Beispiel:

Ein möglicher (Programm-) Ablauf von Π_{bsp} (s.o.):

Schritt	nach dem Schritt: Wert von	a	b
	zu Beginn:	0	1
α_0		0	1
β_0		2	1
α_1		1	1
α_2	($\rightsquigarrow \alpha_4$)	1	1
β_1		1	2
α_4		1	4
α_0		1	16
β_0		3	16
α_1		1	16
α_2	($\rightsquigarrow \alpha_3$)	1	16
α_3		1	15
$\vdots$			

Dabei: Atomare Schritte identifiziert durch die entsprechenden Marken.

PAR-Programme und Transitionssysteme

Ein PAR-Programm

$$\begin{aligned}\Pi \;\equiv\; & \textbf{var } \Delta \\ & \textbf{start } R \\ & \textbf{cobegin } \Pi_1 \parallel \ldots \parallel \Pi_p \textbf{ coend}\end{aligned}$$

beschreibt das (ebenfalls mit Π bezeichnete) rfSTS $\Pi = (X, V, Z, T, Act, start, \mathcal{E})$ über SIG_Π und S_Π mit

$Act =$ Menge aller in Π auftretenden Marken,
$X =$ Menge aller in Δ vereinbarten Programmvariablen (der jeweiligen Sorten),
$V = \{exec\,\lambda, at\,\lambda \mid \lambda \in Act\}$,
$start \equiv R \wedge at\,\alpha_0^{(1)} \wedge \ldots \wedge at\,\alpha_0^{(p)}$ ($\alpha_0^{(i)}$: „Anfangsmarken“ von Π_i, $i = 1, \ldots, p$).
$enabled_\lambda \equiv \begin{cases} at\,\lambda \wedge B & \text{falls } \lambda \text{ Marke einer Anweisung } \textbf{await } B \ (\textbf{then } \ldots) \\ at\,\lambda & \text{sonst,} \end{cases}$
$Z = \{\eta : X \cup V \to |\mathsf{S}| \cup \{\mathsf{ff}, \mathsf{tt}\} \mid$
$\eta(a) \in |\mathsf{S}|_s$ für $a \in X_s, s \in \mathbf{S}_\Pi, \eta(v) \in \{\mathsf{ff}, \mathsf{tt}\}$ für $v \in V$,
η zulässig,
$\eta(exec\,\lambda) = \mathsf{tt}$ für höchstens ein $\lambda \in Act$,
$\eta(at\,\lambda) = \mathsf{tt}$ für genau ein $\lambda \in Act_{\Pi_i} (i = 1, \ldots, p)\}$,
$T = tot(T'),\ T' = \{(\eta, \eta') \in Z \times Z \mid$ „Beschreibung der Wirkung der $\lambda \in Act$“$\}$.

(Act_{Π_j} ist die Menge der Marken in Π_j.)

Beispiel:

Der oben betrachtete Programmablauf des Programms Π_{bsp} wird formalisiert durch den Ablauf

$$\begin{aligned}&[0, 1, at\,\alpha_0, at\,\beta_0, exec\,\alpha_0] \longrightarrow [0, 1, at\,\alpha_1, at\,\beta_0, exec\,\beta_0] \longrightarrow \\ &\quad [2, 1, at\,\alpha_1, at\,\beta_1, exec\,\alpha_1] \longrightarrow [1, 1, at\,\alpha_2, at\,\beta_1, exec\,\alpha_2] \longrightarrow \\ &\qquad [1, 1, at\,\alpha_4, at\,\beta_1, exec\,\beta_1] \longrightarrow [1, 2, at\,\alpha_4, at\,\beta_0, exec\,\alpha_4] \longrightarrow \\ &\qquad\quad [1, 4, at\,\alpha_0, at\,\beta_0, exec\,\alpha_0] \longrightarrow [1, 16, at\,\alpha_1, at\,\beta_0, exec\,\beta_0] \longrightarrow \ldots\end{aligned}$$

des rfSTS Π_{bsp} (η bezeichnet durch $[\eta(a), \eta(b), v_1, v_2, \ldots \in V_{\Pi_{bsp}}$ mit $\eta(v_i) = \mathsf{tt}]$).

Temporale Semantik von PAR-Programmen

Die temporale Semantik eines PAR-Programms Π ist gegeben durch eine temporallogische Spezifikation $(\mathcal{L}_{\mathrm{T}\Pi}, \mathcal{A}_\Pi)$ des (von Π beschriebenen) Transitionssystems Π mit:

- $\mathcal{L}_{\mathrm{T}\Pi} = \mathcal{L}_{\mathrm{T}}(TSIG_\Pi)$ mit $TSIG_\Pi = (SIG_\Pi, X_\Pi, V_\Pi)$,

- $\mathcal{A}_\Pi$ enthält:
  - (data_Π), (root_Π), (nil_Π), (action_Π), (fair_Π),
  - Axiome zur Beschreibung von Z_Π und T_Π.

Allgemeines Schema für die Zusammenstellung der Axiome zur Beschreibung von Z_Π und T_Π (direkt anhand der Gestalt des Programms, ohne explizite Angabe des Transitionssystems):

1. Beschreibung des interleaving und Eindeutigkeit des „Befehlszählers“ in jedem Prozess

   (I) $exec\,\lambda_1 \rightarrow \neg exec\,\lambda_2$ für $\lambda_1, \lambda_2 \in Act_\Pi, \lambda_1 \neq \lambda_2$

   (PC) $at\,\lambda_1 \rightarrow \neg at\,\lambda_2$, für Marken λ_1, λ_2 des gleichen Prozesses, $\lambda_1 \neq \lambda_2$

2. Beschreibung der („durch $\lambda \in Act_\Pi$ gegebenen“) Transitionen

   a) Kontrollflussaxiome (Wirkung auf $at\,\lambda$)

   - Für jeden atomaren Schritt $\lambda_1 \rightsquigarrow \lambda_2$ vermöge einer Zuweisung oder einer Synchronisationsanweisung:

     (C1) $exec\,\lambda_1 \rightarrow \circ at\,\lambda_2$

   - Für jeden atomaren Schritt $\lambda \rightsquigarrow \begin{cases} \lambda_1 & (B) \\ \lambda_2 & (\neg B) \end{cases}$ vermöge λ : **if** B **then** ...:

     (C2) $exec\,\lambda \rightarrow (B \wedge \circ at\,\lambda_1) \vee (\neg B \wedge \circ at\,\lambda_2)$

   - „In nichts ausführenden Prozessen ändert sich der Befehlszähler nicht“:

     (C3) $exec\lambda_1 \wedge at\lambda_2 \rightarrow \circ at\lambda_2$ für alle $\lambda_1 \in Act_{\Pi_i}, \lambda_2 \in Act_\Pi \setminus Act_{\Pi_i}$, $i = 1, \ldots, p$

   b) Beschreibung der Änderung der Programmvariablen

   - Für $\lambda : a := t$ und λ : **await** B **then** $a := t$:

     (ASSIGN) $exec\,\lambda \rightarrow a' = t \wedge \bigwedge_{b \in X_\Pi \setminus \{a\}} b' = b$

   - für λ : **if** B **then** ... und λ : **await** B:

     (UNCH) $exec\,\lambda \rightarrow \bigwedge_{a \in X_\Pi} a' = a$

Beispiel:

Temporale Semantik des Programms Π_{bsp} (s.o.):

(data): Alle Zustandsformeln A mit $\models_{\mathsf{N}} A$,

(root): **init** $\rightarrow a = 0 \wedge b = 1 \wedge at\,\alpha_0 \wedge at\,\beta_0$,

(nil): $nil_{\Pi_{bsp}} \wedge A \rightarrow \circ A$ für Zustandsformeln A von Π_{bsp},

(action): $exec\,\lambda \rightarrow at\,\lambda$ für $\lambda \in \{\alpha_0, \alpha_2, \alpha_3, \alpha_4, \beta_0, \beta_1\}$,
$exec\,\alpha_1 \rightarrow at\,\alpha_1 \wedge a \neq 0$,

(fair): $\Box\Diamond at\,\lambda \rightarrow \Diamond exec\,\lambda$ for $\lambda \in \{\alpha_0, \alpha_2, \alpha_3, \alpha_4, \beta_0, \beta_1\}$,
$\Box\Diamond(at\,\alpha_1 \wedge a \neq 0) \rightarrow \Diamond exec\,\alpha_1$,

(I): $exec\,\lambda_1 \rightarrow \neg exec\,\lambda_2$ für $\lambda_1, \lambda_2 \in \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \beta_0, \beta_1\}, \lambda_1 \neq \lambda_2$,

(PC): $at\,\alpha_i \rightarrow \neg at\,\alpha_j$ für $i, j \in \{0, 1, 2, 3, 4\}, i \neq j$,
$at\,\beta_i \rightarrow \neg at\,\beta_j$ für $i, j \in \{0, 1\}, i \neq j$,

(C1): $exec\,\alpha_0 \rightarrow \circ at\,\alpha_1$,
(usw.),

(C2): $exec\,\alpha_2 \rightarrow (a < b \wedge \circ at\,\alpha_3) \vee (a \geq b \wedge \circ at\,\alpha_4)$,

(C3): $exec\,\alpha_i \wedge at\,\beta_j \rightarrow \circ at\,\beta_j$ für $i \in \{0, 1, 2, 3, 4\}, j \in \{0, 1\}$,
$exec\,\beta_j \wedge at\,\alpha_i \rightarrow \circ at\,\alpha_i$ für $i \in \{0, 1, 2, 3, 4\}, j \in \{0, 1\}$,

(ASSIGN): $exec\,\alpha_0 \rightarrow b' = b * b \wedge a' = a$,
(usw.),

(UNCH): $exec\,\alpha_2 \rightarrow a' = a \wedge b' = b$.

6.3 Verifikationsbeispiele

Programmeigenschaften

In $\mathcal{L}_{\mathrm{T}\Pi}$ sind viele Eigenschaften eines Programms Π ausdrückbar. Wichtige solche Eigenschaften sind:

- ***Terminierung***, ***partielle Korrektheit***, ***totale Korrektheit*** von (sequentiellen oder parallelen) Programmen, die „Ergebnisse berechnen“ (nicht für reaktive Systeme),
- Grundlegende Synchronisationsanforderungen an parallele Programme wie ***wechselseitiger Ausschluss***, ***Verklemmungsfreiheit***, ***Aushungerungsfreiheit*** u.ä.,
- Einhaltung gewisser „Ausführungsreihenfolgen“ bei parallelen Programmen, z.B. ***Bedienstrategien***.

Beispiele:

1. Sei Π Programm mit einer Parallelanweisung der Gestalt

   cobegin loop $\vdots$
   α_i : „Beginn kritischer Abschnitt“;
   $\vdots$
   α_j : „Ende kritischer Abschnitt“;
   $\vdots$
   endloop
   $\|$
   loop $\vdots$
   β_k : „Beginn kritischer Abschnitt“;
   $\vdots$
   β_l : „Ende kritischer Abschnitt“;
   $\vdots$
   endloop
   coend

   Wechselseitiger Ausschluss der kritischen Abschnitte (i. Allg. unter Annahme einer Bedingung P zu Beginn der Programmabarbeitung):

   $$start_\Pi \wedge P \rightarrow \Box\neg((at\,\alpha_i \vee \ldots \vee at\,\alpha_j) \wedge (at\,\beta_k \vee \ldots \vee at\,\beta_l)).$$

2. Sei Π Programm mit einer Parallelanweisung der Gestalt

   cobegin loop $\vdots$
   α_i : **await** B_1 $\ldots$;
   $\vdots$
   endloop
   $\|$
   loop $\vdots$
   β_k : **await** B_2 $\ldots$;
   $\vdots$
   endloop
   coend

   (ohne weitere Synchronisationsanweisungen).

