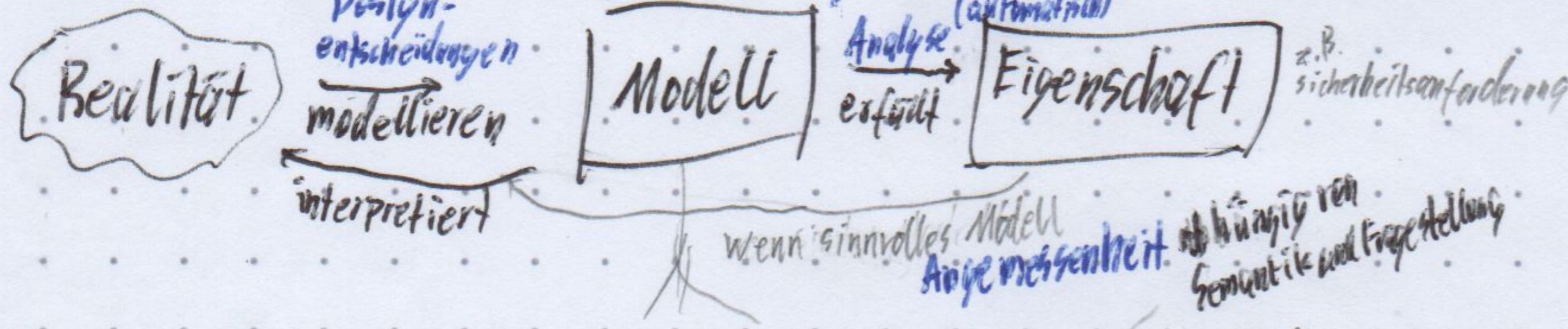






Moses



Techniken: schrittweise: verfeinerte & komponierte

Identifikation:

relevante Objekte

Aspekte

Beschr\u00f6nkungen m\u00f6glicher Situationen

Aktionen

Beschr\u00f6nkungen m\u00f6glicher Aktionen

Aktionen Voraussetzungen durch

Aktionen Nachbedingungen

Modellierung z.B.:

Menge von Symbolen

Paare von Mengen

Menge von

Menge von Termen

zustands\u00fcberg\u00e4nge

$\rightarrow \subseteq Z \times E \times Z$

Konzepte

Wertebereich

Angabe

Meng. gleichartiger Werte  
f\u00fcr Modell

extensional

Aufz\u00f6hlung

intensional

deutl. Bedingung

f\u00f6r l\u00f6sbarkeit

Russells Paradoxon

$M = \{x \mid x \notin M\}$

nicht wohldefiniert

induktive Definition,  $\in, 0, v, \lambda, S(m), X, n$ -Tupel  $(m_1, \dots, m_n)$ ,  $M^*, M^+, M^*$

Relation  $R \subseteq M^{(n)} \times \dots \times M^{(n)}$   $(s_1, \dots, s_n) \in R \rightsquigarrow R(s_1, \dots, s_n) \rightsquigarrow s_1 R s_2$

Beschr\u00f6nkung

reflexiv  $\forall x R(x, x)$

symmetrisch  $R(x, y) \wedge R(y, x) \Rightarrow R(x, x)$  Äquivalenzrelation

transitiv  $R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$

asymmetrisch  $R(x, y) \Rightarrow \neg R(y, x)$

alternativ  $R(x, y) \vee R(y, x)$

antisymmetrisch  $R(x, y) \wedge R(y, x) \Rightarrow x = y$  (partielle) Ordnung

totale/lineare Ordnung

Funktion  $f: D \rightarrow B$  F\u00f6r d

Teilmenge  $D \times B$   $\rightarrow$  Definitionsbereich

Funktion Relation  $D \times B$  Bild

$\forall d \in D$  h\u00f6chstens ein  $b \in B$  mit  $(d, b) \in f$

D  $\rightarrow$  B

Menge Funktionen

$D \rightarrow B$  alle Teilelemente

$f(x) \perp$  - f\u00fcr x nicht definiert

surjektiv  $\forall y \exists x f(x) = y$

injektiv  $\forall x_1 x_2 f(x_1) = f(x_2) \Rightarrow x_1 = x_2$

bijektiv

Verwendung von Indexmengen (und  $\in$  in Ind.)

Implizite Modellierung  $\perp$  von Objekten  
durch Bestandteile  $\perp$  Objektgleicher Bestandteile

Unterspezifizierte Wertebereiche + erfordert keine  
 $M = \{\dots\} \vee N +$  undefiniert

Kenntnis aller  
Instanzen

Kardinalit\u00e4t

$|M| =$  bei endlichem M  
Anzahl Elemente

$|M| \leq |N|, \dots$   
 $|M \times N| = |M| \cdot |N|$   
 $|P(M)| = 2^{|M|}$

Punktvereinigung  $|I = \{i_1, \dots, i_k\}|$  Menge

$W(I) \cup \dots \cup W(N) = \{i_{1,1}, \dots, i_{1,n_1}, \dots, i_{k,1}, \dots, i_{k,n_k}\}$

Folgen  $M^*, M^+, M^{\infty}$   
auch als  $f: \mathbb{N} \rightarrow M$

unendliche z.B. f\u00fcr nicht terminierende

Multimenge \u222a M z.B. Ressourcen

$m: M \rightarrow \mathbb{N}_0$  Info \u222a Anzahl

Pr\u00f6dikate \u222a M. p: M  $\rightarrow \mathbb{R}$  B = f(w, f<sup>3</sup>)

$W_p = \{m \in M \mid p(m) = w\}$  K \u222a M charakteristische Funktion

$F_p = \{m \mid p(m) = f\}$   $X(m) = \begin{cases} w & \text{falls } m \in f \\ \perp & \text{sonst} \end{cases}$

$\sqcup: M \times M \rightarrow M$  Vereinigungsoperator

- kleinste obere Schranke

$m_1 \leq (m_1, m_2) \wedge m_2 \leq (m_1, m_2)$

$\forall m: m_1 \leq m \wedge m_2 \leq m \Rightarrow (m_1, m_2) \leq m$

$\sqcap$  Schnittoperator - gr\u00f6\u00dftige untere Schranke

Verband  $(M, \leq, \sqcup, \sqcap)$

Hasse-Diagramm f\u00fcr partiell geordnete  $(M, \leq)$   
 $\forall m \in M$  Knoten  
 $m_1 \leq m_2 \Rightarrow m_2$  oberhalb von  $m_1$   
nur direkte Verbindungen erlaubt

BRUNNEN

erw\u00e4hnt:

ressourcen-  
konstruktive  
Logik, T\u00f6ringing

✓

# Aussagenlogik

## Illustration: Wahrheitstafel

Alphabet	Syntax
Logikkonstanten	true, false
true, false	$A \rightarrow B$
Junktoren	$(\neg A)$
$\top, \bot$	$(A \wedge B)$
Klammersymbole	(sugar Notation) nur überführbar
$(,)$	$\wedge, \Rightarrow, \neg$ Präzedenz $(, \wedge, \vee, \neg)$ Weglassen, $\neg$ wenn rekonstruierbar
$\alpha$ spezifiziert	
$\models$ Menge Belegungen	

## AL(V)

true, false

$A \rightarrow B$

$(\neg A)$

$(A \wedge B)$  nur

sugar (Notation)

$\wedge, \Rightarrow, \neg$  Präzedenz

(,) Weglassen,  $\neg, \wedge, \vee, \neg$

wenn rekonstruierbar

## Semantik

Belegung  $\beta: V \rightarrow \{w, f\}$

$\beta: AL(V) \rightarrow \{w, f\}$

true $^{\beta} = w$  false $^{\beta} = f$

$A^{\beta} = \beta(A)$

$(\neg A)^{\beta} = \text{nicht } A^{\beta} \text{ w. nicht } A^{\beta} = f$

$(A \wedge B)^{\beta} = \{w \text{ } \alpha^{\beta} = w \text{ und } \beta^{\beta} = w\}$