   Verklemmungsfreiheit:

   $$start_\Pi \wedge P \rightarrow \Box(at\,\alpha_i \wedge at\,\beta_k \rightarrow B_1 \vee B_2).$$

   Aushungerungsfreiheit (für jeden der beiden Prozesse in Π):

   $$at\,\alpha_i \rightarrow \Diamond exec\,\alpha_i,$$
   $$at\,\beta_k \rightarrow \Diamond exec\,\beta_k.$$

3. Sei Π Programm mit Prozessen $\Pi_1, \ldots, \Pi_q, \Psi$ der Gestalt ($i = 1, \ldots, q$):

$$
\begin{array}{lll}
\Pi_i & \equiv & \textbf{loop} \; \vdots \\
& & \quad \alpha_1^{(i)} : \text{„fordere Betriebsmittel BM an“}; \\
& & \quad \vdots \\
& & \quad \alpha_2^{(i)} : \text{„erhalte BM“}; \\
& & \quad \vdots \\
& & \textbf{endloop}, \\
\Psi & \equiv & \textbf{loop} \; \vdots \\
& & \quad \beta : \text{„teile BM an } \Pi_k \; (1 \leq k \leq q) \text{ zu“}; \\
& & \quad \vdots \\
& & \textbf{endloop}
\end{array}
$$

Schreibweise: $ford_i$ für $at\,\lambda_1^{(i)} \vee \ldots \vee at\,\lambda_l^{(i)} \vee at\,\alpha_2^{(i)}$
($\lambda_1^{(i)} \ldots \lambda_l^{(i)}$ Marken zwischen $\alpha_1^{(i)}$ und $\alpha_2^{(i)}$, $i = 1, \ldots, q$).

„Zuteilung nur an Prozess, der BM angefordert hat“:

$$\neg ford_i \rightarrow \neg(exec\,\beta \wedge k = i) \wedge exec\,\alpha_1^{(i)} \; \textbf{before} \; (exec\,\beta \wedge k = i).$$

FIFO-Zuteilungsstrategie:

$$exec\,\alpha_1^{(i)} \wedge \neg ford_j \rightarrow \neg(exec\,\beta \wedge k = j) \; \textbf{unless} \; (exec\,\beta \wedge k = i).$$

Ein Erzeuger-Verbraucher-Programm

$$
\begin{array}{lll}
\Pi_{pc} & \equiv & \textbf{var} \; be, bf : NAT; \ldots \\
& & \textbf{start} \; bf = 0 \wedge be = n \\
& & \textbf{cobegin loop} \; \alpha_0 : \; \vdots \; \text{(„produziere Objekt“)} \\
& & \qquad\qquad \alpha_1 : \; \textbf{await} \; be > 0 \; \textbf{then} \; be := be - 1; \\
& & \qquad\qquad\quad \vdots \; \text{(„speichere Objekt im Puffer“)} \\
& & \qquad\qquad \alpha_2 : \; bf := bf + 1 \\
& & \quad \textbf{endloop} \\
& & \quad \| \\
& & \quad \textbf{loop} \; \beta_0 : \; \textbf{await} \; bf > 0 \; \textbf{then} \; bf := bf - 1; \\
& & \qquad\qquad\quad \vdots \; \text{(„hole Objekt aus dem Puffer“)} \\
& & \qquad\qquad \beta_1 : \; be := be + 1; \\
& & \qquad\qquad\quad \vdots \; \text{(„verbrauche Objekt“)} \\
& & \quad \textbf{endloop} \\
& & \textbf{coend}
\end{array}
$$

Dabei: n: Konstante („Pufferkapazität“).
be: „Anzahl der freien Pufferplätze“.
bf: „Anzahl der besetzten Pufferplätze“.
Keine Änderungen von be und bf in den nicht ausgeführten Programmteilen.

Behauptung (Verklemmungsfreiheit):

$$start_{\Pi_{pc}} \wedge n > 0 \rightarrow \Box(at\,\alpha_1 \wedge at\,\beta_0 \rightarrow be > 0 \vee bf > 0).$$

Schreibweise: $at\,M$ für $at\,\lambda_1 \vee \ldots \vee at\,\lambda_n \quad (M = \{\lambda_1, \ldots, \lambda_n\})$.

Bezeichnungen: $M_1 =$ Menge der Marken in „produziere Objekt“ $\cup \{\alpha_1\}$,
$M_2 =$ Menge der Marken in „verbrauche Objekt“ $\cup \{\beta_0\}$.

Herleitungsskizze: (Sei $A \equiv at\,M_1 \wedge at\,M_2 \rightarrow be > 0 \vee bf > 0$)

(1)	$start_{\Pi_{pc}} \wedge n > 0 \rightarrow A$	(pred)
(2)	$A \rightarrow (at\,\alpha_1 \wedge at\,\beta_0 \rightarrow be > 0 \vee bf > 0)$	(taut)
(3)	A **invof** $Act_{\Pi_{pc}}$	mit temporaler Semantik von Π_{pc} (nicht explizit angegeben)
(4)	$A \rightarrow \Box(at\,\alpha_1 \wedge at\,\beta_0 \rightarrow be > 0 \vee bf > 0)$	(inv),(2),(3)
(5)	$start_{\Pi_{pc}} \wedge n > 0 \rightarrow \Box(at\,\alpha_1 \wedge at\,\beta_0 \rightarrow be > 0 \vee bf > 0)$	(prop),(1),(4)

Der Peterson-Algorithmus

$\Pi_{Pet} \equiv$ **var** $a_1, a_2, t : NAT$
start $a_1 = 0 \wedge a_2 = 0 \wedge t = 1$
cobegin loop α_0 : „nichtkritisch“;
α_1 : $a_1 := 1$;
α_2 : $t := 1$;
α_3 : **await** $a_2 = 0 \vee t = 2$;
α_4 : „kritisch“;
α_5 : $a_1 := 0$
endloop
$\|$
loop β_0 : „nichtkritisch“;
β_1 : $a_2 := 1$;
β_2 : $t := 2$;
β_3 : **await** $a_1 = 0 \vee t = 1$;
β_4 : „kritisch“;
β_5 : $a_2 := 0$
endloop
coend

Dabei: $enabled_\lambda \equiv at\,\lambda$ für $\lambda \in \{\alpha_0, \alpha_4, \beta_0, \beta_4\}$,
Keine Änderungen von a_1, a_2, t in $\alpha_0, \alpha_4, \beta_0, \beta_4$.

Behauptung (Aushungerungsfreiheit des 1. Prozesses):

$$at\,\alpha_1 \rightarrow \Diamond at\,\alpha_4.$$

Beweisskizze: (Zugrundegelegt sei NAT mit $<$ als Sorte WF und $\prec$.)

a) Sei

$$\begin{array}{lcl} H_{\alpha_i} & \equiv & \textbf{true} \quad \text{für } i = 0,1,2,3,4,5, \\ H_{\beta_2} & \equiv & z = 2, \\ H_{\beta_3} & \equiv & z = 3, \\ H_{\beta_i} & \equiv & \textbf{false} \quad \text{für } i = 0,1,4,5 \end{array}$$

sowie

$$\begin{array}{lcl} A & \equiv & (at\,\alpha_1 \wedge z = 5) \vee \\ & & (at\,\alpha_2 \wedge z = 4) \vee \\ & & (at\,\alpha_3 \wedge at\,\beta_3 \wedge t = 1 \wedge z = 3) \vee \\ & & (at\,\alpha_3 \wedge (at\,\beta_4 \vee at\,\beta_5 \vee at\,\beta_0 \vee at\,\beta_1 \vee at\,\beta_2) \wedge z = 2) \vee \\ & & (at\,\alpha_3 \wedge at\,\beta_3 \wedge t \neq 1 \wedge z = 1), \end{array}$$

b) Dann sind folgende Formeln herleitbar:

$$\begin{array}{l} exec\,\lambda \wedge H_\lambda \wedge A \rightarrow \circ(at\,\alpha_4 \vee \exists \bar{z}(\bar{z} \prec z \wedge A_z(\overline{z}))) \quad \text{für alle } \lambda \in Act_{\Pi_{Pet}}, \\ exec\,\lambda \wedge \neg H_\lambda \wedge A \rightarrow \circ(at\,\alpha_4 \vee \exists \bar{z}(\bar{z} \preceq z \wedge A_z(\overline{z}))) \quad \text{für alle } \lambda \in Act_{\Pi_{Pet}}. \\ \Box A \rightarrow \Diamond(at\,\alpha_4 \vee E_{\Pi_{Pet}}) \end{array}$$

(mit $E_{\Pi_{Pet}} \equiv (H_{\alpha_0} \wedge enabled_{\alpha_0}) \vee \ldots \vee (H_{\beta_5} \wedge enabled_{\beta_5})$).

c) Aus b) erhält man

$$\exists z A \rightarrow \Diamond at\,\alpha_4$$

mit (fairwfr), und mit $at\,\alpha_1 \rightarrow A_z(5)$ ergibt sich $at\,\alpha_1 \rightarrow \exists z A$ und somit schließlich

$$at\,\alpha_1 \rightarrow \Diamond at\,\alpha_4.$$

Kapitel 7
Weitere temporale Logiken

7.1 Strukturierte Spezifikationen

Verfeinerungen von Spezifikationen

- „Größere“ Systeme sollten „strukturiert“ entwickelt und spezifiziert werden.
- Wichtige Strukturierungskonzepte: ***Verfeinerung*** und ***Komposition*** (Zusammensetzen von Systemen aus Teilsystemen, hier nicht behandelt).
- Verfeinerung: Die Entwicklung eines Systems geschieht in immer detaillierter werdenden „Stufen“, beschrieben in einer Folge

  $$\ldots \longrightarrow Spec_j \longrightarrow Spec_{j+1} \longrightarrow Spec_{j+2} \longrightarrow \ldots$$

  von Spezifikationen. Eine Spezifikation $Spec_k$ heißt ***Implementierung*** (oder auch: ***Verfeinerung***) von Spezifikationen $Spec_l$ mit $l < k$ und muss (zumindest) deren Anforderungen (neben eventuell weiteren) erfüllen.
- Typische Verfeinerungsart: Zustandsänderungen, die auf einer abstrakteren Stufe als „einschrittig“ betrachtet werden, bestehen auf detaillierterer Stufe aus einer längeren Folge von Transitionen und verwenden dort weitere („interne“) Systemvariablen.

Beispiel:

Zähler als STS Γ_{count} (vgl. Abschnitt 5.1)
Zusammenfassung der Spezifikation (ohne (data)) zu einer Formel:

$$A_{\Gamma_{count}} \equiv A_1 \wedge A_2$$

mit

$$A_1 \equiv ein \rightarrow (ein' \wedge c' = c + 1) \vee (\neg ein' \wedge c' = c),$$
$$A_2 \equiv \neg ein \rightarrow (\neg ein' \wedge c' = c) \vee (ein' \wedge c' = 0).$$

Implementierung von Γ_{count}:
Der Zählschritt $c \rightsquigarrow c+1$ ist realisiert durch Zählen „über die erste Dezimale von c“, z.B.:
Der Schritt $17 \rightsquigarrow 18$ wird zu

$$17.0 \rightsquigarrow 17.1 \rightsquigarrow 17.2 \rightsquigarrow 17.3 \rightsquigarrow \ldots \rightsquigarrow 17.9 \rightsquigarrow 18.0.$$

Spezifikation dieses verfeinerten Systems $\Gamma_{implcount}$: $A_{\Gamma_{implcount}} \equiv A_{impl1} \wedge A_{impl2}$ mit

$$
\begin{aligned}
A_{impl1} \;\equiv\; & ein \rightarrow (ein' \wedge dez' = (dez + 1) \bmod 10 \wedge \\
& \qquad (dez = 9 \rightarrow c' = c + 1) \wedge (dez \neq 9 \rightarrow c' = c)) \vee \\
& (\neg ein' \wedge c' = c \wedge dez' = dez), \\
A_{impl2} \;\equiv\; & \neg ein \rightarrow (\neg ein' \wedge c' = c \wedge dez' = dez) \vee (ein' \wedge c' = 0 \wedge dez' = 0).
\end{aligned}
$$

Korrektheit von Implementierungen

- Zwischen einem System Γ und seiner Implementierung Γ_{impl} besteht die Verfeinerungsbeziehung: Γ_{impl} verhält sich wie Γ (bezogen auf dessen Systemvariablen). Diese drückt die ***Korrektheit*** der Implementierung aus.
- Zur formalen Beschreibung dieser Beziehung sollte gelten:
  - Jedes Modell der Spezifikation von Γ_{impl} ist ein Modell der Spezifikation von Γ.