$(A \wedge B)^{\beta} = \{f \text{ } \alpha^{\beta} = f \text{ und } \beta^{\beta} = f\}$

semantische Äquivalenz

$\alpha \approx \beta \Leftrightarrow \alpha^{\beta} = \beta^{\beta} \forall \beta$

## Normalformen

Literal:  $A, \neg A$

KNF: konj. über Disj. ( $\neg$ ) $A$

DNF: Disj. über Konj. ( $\neg$ ) $A$

Von semantisch Äquivalent

Normalform

## Modellbeziehung

$J \models \alpha \Leftrightarrow J$  Modell von  $\alpha$  erfüllt

$\emptyset \models \alpha \Leftrightarrow \alpha \text{ w. } (\text{Knf})$

Erfüllbarkeit  $J \models \alpha \Leftrightarrow (\text{Knf})$

Allgemeingültigkeit  $\vdash \alpha: J \models \alpha \Leftrightarrow (\text{Knf})$

$\alpha$  allgemeingültig  $\Leftrightarrow$   $\neg \alpha$  nicht erfüllbar

Semantische Folgerungsbeziehung  $\vdash \alpha \vdash \beta: J \models \alpha \Rightarrow J \models \beta$  für Knf

$\vdash \alpha \vdash \beta: J \models \alpha \Rightarrow J \models \beta$  für Knf

$\vdash \alpha \vdash \beta: J \models \alpha \Rightarrow J \models \beta$  unerfüllbar

## Endlichkeitssatz

$\emptyset$  erfüllbar  $\Leftrightarrow$  Endlichen  $\emptyset \subseteq \emptyset$  erfüllbar

derivate:  $\emptyset \vdash \alpha \Leftrightarrow$  Endlich  $\emptyset \models \alpha$

$\emptyset$  unerfüllbar  $\Leftrightarrow \emptyset$  unerfüllbar

# Prädikatenlogik

## Alphabet

Logik Alphabet AL

: Quantor

=

n<sup>2</sup>:  $k_0, k_1, \dots$  Termkonstanten

$P_n$  Menge aus n-stelligen Prädikat symbolen

$F_n$  Menge aus n-stelligen Funktionssymbolen

## Semantik

$\Sigma$ -Struktur  $A = (T, a)$

T nicht leere Trägermenge  
a Abbildung (Grundbereich)

$k \in T \mapsto a(k) \in T$

$f \in F_n \mapsto$  n-stellige Funktion über T

$P \in P_n \mapsto$  n-stelliges Prädikat über T

Multifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

$\beta[r \mapsto x] = (d, \beta[r \mapsto x])$

Modifikation Belegung

$\beta[r \mapsto x](u) = \beta(u)$  falsch wtr

# Formale Sprache

$\Sigma \neq \emptyset$  Alphabet / Menge Buchstaben

Buchstabensequenz  $\rightarrow$  Wort auch ohne () geschrückt

Konkatenation,  $\epsilon = ()$  leeres Wort

Klassifikation

einfach  $\Leftrightarrow L = \emptyset$  oder  $L = \{\epsilon\}$  qFA

regulär  $\Leftrightarrow$  aus einfachen mit  $V, \cup, ^*$  erzeugbar

$$L_1 \cup L_2 = \{v \in V \mid v \in L_1 \text{ oder } v \in L_2\}$$

$$L^* = \{L\}$$

$$L^{i+1} = L^i L$$

$$L^+ = \{v \in V \mid v \in L^*\}$$

$$L_1 \cdot L_2 = \{v \in V \mid v \in L_1 \text{ und } v \in L_2\}$$

$\Sigma$ -Sprache  $L \subseteq \Sigma^*$

Formalismen zur Spezifikation /

Beschreibungs möglichkeiten reguläre Sprache

+ direkte Angabe bei endlichen L

reguläre Ausdrücke

Syntax

$\emptyset \in REG(\Sigma)$

$q \in Q$

$(\alpha + \beta) \in \alpha, \beta \in REG(\Sigma)$

$(\alpha \beta) \in \alpha, \beta \in REG(\Sigma)$

$(\alpha^*) \in \alpha \in REG(\Sigma)$

Semantik  $\Rightarrow REG(\Sigma)$

$$L(x) = \begin{cases} x = \emptyset \rightarrow \{\} \\ x = q \rightarrow \{q\} \\ (\alpha + \beta) \rightarrow L(\alpha) \cup L(\beta) \\ (\alpha \beta) \rightarrow L(\alpha) \cdot L(\beta) \\ (\alpha^*) \rightarrow L(\alpha)^* \end{cases}$$

$L$  regulär

$\Leftrightarrow$  gibt regulären Ausdruck

$\Leftrightarrow$  reguläre Grammatik die L spezifiziert

$\Leftrightarrow$  DFA

NDFA

kontextfrei  $\Rightarrow$  Kellerautomat

z.B. Pausen

Grammatiken allgemein

Syntax

$G = (\Sigma, V, X_0, P)$

$\Sigma$ , endliches Terminalalphabet

$V$ , Menge Variablen  $\Sigma \cap V = \emptyset$

$X_0 \in V$  Startsymbol

$P \subseteq (\Sigma V)^* \times (\Sigma V)^*$

Schreibweise bei klarem Kontext

$P = U_1 + V_1$  P zur Spezganz 6

BNF - Backus-Naur Form

$u ::= r_1 | r_2 | \dots | r_n$

für  $u \rightarrow r_1$

$u \rightarrow r_2$

$\vdots$

$r_i \in r_n$

DFA: deterministische endliche Automaten

Syntax

$A = (E, Q, S, q_0, A)$

$E$ -Menge Aktionen

$Q$ -Menge Zustände  $\neq \emptyset$

$S: Q \times E \rightarrow Q$  Zustandsübergangsfunktion

totale Funktion

$q_0 \in Q$  Anfangszustand

$A \subseteq Q$  Menge akzeptierenden Zustände

Semantik

$x'$  aus  $x$  ableitbar

$\Leftrightarrow x = x_1$  oder

$(x = u \text{ or } v \text{ or } w \text{ or } \dots)$

$\wedge x'$  aus  $u v w \dots$  ableitbar

$\wedge x' \stackrel{S(E,V)}{\Rightarrow} b$  ableitbar

$\Rightarrow$  aus  $x_0$  ableitbar

$$L(G) = \{u \in \Sigma^* \mid u \text{ ist in } G \text{ ableitbar}\}$$

Semantik

$u$  wird in  $A$  akzeptiert

$\Leftrightarrow u \in L(A)$

$\forall u = a_1 \dots a_n, S(q_0, a_1) = q_1$

$\wedge q_2 \dots q_n$  in  $q$  akzeptiert

$u \stackrel{A}{\rightarrow} q$

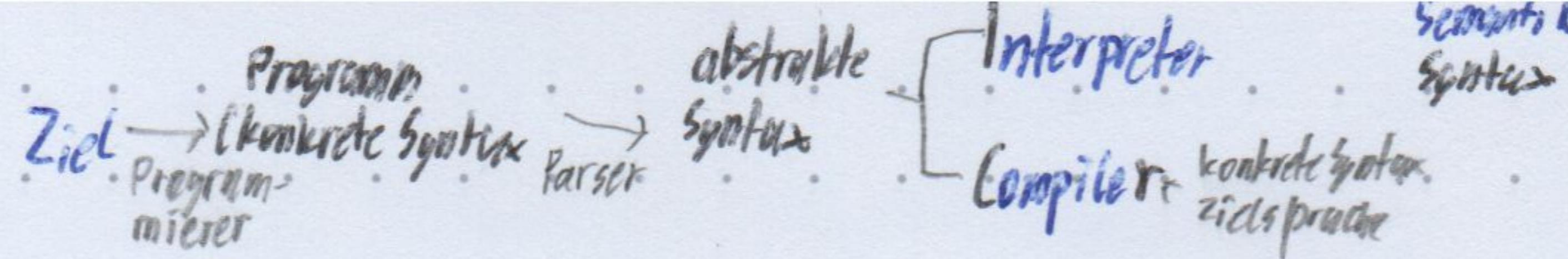
$$L(A) = \{u \mid u \text{ wird } A \text{ akzeptiert}\}$$