  Sind A_Γ und $A_{\Gamma_{impl}}$ die temporalen Spezifikationsaxiome von Γ bzw. Γ_{impl}, so wird dies ausgedrückt durch die (gegebenenfalls verifizierbare) logische Beziehung

  $$A_{\Gamma_{impl}} \models A_\Gamma.$$

Beispiel:

Für den Zähler Γ_{count} in obigem Beispiel ist die (formale) Verfeinerungsbeziehung

$$A_{\Gamma_{implcount}} \models A_{\Gamma_{count}}$$

nicht erfüllt. Z.B. ist (N, W) mit W gemäß

	$\ldots$	η_i	η_{i+1}	η_{i+2}	$\ldots$	η_{i+9}	η_{i+10}	η_{i+11}	η_{i+12}	$\ldots$
ein	$\ldots$	tt	tt	tt	$\ldots$	tt	tt	tt	tt	$\ldots$
c	$\ldots$	17	17	17	$\ldots$	17	18	18	18	$\ldots$
dez	$\ldots$	0	1	2	$\ldots$	9	0	1	2	$\ldots$

ein Modell von $A_{\Gamma_{implcount}}$, nicht jedoch von A_Γ.
Problem: $A_{\Gamma_{count}}$ erlaubt keine Übergänge $[\mathsf{tt}, 17] \longrightarrow [\mathsf{tt}, 17]$ (***Stotterschritte***).

Lösung: Spezifikation von Γ_{count} durch

$$A^r_{\Gamma_{count}} \;\equiv\; A_{\Gamma_{count}} \vee (ein' \leftrightarrow ein \wedge c' = c)$$

(„In jedem Schritt ändern sich ein und c gemäß $A_{\Gamma_{count}}$, oder sie bleiben unverändert“).
Dann gilt:

$$A_{\Gamma_{implcount}} \models A^r_{\Gamma_{count}}$$

(Allgemeiner: Entsprechende Erweiterung auch von $A_{\Gamma_{implcount}}$.)

Die Logik sTLA

Adäquate Logik sTLA gemäß obigen Überlegungen:

- Die Sprache von sTLA ist eine Teilsprache von FOLTL mit speziellen Formeln der Gestalt $\Box(A \vee (v' \leftrightarrow v))$ und $\Box(A \vee a' = a)$ (abgekürzt durch $\Box[A]_v$ bzw. $\Box[A]_a$) mit $v \in \mathcal{V}^F$, $a \in \mathcal{X}^F$ und folgenden Einschränkungen:
  - Operatoren $\circ$ (oder $'$) treten nur in diesen Formeln auf.
  - A enthält keine temporalen Operatoren, sondern nur noch (eventuell gestrichene) Symbole aus $\mathcal{V}^F \cup \mathcal{X}^F$.
- sTLA verwendet die (für FOLTL gegebene) Semantik mit initialer Gültigkeit (siehe Abschnitt 3.3).

Bemerkungen:

- Eine Formel $A \vee (a_1' = a_1 \wedge a_2' = a_2 \wedge \ldots)$ ist gleichwertig mit

  $$(A \vee a_1' = a_1) \wedge (A \vee a_2' = a_2) \wedge \ldots$$

  (analog auch mit $v_i' \leftrightarrow v_i$ statt $a_i' = a_i$).
- Wegen Verwendung der Semantik mit initialer Gültigkeit haben Spezifikationsaxiome für STS die Form $\Box[A]_{a_1} \wedge \Box[A]_{a_2} \wedge \ldots$, z.B. (für Γ_{count} in obigem Beispiel):

  $$\Box[A_{\Gamma_{count}}]_{ein} \wedge \Box[A_{\Gamma_{count}}]_c.$$

Eigenständige Syntax von sTLA

Sei $TSIG = (SIG, \mathcal{X}^F, \mathcal{V}^F)$ eine temporale Signatur und $\mathcal{L}_{\mathrm{FOL}}(SIG^+)$ eine Sprache erster Stufe mit der Variablenmenge $\mathcal{X} = \bigcup_{s \in \mathbf{S}} \mathcal{X}_s$ (SIG^+ wie in Abschnitt 4.2). Eine Sprache $\mathcal{L}_{\mathrm{sTLA}}(TSIG)$ (kurz: $\mathcal{L}_{\mathrm{sTLA}}$) von sTLA ist definiert wie folgt:

Alphabet:

- Alle Zeichen von $\mathcal{L}_{\mathrm{FOL}}(SIG^+)$,
- die Zeichen $\Box \mid ' \mid [\mid]$.

Terme (mit ***ihren Sorten***) und ***atomare Formeln*** von $\mathcal{L}_{\mathrm{sTLA}}(TSIG)$ sind die Terme und atomaren Formeln von $\mathcal{L}_{\mathrm{FOL}}(SIG^+)$ mit der zusätzlichen Termbildungsregel

- Ist $a \in \mathcal{X}^F$, so ist a' ein Term.

Induktive Definition der ***Transitionsausdrücke*** und Formeln:

1. Jede atomare Formel ist ein Transitionsausdruck, und jede atomare Formel ohne Terme der Form a' ($a \in \mathcal{X}^F$) ist eine Formel.
2. **false** ist ein Transitionsausdruck und eine Formel,
3. Sind A und B Transitionsausdrücke oder Formeln und ist $x \in \mathcal{X}$, so sind $(A \to B)$ und $\exists x A$ Transitionsausdrücke bzw. Formeln.
4. Ist $v \in \mathcal{V}^F$, so ist v' ein Transitionsausdruck.
5. Ist A ein Transitionsausdruck und $u \in \mathcal{X}^F \cup \mathcal{V}^F$, so ist $\Box[A]_u$ eine Formel.
6. Ist A eine Formel, so ist $\Box A$ eine Formel.

Abkürzungen, Schreibweisen, Vereinbarungen: wie bisher.
Zusätzliche Abkürzung: $\Box[A]_{u_1,\ldots,u_n}$ für $\Box[A]_{u_1} \wedge \ldots \wedge \Box[A]_{u_n}$.

Semantik von sTLA

Interpretationen für $\mathcal{L}_{\mathrm{sTLA}}$ sind gegeben durch temporale Strukturen wie bei FOLTL, und die Definition von $\mathsf{K}_i^{(\xi)}(\ldots)$ ist gegeben gemäß der „Einbettung“ von sTLA in FOLTL, z.B.:

- $\mathsf{K}_i^{(\xi)}(v') = \mathsf{K}_{i+1}^{(\xi)}(v) \quad (v \in \mathcal{V}^F)$,
- $\mathsf{K}_i^{(\xi)}(\Box[A]_u) = \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_j^{(\xi)}(A) = \mathsf{tt}$ oder $\mathsf{K}_{j+1}^{(\xi)}(u) = \mathsf{K}_j^{(\xi)}(u)$ für alle $j \geq i$.

Damit:

- $\mathsf{K}_i^{(\xi)}(\Box[A]_{u_1,\ldots,u_n}) = \mathsf{tt} \;\Leftrightarrow$ „In allen Zuständen η_j mit $j \geq i$ trifft A zu, oder $u_1, \ldots, u_n$ bleiben beim Übergang zu η_{j+1} unverändert“.

Die Gültigkeit $\models_{\mathsf{K}}$ („(initial) gültig in K“) ist definiert durch:

$$\models_{\mathsf{K}} A \;\Leftrightarrow\; \mathsf{K}_0^{(\xi)}(A) = \mathsf{tt} \text{ für alle Variablenbelegungen } \xi.$$

Spezifikationen in sTLA

Allgemeine Form von Systemspezifikationen in sTLA:

- STS Γ (mit Systemvariablen $u_1, \ldots, u_n$) und Γ_{impl} (Implementierung von Γ mit Systemvariablen $u_1, \ldots, u_n, u_{n+1}, \ldots, u_l$:

  $$\begin{aligned} \Gamma\text{:} \quad & F_\Gamma \equiv \Box[A_\Gamma]_{u_1,\ldots,u_n}, \\ \Gamma_{impl}\text{:} \quad & F_{\Gamma_{impl}} \equiv \Box[A_{\Gamma_{impl}}]_{u_1,\ldots,u_l}. \end{aligned}$$

- Analog für rSTS Γ und Γ_{impl}

  $$\begin{aligned} \Gamma\text{:} \quad & F_\Gamma \equiv start_\Gamma \wedge \Box[A_\Gamma]_{u_1,\ldots,u_n}, \\ \Gamma_{impl}\text{:} \quad & F_{\Gamma_{impl}} \equiv start_{\Gamma_{impl}} \wedge \Box[A_{\Gamma_{impl}}]_{u_1,\ldots,u_l}. \end{aligned}$$

- Korrektheit der Implementierung (in beiden Fällen):

  $$F_{\Gamma_{impl}} \models F_\Gamma.$$

  (Beachte: Dies ist hier gleichwertig mit $\models F_{\Gamma_{impl}} \rightarrow F_\Gamma$.)

7.2 Die Logik TLA

Verbergen von internen Systemvariablen

- Korrektheitsbeziehung zwischen einem System Γ und seiner Implementierung Γ_{impl} (siehe Abschnitt 7.1):
  - Jedes Modell der Spezifikation von Γ_{impl} ist ein Modell der Spezifikation von Γ.
- Weiterhin wünschenswert: Γ_{impl} sollte „das Verhalten von Γ nicht einschränken“, formaler ausgedrückt:
  - Jedes Modell der Spezifikation von Γ ist ein Modell der Spezifikation von Γ_{impl}, wenn man dessen zusätzliche Schritte und internen Systemvariablen „vernachlässigt“.

  Für die Spezifikationsaxiome F_Γ und $F_{\Gamma_{impl}}$ gemäß Abschnitt 7.1 würde das in sTLA bedeuten:

  $$F_\Gamma \vDash F_{\Gamma_{impl}} \qquad (\text{oder } \vDash F_\Gamma \rightarrow F_{\Gamma_{impl}}).$$

  (Für die bisherige Form von Spezifikationen in sTLA gilt dies nicht. Informell: „Γ_{impl} ist nicht die einzig mögliche Implementierung von Γ“.)
- Gesucht hierfür: Formel zur Beschreibung von (i.W.)

  „$F_{\Gamma_{impl}}$ mit ***verborgenen*** internen Variablen“.

TLA (Temporal Logic of Actions)

Eine Sprache $\mathcal{L}_{\mathrm{TLA}}(TSIG)$ (kurz: $\mathcal{L}_{\mathrm{TLA}}$) von TLA ist gegeben wie $\mathcal{L}_{\mathrm{sTLA}}(TSIG)$ mit der zusätzlichen Formelbildungsregel

7. Ist A eine Formel und $u \in \mathcal{X}^F \cup \mathcal{V}^F$, so ist $\exists^{st} uA$ eine Formel.

$\exists^{st}$ ist ein neuer Operator, ähnlich dem Operator $\exists$ von FOLTL, allerdings mit einigen wesentlichen Modifikationen.