NDFA  $A \subseteq Q \times E \times Q$  Zustandsübergangsrelation  $(q, a, q') \in A$

Äquivalenz Spezifikationen, wenn gleiche Modelle spezifizieren  
aber unterschiedlich intuitiv.

z.B. Deadlocks von NDFA's nicht in Semantik erfasst.

# Programmiersprachen



## Syntax

oft Grammatik in BNF und Intuition  
syntaktische Gleichheit = gleiche Ableitung

## operationalle

## formale Semantik

Substitution - Zuordnung in Metavariablen  $\rightarrow$  Ausdrücke

geschrieben  $[x_1 \mapsto t_1, \dots, x_n \mapsto t_n] = n$

$\alpha \eta =$  Ersetze jedes freie Auftreten  $x_i$  durch  $\eta(x_i)$

Grundsubstitution - keine Metavariablen im Bildbereich

## 5 Urteil - Schema für Ausdrücke

keine Metavariablen enthalten

5 Instanz - durch Ersetzen aus Urteil

Grundinstanz - keine Metavariablen mehr

## Kalkül - Menge Kalkülregeln

erlaubt Herleitungen von Urteilstypen - grammatisch

Angemessenheit - Herleitbare Grundinstanzen zutreffend

## Kalkülregel

$$\text{r-name} \frac{s_1, \dots, s_n}{s} \quad \begin{array}{l} \text{Prämissen} \\ \text{kennen leeren} \\ \text{Seitenbedingungen} \end{array}$$

(Grund-)Instanz der Kalkülregel

- Substitution  $\eta$  Prämissen, Konklusion und  $\phi_1 \eta, \dots, \phi_n \eta$  erfüllt

Notation  
 $a \vdash b$

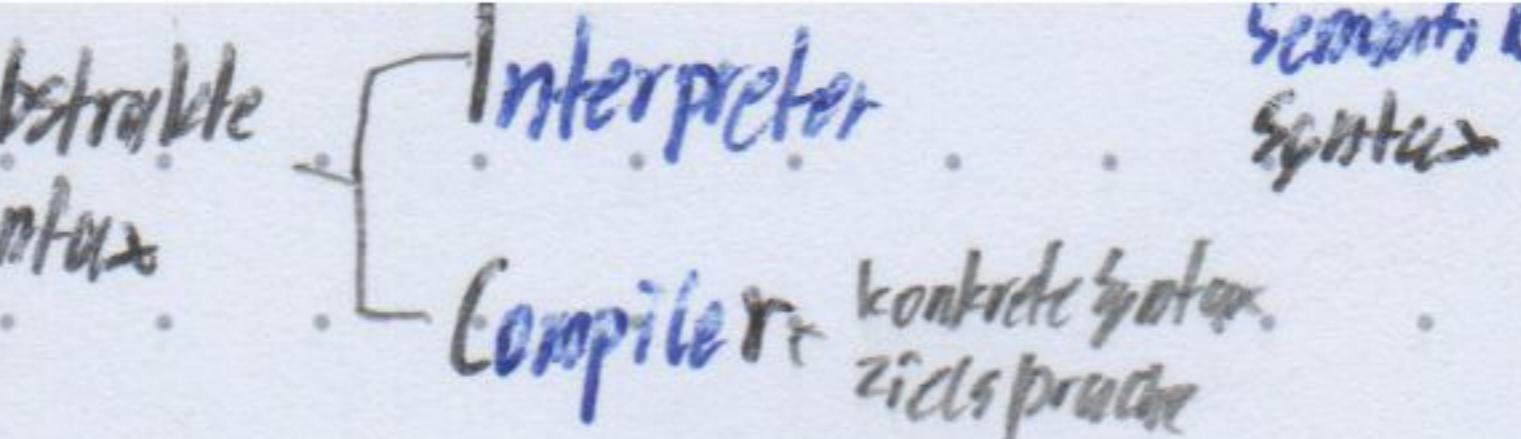
Grundinstanzen, semantisch äquivalent  
 $\Leftrightarrow$  Zustände gleicher herleitbar

Instanzen, wenn alle ihre Grundinstanzen

$b_1, b_2$  wenn  $\vdash b_1, \sigma \Vdash \text{true}$  herleitbar  
 $\Rightarrow \vdash b_2, \sigma \Vdash \text{true}$  herleitbar  
 $\vdash b_1, \sigma \Vdash \text{false}$  herleitbar  
 $\Leftrightarrow \vdash b_2, \sigma \Vdash \text{false}$  herleitbar

$a_1, a_2$  wenn  $\vdash a_1, \sigma \Vdash \text{true}$  herleitbar  
 $\Rightarrow \vdash a_2, \sigma \Vdash \text{true}$  herleitbar

$c_1, c_2$  wenn  $\vdash c_1, \sigma \Rightarrow \sigma'$  herleitbar  
 $\Leftrightarrow \vdash c_2, \sigma \Rightarrow \sigma'$  herleitbar



Bsp. IMP-einfache imperativer Programmiersprache

Num = {03 vfn 1n613vf-nnem} Metavariablen  
Bool = {true, false} + bezeichnen beliebiges Element aus Menge  
Var = {unspezifiziert}  $X, Y$

AExp:  $a ::= n | X | (a \oplus a) | (a \ominus a)$

BExp:  $b ::= t | (a \text{eq } a) | (a \text{leq } a) | \text{not } b | (b \text{and } b) | (b \text{or } b)$

Com:  $c ::= \text{skip} | X := a | c;c | \text{if } b \text{ then } c \text{ else } c \text{ fi}$

In while b. do. c od

Programm synonym zu Com

Zustand  $\sigma$ : Var  $\rightarrow$  Num

Menge aller Zustände  $\Sigma$

$$\sigma[x \mapsto n] = \begin{cases} \sigma(t) & \text{falls } t \neq X \\ n & \text{falls } t = X \end{cases}$$

Urteil  $\langle a, \sigma \rangle \Vdash n$  wird in  $\sigma$  zu  $n$  ausgetauscht

kalkül:

$$r \oplus \frac{\langle a_1, \sigma \rangle \Vdash n_1 \quad \langle a_2, \sigma \rangle \Vdash n_2}{\langle (a_1 \oplus a_2), \sigma \rangle \Vdash n} \quad n = n_1 + n_2$$

$r \ominus, r \otimes$  analog

$$r \text{Var} \frac{}{\langle X, \sigma \rangle \Vdash n} \quad n = \sigma(X)$$

$$r / / \text{Com} \frac{}{\langle b, \sigma \rangle \Vdash n}$$

$$\langle b, \sigma \rangle \Vdash t \text{ analog}$$

kalkül

$$r \text{true} \frac{}{\langle \text{true}, \sigma \rangle \Vdash \text{true}} \quad r \text{false} \text{ analog}$$

$$\text{regt} \frac{\langle a_1, \sigma \rangle \Vdash n_1 \quad \langle a_2, \sigma \rangle \Vdash n_2}{\langle (a_1 \text{eq } a_2), \sigma \rangle \Vdash n} \quad n_1 = n_2$$