Semantikdefinition für TLA: Wie in sTLA, zusätzlich zu definieren: $\mathsf{K}_i^{(\xi)}(\exists^{st} uA)$.
Ziel dabei: $\exists^{st} uA$ formalisiert „A mit verborgenem u und „robust“ unter Stotterschritten“.

Beispiel:

Spezifiziert man die Zähler Γ_{count} und $\Gamma_{implcount}$ (Abschnitt 7.1) durch

$$F_\Gamma \equiv \Box[A_{\Gamma_{count}}]_{ein,c},$$
$$F_{\Gamma_{impl}} \equiv \exists^{st} dez \Box[A_{\Gamma_{implcount}}]_{ein,c,dez},$$

so gilt

$$F_\Gamma \vDash F_{\Gamma_{impl}} \quad \text{und} \quad F_{\Gamma_{impl}} \vDash F_\Gamma \qquad (\vDash F_\Gamma \leftrightarrow F_{\Gamma_{impl}}).$$

Stotteräquivalenz

Sei $\mathsf{W} = (\eta_0, \eta_1, \eta_2, \ldots)$ eine unendliche Folge von Zuständen und $\natural_{\mathsf{W}} : \mathbb{N} \to \mathbb{N}$ induktiv definiert durch

$$\natural_{\mathsf{W}}(0) = 0,$$
$$\natural_{\mathsf{W}}(i+1) = \begin{cases} \natural_{\mathsf{W}}(i), & \text{falls } \eta_{i+1} = \eta_i \text{ und } \eta_j \neq \eta_i \text{ für ein } j > i \\ \natural_{\mathsf{W}}(i) + 1, & \text{sonst.} \end{cases}$$

Die Zustandsfolge $\natural(\mathsf{W}) = (\eta_{i_0}, \eta_{i_1}, \eta_{i_2}, \ldots)$ mit

$$i_0 = 0,$$
$$i_{k+1} = \text{kleinstes } j > i_k \text{ mit } \natural_{\mathsf{W}}(j) = \natural_{\mathsf{W}}(i_k) + 1 \quad (\text{für } k \in \mathbb{N})$$

heißt ***stotterfreie Variante*** von W.
(Informell: $\natural(\mathsf{W})$ ist „W ohne Stotterschritte“.)

Definition. Zwei unendliche Zustandsfolgen W_1 und W_2 heißen ***stotteräquivalent*** (geschrieben: $\mathsf{W}_1 \equiv_s \mathsf{W}_2$), falls $\natural(\mathsf{W}_1) = \natural(\mathsf{W}_2)$.

Semantik von $\exists^{st}$

Die Definition von $\mathsf{K}_i^{(\xi)}$ (für $\mathsf{K} = (\mathsf{S}, \mathsf{W})$ und $\xi : \mathcal{X} \to |\mathsf{S}|$) wird in TLA auf Formeln $\exists^{st} u A$ erweitert wie folgt:

$$\mathsf{K}_i^{\xi}(\exists^{st} u A) = \mathsf{tt} \;\Leftrightarrow\; \overline{\mathsf{K}_i}^{(\xi)}(A) = \mathsf{tt} \text{ für ein } \overline{\mathsf{K}} = (\mathsf{S}, \overline{\mathsf{W}}) \text{ mit } \overline{\mathsf{W}} \simeq_u \mathsf{W}.$$

Dabei:

$$\begin{aligned} \overline{W} \simeq_u \mathsf{W} \;\Leftrightarrow\; & \text{es gibt Zustandsfolgen} \\ & \overline{W}' = (\overline{\eta_0}', \overline{\eta_1}', \overline{\eta_2}', \ldots) \text{ und } \mathsf{W}' = (\eta_0', \eta_1', \eta_2', \ldots) \text{ mit} \\ & \quad \overline{\mathsf{W}} \equiv_s \overline{\mathsf{W}}', \\ & \quad \mathsf{W} \equiv_s \mathsf{W}', \\ & \quad \overline{\eta_i}'(\overline{u}) = \eta_i'(\overline{u}) \text{ für alle von } u \text{ verschiedenen } \overline{u} \in \mathcal{X}^F \cup \mathcal{V}^F \\ & \qquad\qquad \text{und alle } i \in \mathbb{N} \end{aligned}$$

Axiomatisierung von TLA

- Nicht ganz einfach (insbesondere, da Operator $\circ$ nicht allgemein vorhanden).
- Erst seit kurzem „zufriedenstellend“ gelöst.
  (Logik GTLA von MERZ: Erweiterung von TLA mit mehr „Ausdrucksstärke“, ohne die wesentlichen Eigenschaften von TLA bezüglich strukturierten Spezifikationen zu verlieren.)

Spezifikation und Verifikation in TLA

Allgemeine Form von Spezifikationen in TLA:

$$\exists^{st} u_1 \ldots \exists^{st} u_m (start_\Gamma \wedge \Box[N]_{u_1,\ldots,u_m,u_{m+1},\ldots,u_n} \wedge fair_\Gamma)$$

(für rSTS Γ; im Fall von STS entfällt $start_\Gamma$) mit

$u_1, \ldots, u_n$:	Systemvariablen von Γ,
$u_1, \ldots, u_m$:	interne Systemvariablen von Γ,
N:	Transitionsausdruck, der alle möglichen Zustandsänderungen beschreibt
$fair_\Gamma$:	Formel, die gewisse Fairness-Bedingungen beschreibt.

Beispiel:

„Unzuverlässige“ Warteschlange

Informelle Spezifikation:

- Schnittstelle des Systems: Systemvariablen in, out für Ein-/Ausgabe.
- Interne Systemvariable q: Zur Speicherung der Objekte in der Warteschlange
- Eingaben können verloren gehen oder vervielfacht werden.
- Ansonsten: Ein-/Ausgabe in FIFO-Weise.
- Zu Beginn: q leer.

Formale Spezifikation (passende Signatur vorausgesetzt; ohne Fairness):

$$\exists^{st} q(start \wedge \Box[eing \vee ausg]_{out,q})$$

mit

$$\begin{aligned} start &\equiv q = EMPTY, \\ eing &\equiv q' = APPEND(in, q) \wedge in' = in \wedge out' = out, \\ ausg &\equiv q \neq EMPTY \wedge out' = HEAD(q) \wedge q' = TAIL(q) \wedge in' = in. \end{aligned}$$

Allgemeine Form der Verifikation von Systemeigenschaften:

- Beweise $\models F_\Gamma \rightarrow A$.

(F_Γ Spezifikationsformel, A beschreibt Systemeigenschaft.)
Beachte: Darunter fallen auch die Aussagen

$\models F_{\Gamma_{impl}} \rightarrow F_\Gamma$ (Abschnitt 7.1) und
$\models F_\Gamma \rightarrow F_{\Gamma_{impl}}$ (s.o.).

Deduktive Verifikation: Herleitung solcher Aussagen in einem formalen System.

7.3 Verzweigte temporale Logiken

Übersicht

Grobeinteilung verschiedener Temporallogiken:

1. Lineare temporale Logiken
   - (FO)LTL$_{(0)}$ (mit eventuellen Erweiterungen)
   - TLA
   - Weitere Varianten
2. Verzweigte temporale Logiken
   - BTL
   - CTL
   - CTL* (Erweiterung von CTL)
3. Intervall-Logiken
   - Grundidee: „Auswertung“ von Formeln in Zustands-„Intervallen“

Verzweigte Zeitstrukturen

Grundkonzept für verzweigte temporale Logiken:

- ***Verzweigte Zeit***: Jeder „Zeitpunkt“ kann mehr als einen Nachfolge-„Zeitpunkt“ haben. Dies wird formalisiert wie folgt:

  Eine ***verzweigte Zeitstruktur*** $(I, \rightarrowtriangle)$ ist gegeben durch eine nicht-leere Menge I („Zeitpunkte“) und eine totale binäre („Nachfolge-“) Relation $\rightarrowtriangle$ auf I.

  Ein Pfad in I ist eine unendliche Folge $(\iota_0, \iota_1, \iota_2, \ldots)$ von Elementen von I mit $\iota_k \rightarrowtriangle \iota_{k+1}$ für $k \in \mathbb{N}$.

- Neuartige temporale Operatoren mit Bezug auf eine verzweigte Zeitstruktur.

  Basisoperatoren:

  $\exists\bigcirc A$: „Es gibt einen Nachfolgezeitpunkt, in dem A zutrifft“;

  $\exists\Box A$: „Es gibt einen Pfad (ausgehend vom jetzigen Zeitpunkt), auf dem A in allen (nachfolgenden) Zeitpunkten zutrifft“;

  $\exists\Diamond A$: „Es gibt einen Pfad (ausgehend vom jetzigen Zeitpunkt), auf dem A in (mindestens) einem (nachfolgenden) Zeitpunkt zutrifft“.

Die Basissprache

Sei $\mathcal{V}$ eine (endliche oder abzählbar unendliche) Menge von ***atomaren Aussagen***. Eine (Basis-) Sprache $\mathcal{L}_{\mathrm{BTL}}(\mathcal{V})$ (kurz: $\mathcal{L}_{\mathrm{BTL}}$) der ***verzweigten temporalen Aussagenlogik (BTL)*** ist definiert wie folgt:

Alphabet:

- Alle Elemente von $\mathcal{V}$,
- die Zeichen **false** $\mid \rightarrow \mid \exists\bigcirc \mid \exists\Box \mid \exists\Diamond \mid (\mid)$.

Induktive Definition der Formeln:

1. Jede atomare Aussage $v \in \mathcal{V}$ ist eine Formel.
2. **false** ist eine Formel.
3. Sind A und B Formeln, so ist auch $(A \rightarrow B)$ eine Formel.
4. Ist A eine Formel, so sind auch $\exists\bigcirc A$, $\exists\Box A$ und $\exists\Diamond A$ Formeln.

Abkürzungen: $\neg, \vee, \wedge, \leftrightarrow,$ **true** wie in PL, außerdem:

$\forall\bigcirc A$ für $\neg\exists\bigcirc\neg A$ („A trifft in allen Nachfolge-Zeitpunkten zu“),
$\forall\Box A$ für $\neg\exists\Diamond\neg A$ („A trifft auf allen Pfaden in allen Zeitpunkten zu“),
$\forall\Diamond A$ für $\neg\exists\Box\neg A$ („A trifft auf allen Pfaden in irgendeinem Zeitpunkt zu“).

Klammergebrauch: Analog wie in LTL.

Semantik

Interpretationen für $\mathcal{L}_{\mathrm{BTL}}$ sind gegeben durch ***(verzweigte) temporale Strukturen***.

Eine (verzweigte) temporale Struktur $\mathsf{K} = (\{\eta_\iota\}_{\iota\in I}, \rightarrowtriangle)$ für eine Menge $\mathcal{V}$ von atomaren Aussagen ist gegeben durch eine verzweigte Zeitstruktur $(I, \rightarrowtriangle)$ und eine Multimenge $\{\eta_\iota\}_{\iota\in I}$ von Abbildungen

$$\eta_\iota : \mathcal{V} \rightarrow \{\mathsf{ff}, \mathsf{tt}\}.$$

Die η_ι heißen ***Zustände***.

Für eine temporale Struktur $\mathsf{K} = (\{\eta_\iota\}_{\iota\in I}, \rightarrowtriangle)$ wird $\mathsf{K}_\iota(F) \in \{\mathsf{ff}, \mathsf{tt}\}$ für jede Formel F von $\mathcal{L}_{\mathrm{BTL}}$ und jedes $\iota \in I$ („Wahrheitswert von F im Zustand η_ι“) induktiv definiert wie folgt:

1. $\mathsf{K}_\iota(v) = \eta_\iota(v)$ für $v \in \mathcal{V}$.
2. $\mathsf{K}_\iota(\textbf{false}) = \mathsf{ff}$.
3. $\mathsf{K}_\iota(A \rightarrow B) = \mathsf{tt} \Leftrightarrow \mathsf{K}_\iota(A) = \mathsf{ff}$ oder $\mathsf{K}_\iota(B) = \mathsf{tt}$.
4. $\mathsf{K}_\iota(\exists\bigcirc A) = \mathsf{tt} \Leftrightarrow \mathsf{K}_\kappa(A) = \mathsf{tt}$ für ein κ mit $\iota \rightarrowtriangle \kappa$.
5. $\mathsf{K}_\iota(\exists\Box A) = \mathsf{tt} \Leftrightarrow$ es gibt einen Pfad $(\iota_0, \iota_1, \iota_2, \ldots)$ in I mit $\iota_0 = \iota$ und $\mathsf{K}_{\iota_k}(A) = \mathsf{tt}$ für alle $k \in \mathbb{N}$.
6. $\mathsf{K}_\iota(\exists\Diamond A) = \mathsf{tt} \Leftrightarrow$ es gibt einen Pfad $(\iota_0, \iota_1, \iota_2, \ldots)$ in I mit $\iota_0 = \iota$ und $\mathsf{K}_{\iota_k}(A) = \mathsf{tt}$ für ein $k \in \mathbb{N}$.

Auf die weiteren temporalen Operationen übertragen sich diese Festlegungen gemäß deren intuitiver Bedeutung, z.B.:

- $\mathsf{K}_\iota(\forall\bigcirc A) = \mathsf{tt} \Leftrightarrow \mathsf{K}_\kappa(A) = \mathsf{tt}$ für alle κ mit $\iota \rightarrowtriangle \kappa$.

Die Gültigkeit $\models_{\mathsf{K}}$ („gültig in K“) ist definiert durch:

$$\models_{\mathsf{K}} A \quad \Leftrightarrow \quad \mathsf{K}_\iota(A) = \mathsf{tt} \text{ für alle } \iota \in I.$$

Beispiel:

Sei $A \equiv \exists\Box(v_1 \vee v_2) \wedge \forall\circ v_2$ und K gegeben mit

$$I = \{1,2,3\}, \quad \rightarrowtriangle = \{(1,2),(1,3),(2,1),(2,3),(3,3)\},$$

	η_1	η_2	η_3
v_1	tt	tt	ff
v_2	ff	tt	tt

Pfade in I: $(1,2,1,3,3,3,\ldots)$,
$(1,3,3,3,\ldots)$,
$(2,1,2,1,2,\ldots)$,
$(3,3,3,\ldots)$,
$\vdots$

Dann gilt:

$\mathsf{K}_1(v_1 \vee v_2) = \mathsf{K}_3(v_1 \vee v_2) = \mathsf{tt} \;\Rightarrow\; \mathsf{K}_1(\exists\Box(v_1 \vee v_2)) = \mathsf{tt}$ (mit Pfad $(1,3,3,3,\ldots)$),
$\mathsf{K}_2(v_2) = \mathsf{K}_3(v_2) = \mathsf{tt} \;\Rightarrow\; \mathsf{K}_1(\forall\circ v_2) = \mathsf{tt}$,
zusammen: $\mathsf{K}_1(A) = \mathsf{tt}$.

$\mathsf{K}_1(v_2) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_2(\forall\circ v_2) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_2(A) = \mathsf{ff}$.

$\mathsf{K}_3(A) = \mathsf{tt}$ (wie bei $\mathsf{K}_1(A)$ mit Pfad $(3,3,3,\ldots)$).

Insbesondere: A ist nicht gültig in K.

Axiomatisierung

Ein mögliches (korrektes und schwach vollständiges) formales System Σ_{BTL} ist gegeben durch:

Axiome:

- Alle aussagenlogisch gültigen Formeln (analog definiert wie in LTL),
- $\exists\circ$**true**,
- $\exists\circ(A \vee B) \leftrightarrow \exists\circ A \vee \exists\circ B$,
- $\exists\Box A \leftrightarrow A \wedge \exists\circ\exists\Box A$,
- $\exists\Diamond A \leftrightarrow A \vee \exists\circ\exists\Diamond A$.

Regeln:

- $A, A \rightarrow B \vdash B$,
- $A \rightarrow B \vdash \exists\circ A \rightarrow \exists\circ B$,
- $A \rightarrow B \wedge \exists\circ A \vdash A \rightarrow \exists\Box B$,
- $A \rightarrow \neg B \wedge \forall\circ(A \vee \neg\exists\Diamond B) \vdash A \rightarrow \neg\exists\Diamond B$.

7.4 Die Logik CTL

CTL als Erweiterung von BTL

- BTL kann wie LTL in verschiedener Weise erweitert werden.
- Weit verbreitete Erweiterung: CTL (***Computation Tree Logic***). CTL ist das „verzweigte“ Analogon zu LTL+b, d.h. es ergibt sich aus BTL durch Hinzunahme von binären Operatoren (unter $\exists$ und $\forall$).
- Als binärer Operator wird üblicherweise **until** gewählt, d.h. CTL erweitert BTL um den zusätzlichen Operator $\exists$**until**. Die intuitive Bedeutung einer Formel $A\ \exists$**until** B ist:
  - „Es gibt einen Pfad, auf dem B in einem nachfolgenden Zeitpunkt zutreffen wird, und A trifft auf diesem Pfad bis zu diesem Zeitpunkt zu“.

Die Spracherweiterung $\mathcal{L}_{\mathrm{CTL}}$

Die Sprache $\mathcal{L}_{\mathrm{BTL}}$ wird zur Sprache $\mathcal{L}_{\mathrm{CTL}}$ (genauer: $\mathcal{L}_{\mathrm{CTL}}(\mathcal{V})$) erweitert durch

- Erweiterung des Alphabets von $\mathcal{L}_{\mathrm{BTL}}$ um $\exists$**until**;
- Erweiterung der induktiven Formeldefinition um die Regel

  5. Sind A und B Formeln, so ist auch $(A\ \exists$**until** $B)$ eine Formel;

- Erweiterung der Semantikdefinition um
  - $\mathsf{K}_\iota(A\ \exists$**until** $B) = \mathsf{tt} \Leftrightarrow$ es gibt einen Pfad $(\iota_0, \iota_1, \iota_2, \ldots)$ in I mit $\iota_0 = \iota$ und $\mathsf{K}_{\iota_j}(B) = \mathsf{tt}$ für ein $j \in \mathbb{N}$ und $\mathsf{K}_{\iota_k}(A) = \mathsf{tt}$ für alle $k, 0 \leq k < j$.

Abkürzung:

$A\ \forall$**until** B für $\neg(\neg B\ \exists$**until** $(\neg A \wedge \neg B)) \wedge \neg\exists\Box\neg B$

(„Auf jedem Pfad irgendwann B und bis dahin A“).

Außerdem: Der Operator $\exists\Diamond$ kann durch $\exists$**until** ausgedrückt werden, wird deshalb in der Syntaxdefinition von $\mathcal{L}_{\mathrm{CTL}}$ üblicherweise weggelassen und auch als Abkürzung eingeführt ($\exists\Diamond A$ für **true** $\exists$**until** A).

Schreibweise häufig auch:

$\exists(A$ **until** $B)$ und $\forall(A$ **until** $B)$ statt $A\ \exists$**until** B bzw. $A\ \forall$**until** B.

Beispiel:

Sei $A \equiv v_1 \; \exists\textbf{until} \; (v_1 \wedge v_2)$ und K gegeben wie im Beispiel in Abschnitt 7.3.

Dann gilt:

$\mathsf{K}_2(v_1 \wedge v_2) = \mathsf{tt}, \mathsf{K}_1(v_1) = \mathsf{tt} \;\Rightarrow\; \mathsf{K}_1(A) = \mathsf{tt}$ (mit Pfad $(1, 2, \ldots)$),
$\mathsf{K}_2(v_1 \wedge v_2) = \mathsf{tt} \;\Rightarrow\; \mathsf{K}_2(A) = \mathsf{tt}$ (mit beliebigem Pfad $(2, \ldots)$),
$\mathsf{K}_3(v_1 \wedge v_2) = \mathsf{ff} \;\Rightarrow\; \mathsf{K}_3(A) = \mathsf{ff}$ (da nur der Pfad $(3, 3, 3, \ldots)$ mit 3 beginnt).

Axiomatisierung

Ein mögliches (korrektes und schwach vollständiges) formales System Σ_{CTL} ist gegeben durch:

Axiome:

- Alle aussagenlogisch gültigen Formeln (analog definiert wie in LTL),
- $\exists\bigcirc\textbf{true}$,
- $\exists\bigcirc(A \vee B) \leftrightarrow \exists\bigcirc A \vee \exists\bigcirc B$,
- $\exists\Box A \leftrightarrow A \wedge \exists\bigcirc\exists\Box A$,
- $A \; \exists\textbf{until} \; B \leftrightarrow B \vee (A \wedge \exists\bigcirc(A \; \exists\textbf{until} \; B))$.

Regeln:

- $A, A \rightarrow B \vdash B$,
- $A \rightarrow B \vdash \exists\bigcirc A \rightarrow \exists\bigcirc B$,
- $A \rightarrow B \wedge \exists\bigcirc A \vdash A \rightarrow \exists\Box B$,
- $A \rightarrow \neg C \wedge \forall\bigcirc(A \vee \neg(B \; \exists\textbf{until} \; C)) \vdash A \rightarrow \neg(B \; \exists\textbf{until} \; C)$.

Spezifikation und Verifikation in CTL

- In CTL können (propositionale) STS spezifiziert und verifiziert werden. Hauptsächlich wird CTL allerdings für eine andere Art von Verifikation (nicht „deduktiv“ wie bisher, siehe Kapitel 8) verwendet.
- Ein Ablauf eines STS entspricht in verzweigten temporalen Logiken einem Pfad in einer verzweigten temporalen Struktur. Damit sind auch Aussagen der Art

  > „Es gibt einen Ablauf, so dass . . . “

  behandelbar, z.B. (Türme von Hanoi, Abschnitt 5.2):

  > „Es gibt einen Ablauf, in dem irgendwann einmal der Turm wie gewünscht versetzt ist“.

- Beachte: In CTL ist z.B. nicht ausdrückbar:

  > „Es gibt einen Ablauf, in dem unendlich oft A zutrifft“

  Das naheliegende $\exists\Box\Diamond A$ ist keine Formel von CTL. Die Erweiterung CTL* von CTL ermöglicht solche Formelbildungen.

Kapitel 8
Verifikation endlicher Zustandssysteme

8.1 Endliche Zustandssysteme

Automatisierbare Verifikation

- Verifikationsprobleme für Zustandssysteme Γ sind „Beweisaufgaben“ innerhalb einer temporalen Logik, und (z.B.) durch den Einsatz formaler Systeme möglicherweise zumindest „halb-automatisch“ unterstützbar.
- Für Zustandssysteme, die als propositionale STS formalisiert sind, sind die zugrundeliegenden temporalen Logiken (LTL, CTL, ...) entscheidbar, Verifikationsprobleme also (voll) algorithmisch behandelbar (zumindest „grundsätzlich“).
- Für Zustandssysteme, die durch STS 1. Stufe formalisiert sind, gilt dies i. Allg. nicht. Ausnahme (in vielen praktischen Anwendungen gegeben): Systeme, in denen alle Trägermengen (insbesondere also die „Wertmengen“ der Systemvariablen) endlich sind. Diese können als propositionale Systeme kodiert und Systemeigenschaften dann in einer temporalen Aussagenlogik formuliert werden.
- Kodierung eines derartigen STS Γ:
  Für jeses $a \in X_\Gamma$ sei $\mathbb{D}_a = |\mathsf{S}_\Gamma|_s$ (s Sorte von a). Seien weiter (für alle $a \in X_\Gamma$)

  $$V_a = \{v_d^a \mid d \in \mathbb{D}_a\}$$

  paarweise disjunkte Mengen von Systemvariablen (nicht enthalten in V_Γ) und

  $$\bar{V}_\Gamma \;=\; V_\Gamma \cup \bigcup_{a \in X_\Gamma} V_a.$$

  Für jedes $\eta \in Z_\Gamma$ sei $\bar{\eta} : \bar{V}_\Gamma \to \{\mathsf{ff}, \mathsf{tt}\}$ gegeben durch $\bar{\eta}(v) = \eta(v)$ für $v \in V_\Gamma$ und

  $$\bar{\eta}(v_d^a) = \mathsf{tt} \;\Leftrightarrow\; \eta(a) = d \quad \text{für } v_d^a \in \bigcup_{a \in X_\Gamma} V_a.$$

  Ist $\bar{Z}_\Gamma$ die Menge aller solcher $\bar{\eta}$ und $\bar{T}_\Gamma = \{(\bar{\eta}, \bar{\eta}') \in \bar{Z}_\Gamma \times \bar{Z}_\Gamma \mid (\eta, \eta') \in T_\Gamma\}$, so kodiert das propositionale STS

  $$\bar{\Gamma} = (\bar{V}_\Gamma, \bar{Z}_\Gamma, \bar{T}_\Gamma)$$

  das System Γ (und $\bar{V}_\Gamma$ ist endlich, falls $V_\Gamma \cup X_\Gamma$ endlich).

  Beispiel für eine Eigenschaft von Γ: $\exists x \Box(a \neq x)$.
  Beschrieben in $\mathcal{L}_{\mathrm{T}\bar{\Gamma}}$ durch: $\bigvee_{d \in \mathbb{D}_a} \Box\neg v_d^a$.

Propositionale Systeme mit endlich vielen Systemvariablen

Gemäß obiger Diskussion: Untersuchung algorithmisch behandelbarer Verifikationsprobleme „normiert“ auf propositionale STS.

Jetzt noch zusätzliche Einschränkung (in der Anwendungspraxis irrelevant; vgl. bisherige Beispiele): System hat nur endlich viele Systemvariablen (und damit auch nur endlich viele Zustände und eine endliche Transitionsrelation).

Eigenständige Definition solcher Systeme:

Definition. Ein ***endliches Zustandssystem*** (kurz: FSS) $\Psi = (V, Z, T)$ ist gegeben durch:

- eine endliche Menge V von ***Systemvariablen*** (***Zustandsvariablen***),
- eine Menge Z von ***(System-) Zuständen*** $\eta : V \to \{\mathsf{ff}, \mathsf{tt}\}$,
- eine totale ***Transitionsrelation*** $T \subseteq Z \times Z$.

Ein ***Ablauf*** von Ψ ist eine unendliche Folge $\mathsf{W} = (\eta_0, \eta_1, \eta_2, \ldots)$ von Systemzuständen mit $(\eta_i, \eta_{i+1}) \in T$ für alle $i \in \mathbb{N}$.

Bezeichnungen, Schreibweisen: wie bei allgemeinen STS (z.B.: $\Psi(V)$, Z_Ψ, ...).

Beispiele:

1. Zähler modulo 3
   (wie Γ_{count} in Abschnitt 5.2, aber mit „Zählschritt“ $c \rightsquigarrow c \boxplus 1$;
   $\boxplus$: Addition modulo 3; mögliche Werte von c: $0, 1, 2$)

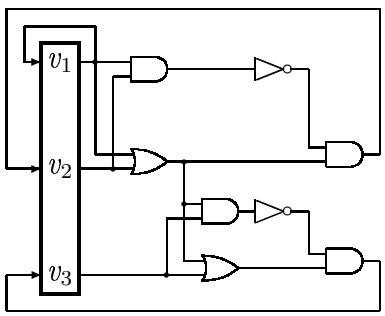
   Formalisiert (kodiert) als FSS $\Psi_{c3} = (V, Z, T)$ mit:

   $$V = \{ein, c_0, c_1, c_2\},$$
   $$Z = \{\eta : V \to \{\mathsf{ff}, \mathsf{tt}\} \mid \eta(c_i) = \mathsf{tt} \text{ für genau ein } i = 0, 1, 2\},$$
   (Notation für $\eta \in Z$: $[v \in V \text{ mit } \eta(v) = \mathsf{tt}]$.)
   $$T = \{([ein, c_i], [ein, c_{i\boxplus 1}]), ([ein, c_i], [c_i]), ([c_i], [c_i]), ([c_i], [ein, c_0]) \mid i \in \{0, 1, 2\}\}.$$

   (Ein möglicher) Ablauf von Ψ_{c3}:

   $$[c_2] \longrightarrow [ein, c_0] \longrightarrow [ein, c_1] \longrightarrow [ein, c_2] \longrightarrow [ein, c_0] \longrightarrow [ein, c_1] \longrightarrow [c_1] \longrightarrow [c_1] \longrightarrow \ldots$$

2. (Synchrones) Schaltwerk



Dargestellt als FSS $\Psi_{osz} = (V, Z, T)$ mit:

$$
\begin{aligned}
&V = \{v_1, v_2, v_3\},\\
&Z = \{\eta : V \to \{\mathsf{ff}, \mathsf{tt}\}\},\\
&T = \{(\eta, \eta') \in Z \times Z \mid \eta'(v_1) = \eta(v_1),\\
&\qquad\qquad \eta'(v_2) = \mathsf{tt} \;\Leftrightarrow\; \eta(v_1) \neq \eta(v_2),\\
&\qquad\qquad \eta'(v_3) = \mathsf{tt} \;\Leftrightarrow\; (\eta(v_3) = \mathsf{tt} \Leftrightarrow \eta(v_1) = \eta(v_2) = \mathsf{ff})\}
\end{aligned}
$$

(Ein möglicher) Ablauf von Ψ_{osz}:

$$011 \longrightarrow 101 \longrightarrow 011 \longrightarrow 101 \longrightarrow 011 \longrightarrow \ldots$$

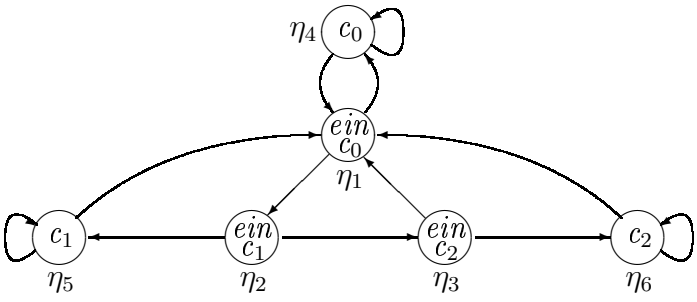
(η dargestellt durch das entsprechende Binärwort; 011 bedeutet z.B. $\eta(v_1) = \mathsf{tt}$, $\eta(v_2) = \mathsf{tt}$, $\eta(v_3) = \mathsf{ff}$).

Transitionsdiagramme

Graphische Darstellung von FSS (***Transitionsdiagramme***): Zustände η werden durch Knoten repräsentiert, die diejenigen Variablen v mit $\eta(v) = \mathsf{tt}$ „enthalten“. Pfeile zwischen den Knoten repräsentieren die Transitionsrelation.

Beispiel:

Transitionsgraph für Ψ_{c3} (s.o.):



8.2 Model Checking

Verifikationsmethoden

Zwei Ansätze zur Verifikation einer Eigenschaft(sformel) F für ein Zustandssystem Γ:

- Wie bisher (deduktiv):
  - Spezifiziere Γ durch Axiomenmenge $\mathcal{A}_\Gamma$,
  - Beweise $\mathcal{A}_\Gamma \vdash F$.

- ***Model Checking***:
  - Zeige durch „direkte“ Betrachtung aller Abläufe von Γ (ohne ihre Beschreibung durch $\mathcal{A}_\Gamma$), dass F „in Γ gilt“.
- Im Fall endlicher Zustandssysteme sind beide Verifikationsmethoden (grundsätzlich) algorithmisierbar, Model Checking hat sich dabei als besonders erfolgreich erwiesen.
- Model Checking im Fall eines endlichen Zustandssystems Ψ bedeutet
  - bei zugrundegelegter linearer temporaler Logik: Überprüfe, dass F in jeder temporalen Struktur W_Ψ gültig ist;
  - bei zugrundegelegter verzweigter temporaler Logik: Überprüfe, dass F in der durch Ψ bestimmten verzweigten temporalen Struktur K_ψ (s.u.) gültig ist.

  (Am weitesten verbreitet: Model Checking mit CTL.)

Verzweigte temporale Strukturen von endlichen Zustandssystemen

Ein FSS $\Psi = (V, Z, T)$ mit $Z = \{\eta_1, \ldots, \eta_m\}$ induziert eine verzweigte temporale Struktur $\mathsf{K}_\Psi = (\{\eta_\iota\}_{\iota \in I}, \rightarrowtriangle)$ für V mit

$$
\begin{array}{l}
I = \{1, \ldots, m\}, \\
\{\eta_\iota\}_{\iota \in I} = \{\eta_1, \ldots, \eta_m\} = Z, \\
\iota \rightarrowtriangle \kappa \;\Leftrightarrow\; (\eta_\iota, \eta_\kappa) \in T \quad \text{für } \iota, \kappa \in I.
\end{array}
$$

Beispiel:

Für Ψ_{c3} (Abschnitt 8.1) ergibt sich $\mathsf{K}_{\Psi_{c3}} = (\{\eta_1, \eta_2, \eta_3, \eta_4, \eta_5, \eta_6\}, \rightarrowtriangle)$ mit

$$
\begin{array}{rl}
\rightarrowtriangle \;=\; \{&(1,2),(1,4),(2,3),(2,5),(3,1),(3,6), \\
&(4,1),(4,4),(5,1),(5,5),(6,1),(6,6)\}.
\end{array}
$$

Schreibweise: $\mathsf{K}_\Psi^{(\eta_\iota)}$ statt $(\mathsf{K}_\Psi)_\iota$

CTL Model Checking

Für ein FSS $\Psi = (V, Z, T)$ sei $\mathcal{L}_{\mathrm{CTL}\Psi} = \mathcal{L}_{\mathrm{CTL}}(V)$.

Definition. Sei $\Psi = (V, Z, T)$ ein FSS und F eine Formel von $\mathcal{L}_{\mathrm{CTL}\Psi}$. Die ***Erfüllungsmenge*** $[\![F]\!]_\Psi$ (kurz: $[\![F]\!]$) von F (in Ψ) ist die Menge

$$[\![F]\!]_\Psi = \{\eta \in Z \mid \mathsf{K}_\Psi^{(\eta)}(F) = \mathsf{tt}\}.$$

Offenbar gilt für jedes FSS Ψ und jede Formel (Eigenschaft) F von $\mathcal{L}_{\mathrm{CTL}\Psi}$:

- F ist gültig in $\mathsf{K}_\Psi \;\Leftrightarrow\; [\![F]\!]_\Psi$ enthält alle Zustände von Ψ.

Grundprinzip des Model Checking mit zugrundeliegendem CTL (***CTL Model Checking***):

- Gegeben ein FSS Ψ und eine Formel F von $\mathcal{L}_{\mathrm{CTL}\Psi}$;
  bestimme $[\![F]\!]_\Psi$.

8.3 Grundlagen des CTL Model Checking

Berechnung von Erfüllungsmengen

Die Erfüllungsmenge $[\![F]\!]$ für ein FSS $\Psi = (V, Z, T)$ und eine Formel F von $\mathcal{L}_{\mathrm{CTL}\Psi}$ kann induktiv wie folgt konstruiert werden:

$[\![v]\!] = \{\eta \in Z \mid \eta(v) = \mathsf{tt}\}$ für $v \in V$.

$[\![\mathbf{false}]\!] = \emptyset$.

$[\![A \rightarrow B]\!] = (Z \setminus [\![A]\!]) \cup [\![B]\!]$.

(Außerdem: $[\![\neg A]\!] = Z \setminus [\![A]\!]$.

$[\![A \vee B]\!] = [\![A]\!] \cup [\![B]\!]$.

$[\![A \wedge B]\!] = [\![A]\!] \cap [\![B]\!]$.

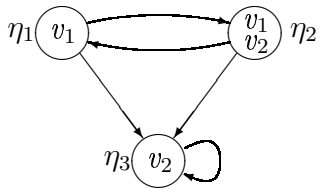
$[\![\mathbf{true}]\!] = Z$.)

$[\![\exists\bigcirc A]\!] = \{\eta \in Z \mid$ es gibt $\eta' \in Z$ mit $(\eta, \eta') \in T$ und $\eta' \in [\![A]\!]\}$.

$[\![\exists\Box A]\!]$ und $[\![A\ \exists\mathbf{until}\ B]\!]$: siehe unten.

Beispiel:

Sei Ψ_{bsp} das durch folgendes Transitionsdiagramm gegebene FSS (vgl. Beispiel in 7.3):



Dann ergibt sich:

$[\![v_1]\!] = \{\eta_1, \eta_2\}$,
$[\![v_2]\!] = \{\eta_2, \eta_3\}$,
$[\![\neg v_2]\!] = Z \setminus \{\eta_2, \eta_3\} = \{\eta_1\}$,
$[\![v_1 \vee v_2]\!] = [\![v_1]\!] \cup [\![v_2]\!] = \{\eta_1, \eta_2\} \cup \{\eta_2, \eta_3\} = Z$,
$[\![v_1 \wedge v_2]\!] = [\![v_1]\!] \cap [\![v_2]\!] = \{\eta_1, \eta_2\} \cap \{\eta_2, \eta_3\} = \{\eta_2\}$,
$[\![\exists\bigcirc\neg v_2]\!] = \{\eta \in Z \mid (\eta, \eta_1) \in T\} = \{\eta_2\}$,
$[\![\forall\bigcirc v_2]\!] = [\![\neg\exists\bigcirc\neg v_2]\!] = Z \setminus \{\eta_2\} = \{\eta_1, \eta_3\}$.

Einige Fixpunktsätze

Die noch fehlenden Bestimmungen von $[\![\exists\Box A]\!]$ und $[\![A\ \exists\mathbf{until}\ B]\!]$ benötigen einige Vorbereitungen.

Sei $\Psi = (V, Z, T)$ ein FSS, $\mathcal{P}(Z)$ die Potenzmenge von Z und $\mathcal{P} \subseteq \mathcal{P}(Z)$ mit $\emptyset \in \mathcal{P}$ und $Z \in \mathcal{P}$.

Definition. Eine Abbildung $\pi : \mathcal{P} \to \mathcal{P}$ heißt ***monoton***, wenn für alle $P, Q \in \mathcal{P}$ gilt:

$$P \subseteq Q \;\Rightarrow\; \pi(P) \subseteq \pi(Q).$$

$P \in \mathcal{P}$ heißt ***Fixpunkt*** von π, wenn gilt: $\pi(P) = P$.

Lemma 8.3.1 Ist $\pi : \mathcal{P} \to \mathcal{P}$ monoton, so gilt für alle $i \in \mathbb{N}$:

a) $\pi^i(\emptyset) \subseteq \pi^{i+1}(\emptyset)$.

b) $\pi^i(Z) \supseteq \pi^{i+1}(Z)$.

Lemma 8.3.2 Sei $\pi : \mathcal{P} \to \mathcal{P}$ monoton, $LFP(\pi) = \bigcup_{i \in \mathbb{N}} \pi^i(\emptyset)$, $GFP(\pi) = \bigcap_{i \in \mathbb{N}} \pi^i(Z)$. Dann gilt: $LFP(\pi) \in \mathcal{P}$, $GFP(\pi) \in \mathcal{P}$, und $LFP(\pi)$ und $GFP(\pi)$ sind Fixpunkte von π.

Sei $\Psi = (V, Z, T)$ ein FSS und $\mathcal{P}_\Psi = \{[\![F]\!] \mid F \text{ Formel von } \mathcal{L}_{\mathrm{CTL}\Psi}\}$ die Menge aller Erfüllungsmengen von Formeln von $\mathcal{L}_{\mathrm{CTL}\Psi}$. (Dann gilt: $\mathcal{P}_\Psi \subseteq \mathcal{P}(Z)$, $\emptyset = [\![\mathbf{false}]\!] \in \mathcal{P}_\Psi$, $Z = [\![\mathbf{true}]\!] \in \mathcal{P}_\Psi$.) A und B seien Formeln von $\mathcal{L}_{\mathrm{CTL}\Psi}$.
Die Abbildungen $\pi_1, \pi_2 : \mathcal{P}_\Psi \to \mathcal{P}_\Psi$ seien definiert durch:

$$\pi_1([\![F]\!]) = [\![A \wedge \exists\bigcirc F]\!],$$
$$\pi_2([\![F]\!]) = [\![B \vee (A \wedge \exists\bigcirc F)]\!].$$

(Leicht zu zeigen: π_1 und π_2 sind wohldefiniert.)

Lemma 8.3.3 π_1 und π_2 sind monoton.

Satz 8.3.4 $[\![\exists\Box A]\!] = GFP(\pi_1)$.

Satz 8.3.5 $[\![A \;\exists\mathbf{until}\; B]\!] = LFP(\pi_2)$.

Die Fixpunktkonstruktionen

$[\![\exists\Box A]\!]$ und $[\![A \;\exists\mathbf{until}\; B]\!]$ können gemäß den obigen Ergebnissen wie folgt konstruiert werden.

- Konstruktion von $[\![\exists\Box A]\!]$:
  Die Formeln $C_0, C_1, C_2, \ldots$ seien gegeben durch

  $$C_0 \equiv \mathbf{true},$$
  $$C_{i+1} \equiv A \wedge \exists\bigcirc C_i \quad \text{für } i = 0, 1, 2, \ldots$$

  Konstruiere $[\![C_i]\!]$ für $i = 0, 1, 2, \ldots$ so lange, bis $[\![C_{k+1}]\!] = [\![C_k]\!]$ für ein k.
  Dann ist $[\![\exists\Box A]\!] = [\![C_k]\!]$.

- Konstruktion von $[\![A\ \exists\textbf{until}\ B]\!]$:
  Die Formeln $D_0, D_1, D_2, \ldots$ seien gegeben durch

  $$D_0 \equiv \textbf{false},$$
  $$D_{i+1} \equiv B \vee (A \wedge \exists\bigcirc D_i) \quad \text{für } i = 0, 1, 2, \ldots$$

  Konstruiere $[\![D_i]\!]$ für $i = 0, 1, 2, \ldots$ so lange, bis $[\![D_{k+1}]\!] = [\![D_k]\!]$ für ein k.
  Dann ist $[\![A\ \exists\textbf{until}\ B]\!] = [\![D_k]\!]$.

Beispiel:

Für Ψ_{bsp} von oben:

- Berechnung von $[\![\exists\Box(v_1 \vee v_2)]\!]$

  $$\begin{aligned}
  [\![C_0]\!] &= [\![\textbf{true}]\!] = Z,\\
  [\![C_1]\!] &= [\![(v_1 \vee v_2) \wedge \exists\bigcirc\textbf{true}]\!]\\
  &= [\![v_1 \vee v_2]\!] \cap \{\eta \in Z \mid \text{es gibt } \eta' \in Z \text{ mit } (\eta, \eta') \in T\}\\
  &= Z \cap Z\\
  &= Z\\
  &= [\![C_0]\!].
  \end{aligned}$$

  Also: $[\![\exists\Box(v_1 \vee v_2)]\!] = Z$.

- Berechnung von $[\![v_1\ \exists\textbf{until}\ (v_1 \wedge v_2)]\!]$

  $$\begin{aligned}
  [\![D_0]\!] &= [\![\textbf{false}]\!] = \emptyset,\\
  [\![D_1]\!] &= [\![(v_1 \wedge v_2) \vee (v_1 \wedge \exists\bigcirc\textbf{false})]\!]\\
  &= [\![v_1 \wedge v_2]\!] \cup ([\![v_1]\!] \cap \{\eta \in Z \mid \text{es gibt } \eta' \in Z \text{ mit}\\
  &\qquad\qquad (\eta, \eta') \in T \text{ und } \eta' \in \emptyset\})\\
  &= \{\eta_2\} \cup (\{\eta_1, \eta_2\} \cap \emptyset)\\
  &= \{\eta_2\},\\
  [\![D_2]\!] &= [\![(v_1 \wedge v_2) \vee (v_1 \wedge \exists\bigcirc D_1)]\!]\\
  &= [\![v_1 \wedge v_2]\!] \cup ([\![v_1]\!] \cap \{\eta \in Z \mid \text{es gibt } \eta' \in Z \text{ mit}\\
  &\qquad\qquad (\eta, \eta') \in T \text{ und } \eta' \in \{\eta_2\}\})\\
  &= \{\eta_2\} \cup (\{\eta_1, \eta_2\} \cap \{\eta_1\})\\
  &= \{\eta_1, \eta_2\},\\
  [\![D_3]\!] &= [\![(v_1 \wedge v_2) \vee (v_1 \wedge \exists\bigcirc D_2)]\!]\\
  &= [\![v_1 \wedge v_2]\!] \cup ([\![v_1]\!] \cap \{\eta \in Z \mid \text{es gibt } \eta' \in Z \text{ mit}\\
  &\qquad\qquad (\eta, \eta') \in T \text{ und } \eta' \in \{\eta_1, \eta_2\}\})\\
  &= \{\eta_2\} \cup (\{\eta_1, \eta_2\} \cap \{\eta_1, \eta_2\})\\
  &= \{\eta_1, \eta_2\}\\
  &= [\![D_2]\!].
  \end{aligned}$$

  Also: $[\![v_1\ \exists\textbf{until}\ (v_1 \wedge v_2)]\!] = \{\eta_1, \eta_2\}$.

Bemerkungen

- Ist die untersuchte Formel F von der Form $\exists \ldots$, so kann, falls $[\![F]\!]_\Psi = Z_\Psi$, auch leicht ein „erfüllender“ Ablauf konstruiert werden. Ist $F \equiv \forall \ldots$ und $[\![F]\!]_\Psi \neq Z_\Psi$, so kann ein „Gegenbeispiel“ konstruiert werden.
- Model Checking wird industriell angewendet.
- Model Checking ist ein großes aktuelles Forschungsthema. Stichpunkte (unter vielen anderen):
  - Berücksichtigung von Fairness.
  - Ausweitung auf gewisse „nicht-endliche“ Systeme.
  - Integration in den Entwicklungsprozess komplexer Software.
  - Behandlung von Systemen mit Realzeit-Bedingungen.
  - Entwicklung von Werkzeugen.

8.4 Binäre Entscheidungsdiagramme

Nutzbarkeit von Model Checking

- Wesentliche Voraussetzung für die industrielle Nutzbarkeit von Model Checking: Effiziente Implementierbarkeit der Berechnung von Erfüllungsmengen.

  Dazu nötig: Effiziente Realisierungen von
  - Mengen von Zuständen und Zustandspaaren,
  - Operationen auf diesen Mengen

  gemäß den Konstruktionsregeln in Abschnitt 8.3.
- „Historischer“ Durchbruch hierfür: „Symbolische“ Darstellung von Zustandsmengen durch ***(geordnete) binäre Entscheidungsdiagramme***.

Binäre Entscheidungsdiagramme

Für die folgenden Betrachtungen sei $V = \{w_1, \ldots, w_m\}$ jeweils eine endliche Menge von Systemvariablen, $Z(V)$ die Menge aller Zustände $\eta : V \to \{\mathsf{ff}, \mathsf{tt}\}$.

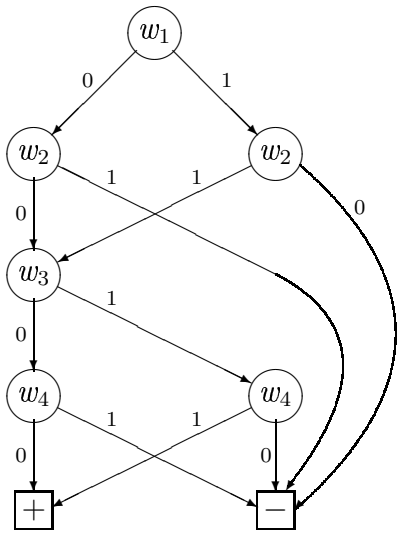
Ein binäres Entscheidungsdiagramm (***BDD***) (über V) ist ein gerichteter azyklischer Wurzelgraph mit den Eigenschaften:

- Jeder Nicht-Endknoten k ist mit $var(k) \in V$ markiert und hat zwei Nachfolgeknoten: $null(k)$ und $eins(k)$.
- Jeder Endknoten k ist mit $erg(k) \in \{+, -\}$ markiert.

Ein BDD D über V heißt geordnet (***OBDD***), wenn gilt: Es gibt eine totale Ordnung $<$ auf V, und für jeden Knoten k von D mit Nicht-Endknoten k' als Nachfolgeknoten gilt $var(k) < var(k')$.

Beispiel:

Ein OBDD D_{bsp} über $\{w_1, w_2, w_3, w_4\}$:



(Mit 0 markierte Kanten führen von einem Knoten k zu $null(k)$, analog für Markierung 1.)

Darstellung von Zustandsmengen durch OBDD

Für jeden Knoten k in einem BDD über V ist die Menge $Z_k \subseteq Z(V)$ induktiv definiert wie folgt:

1. Ist k Endknoten, so ist $Z_k = \begin{cases} Z(V), & \text{falls } erg(k) = +, \\ \emptyset, & \text{falls } erg(k) = -. \end{cases}$
2. Ist k nicht Endknoten und $var(k) = w_i$ $(1 \leq i \leq m)$, so ist

$$Z_k = (\{\eta \in Z(V) \mid \eta(w_i) = \mathsf{ff}\} \cap Z_{null(k)}) \cup (\{\eta \in Z(V) \mid \eta(w_i) = \mathsf{tt}\} \cap Z_{eins(k)}).$$

Ein BDD D über V mit Wurzel k definiert die Zustandsmenge $Z_D = Z_k \subseteq Z(V)$.

Es gilt: Zu vorgegebener Ordnung auf V kann aus einem OBDD D über V ein eindeutig bestimmter „minimaler“ OBDD D' über V mit $Z_{D'} = Z_D$ konstruiert werden.

Für eine Menge $M \subseteq Z(V)$ (und vorgegebene Ordnung auf V) bezeichne $D(M)$ das minimale OBDD mit $Z_{D(M)} = M$. ($D(M)$ „stellt M dar“.)

Beispiel:

Das OBDD D_{bsp} von oben stellt die Menge

$$\{\eta \in Z(V) \mid \eta(w_1) = \eta(w_2) \text{ und } \eta(w_3) = \eta(w_4)\}$$

dar (mit $V = \{w_1, w_2, w_3, w_4\}$).

Nachbildung von Mengenoperationen auf OBDD

Für $\eta \in Z(V)$, $w \in V$, $b \in \{\mathsf{ff}, \mathsf{tt}\}$ sei der Zustand $\eta_{w \mapsto b} \in Z(V)$ definiert durch:

$$\eta_{w \mapsto b}(v) = \begin{cases} \eta(v) & \text{für } v \neq w \\ b & \text{für } v = w. \end{cases}$$

Seien $M, M' \subseteq Z(V)$. Durch die folgenden Konstruktionen auf OBDD werden Mengenoperationen auf M, M' nachgebildet.

- Konstruktion von $D(M_{w \mapsto b})$ aus $D(M)$ $\quad (M_{w \mapsto b} = \{\eta \in Z(V) \mid \eta_{w \mapsto b} \in M\})$:
  - Entferne in $D(M)$ jede Kante von einem Knoten k mit $var(k) = w$ zu $null(k)$, falls $b = \mathsf{tt}$, bzw. zu $eins(k)$, falls $b = \mathsf{ff}$. Entferne alle Knoten, die danach von der Wurzel aus nicht mehr erreichbar sind.
  - Entferne alle Knoten k mit $var(k) = w$, und ersetze alle Kanten zu k durch Kanten zu $null(k)$, falls $b = \mathsf{ff}$, bzw. zu $eins(k)$, falls $b = \mathsf{tt}$.
  - Minimalisiere (falls notwendig).
- Konstruktion von $D(\overline{M})$ aus $D(M)$ $\quad (\overline{M} = \{\eta \in Z(V) \mid \eta \notin M\})$:
  - Ersetze in allen Endknoten von $D(M)$ $+$ durch $-$ und umgekehrt.
- Konstruktion von $D(M \circ M')$ aus $D(M)$ und $D(M')$ $\quad (\circ \in \{\cap, \cup\})$:

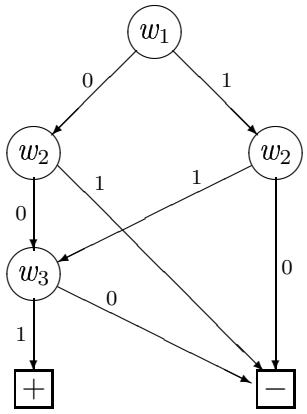
  Seien k Wurzel von $D(M)$, k' Wurzel von $D(M')$.
  - Falls k und k' Endknoten, so besteht auch $D(M \circ M')$ nur aus einem Endknoten k'' mit
    $$erg(k'') = + \;\Leftrightarrow\; \begin{cases} erg(k) = + \text{ und } erg(k') = +, & \text{falls } \circ = \cap \\ erg(k) = + \text{ oder } erg(k') = +, & \text{falls } \circ = \cup. \end{cases}$$
  - Falls k und k' nicht Endknoten mit $var(k) \leq var(k')$ oder falls k nicht Endknoten und k' Endknoten, $var(k) = w$, so konstruiere $D_1 = D(M_{w \mapsto \mathsf{ff}} \circ M'_{w \mapsto \mathsf{ff}})$ und $D_2 = D(M_{w \mapsto \mathsf{tt}} \circ M'_{w \mapsto \mathsf{tt}})$. $D(M \circ M')$ besteht dann aus k als Wurzel sowie D_1 und D_2, so dass $null(k)$ die Wurzel von D_1 und $eins(k)$ die Wurzel von D_2 sind.
  - Wie voriger Fall, k und k' vertauscht: Verfahre analog.
  - Minimalisiere (falls notwendig).

Beispiel:

Für die durch das OBDD D_{bsp} (s.o.) dargestellte Menge M gilt

$$M_{w_4 \mapsto \mathsf{tt}} = \{\eta \in Z(V) \mid \eta(w_1) = \eta(w_2) \text{ und } \eta(w_3) = \mathsf{tt}\}.$$

Die entsprechende Konstruktion, angewendet auf $D(M) = D_{bsp}$, liefert:



Darstellung von Mengen von Zustandspaaren durch OBDD

Sei $V' = \{w' \mid w \in V\}$. Für $\eta_1, \eta_2 \in Z(V)$ sei $\eta_1 \times \eta_2 \in Z(V \cup V')$ definiert durch:

$$\eta_1 \times \eta_2(w) = \begin{cases} \eta_1(w), & \text{falls } w \in V \\ \eta_2(v), & \text{falls } w = v' \in V'. \end{cases}$$

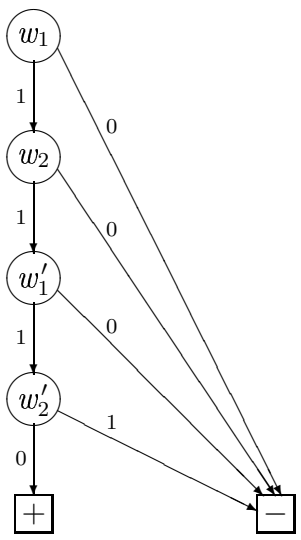
Für eine Paarmenge $M \subseteq Z(V) \times Z(V)$ sei

$$M^{\times} = \{\eta_1 \times \eta_2 \in Z(V \cup V') \mid (\eta_1, \eta_2) \in M\}.$$

$D(M^{\times})$ ist die OBDD-Darstellung von M (als OBDD über $V \cup V'$).

Beispiel:

Sei $V = \{w_1, w_2\}$, $M = \{(\eta_1, \eta_2)\}$ mit $\eta_1(w_1) = \eta_1(w_2) = \mathsf{tt}$, $\eta_2(w_1) = \mathsf{tt}$, $\eta_2(w_2) = \mathsf{ff}$.
OBDD $D(M^{\times})$ zur Darstellung von M:



Der Algorithmus SMC

Der grundlegende Algorithmus SMC (***symbolic model checking***) arbeitet gemäß den vorangegangenen Betrachtungen und realisiert die in Abschnitt 8.3 angegebenen Berechnungen in effizienter Weise.

Für ein FSS $\Psi = (V, Z, T)$ und eine Formel F von $\mathcal{L}_{\mathrm{CTL}\Psi}$ sei

$$M_F = \{\eta \in Z(V) \mid \mathsf{K}_\Psi^{(\eta)}(F) = \mathsf{tt}\}.$$

(Dann ist $[\![F]\!]_\Psi = M_F \cap Z$.) $D(F)$ bezeichne das OBDD $D(M_F)$.

- Eingabe für SMC:
  - $D(T^\times)$ als Darstellung der Transitionsrelation T eines FSS $\Psi = (V, Z, T)$, $V = \{v_1, \ldots, v_n\}$,
  - Formel F von $\mathcal{L}_{\mathrm{CTL}\Psi}$.
- Ausgabe von SMC: $D(F)$.
- Arbeitsweise von SMC (gemäß dem induktiven Aufbau von F):
  1. Falls $F \equiv v \in V$ oder $F \equiv$ **false** (oder $F \equiv$ **true**), so konstruiere (das triviale) $D(F)$ direkt.
  2. Falls $F \equiv A \to B$ (oder $F \equiv \neg A$, $F \equiv A \vee B$, $F \equiv A \wedge B$), so konstruiere $D(F)$ gemäß den Definitionen in Abschnitt 8.3 (und den Nachbildungen der entsprechenden Mengenoperationen).
  3. Falls $F \equiv \exists\bigcirc A$, so:
     - Konstruiere $D(A)$, und ersetze darin alle Knotenmarkierungen $v \in V$ durch v'. (Das so veränderte OBDD sei mit $D'(A)$ bezeichnet.)
     - Sei $M^{(0)} = T^\times \cap Z_{D'(A)}$,
       $M^{(1)} = M^{(0)}_{v_1' \mapsto \mathsf{ff}} \cup M^{(0)}_{v_1' \mapsto \mathsf{tt}}$,
       $M^{(2)} = M^{(1)}_{v_2' \mapsto \mathsf{ff}} \cup M^{(1)}_{v_2' \mapsto \mathsf{tt}}$,
       $\vdots$
       $M^{(n)} = M^{(n-1)}_{v_n' \mapsto \mathsf{ff}} \cup M^{(n-1)}_{v_n' \mapsto \mathsf{tt}}$.

       Konstruiere sukzessive $D(M^{(0)}), D(M^{(1)}), D(M^{(2)}), \ldots$ (mit den Nachbildungen der jeweiligen Mengenoperationen). Dann ist $D(F) = D(M^{(n)})$.
  4. Falls $F \equiv \exists\Box A$ oder $F \equiv A\ \exists$**until** B, so konstruiere $D(F)$ gemäß den Fixpunktkonstruktionen in Abschnitt 8.3 (und den Nachbildungen der entsprechenden Mengenoperationen).