regf, regt, rif, rff analog

$$r \text{notf} \frac{}{\langle b, \sigma \rangle \Vdash \text{false}}$$

rnotf, rnotf1, rnotf2

$$r \text{andf1} \frac{\langle b_1, \sigma \rangle \Vdash \text{false}}{\langle b_1 \text{ and } b_2, \sigma \rangle \Vdash \text{false}}$$

rnot1, rnot2, rorf analog

beschreibt  $\langle c, \sigma \rangle \rightarrow \sigma'$   $c$  weitet in  $\sigma$  zu  $\sigma'$  aus  
nur terminierende Kalkül

$$\text{rsk} \frac{\langle \text{skip}, \sigma \rangle \rightarrow \sigma}{\langle X := a, \sigma \rangle \rightarrow \sigma'} \quad t := \frac{\langle a, \sigma \rangle \Vdash n}{\langle X := a, \sigma \rangle \rightarrow \sigma'} \quad \sigma' = \sigma[X \mapsto n]$$

$$\text{rwit} \frac{\langle b, \sigma \rangle \Vdash \text{true}}{\langle (c, \sigma) \rightarrow \sigma' \rangle \rightarrow \sigma'} \quad \text{abitib doc od, } \langle c, \sigma \rangle \rightarrow \sigma'$$

$$\text{rnif} \frac{\langle b, \sigma \rangle \Vdash \text{false}}{\langle \text{while } b \text{ do } c \text{ od}, \sigma \rangle \rightarrow \sigma'} \quad \text{rif, riff analog}$$

## Beweis semantische Äquivalenz

$a \sim b \Leftrightarrow \langle a, \dots \rangle \text{ herleitbar} \Rightarrow \langle b, \dots \rangle \text{ herleitbar}$

~~Teil 1:~~  
muss folgende Form haben:  $\vdash H_1 \vdash \dots \vdash \vdash \langle a, \dots \rangle$  Fallunterscheidung für Kalkülregeln mit denen  $a$  hergeleitet sein kann  
„Es gibt Möglichkeiten für die letzte Regel in der Herleitung.“

Herleitung  
Prämissen, die ja existieren müssen  
 $\vdash \frac{\langle ? , \dots \rangle}{\langle a, \dots \rangle}$   $\Rightarrow$  Es gilt z.B.  $\sigma = \sigma'$   
Wir haben eine Herleitung  $H_1$  für  
Konstruieren Herleitung von  $\langle b, \dots \rangle$

Teil 2: analog

## Widerspruchsbeweis $\langle \dots \rangle$ nicht herleitbar

Angenommen es gäbe Zustände  $\vdash$  sodass  $\langle \dots \rangle$  herleitbar

Sei  $H$  eine Herleitung von  $\langle \dots \rangle$  sodass es keine mit weniger Regelnwendungen gibt. (eine kleinste Herleitung)

Fallunterscheidung letzte Regel

(muss folgende Form haben)  $\vdash$  Es folgt z.B.  $\sigma = \sigma'$

Somit hat  $H$  folgende Form  $\vdash \frac{\vdash \langle \dots \rangle}{\langle \dots \rangle}$

$H_1$  hat weniger Regelnwendungen als  $H$   
 $\Rightarrow$  nicht kleinste  $\square$

## Induktion

### wohlfundierte (Grundlage)

#### Beweisprinzip

$P \subseteq D$  einstellige Relation auf  $D$  - Induktionsformel

$\subseteq D \times D$  wohlfundierte Relation

$\forall d \in D : [(\forall d' \in D : (d' \neq d \Rightarrow P(d')) \Rightarrow P(d))]$

$\forall d \in D : P(d)$  Induktionsannahme

$\subseteq D \times D$   
unendlich absteigende Kette  
~~→ unendliche Folge, wenn  $f_i$~~   
 $\Leftrightarrow f : \mathbb{N} \rightarrow D$  mit  $f_i \in \mathbb{N} : f(i+1) \leq f(i)$

wohlfundierte binäre Relation

$\Leftrightarrow$  keine absteigende unendliche absteigende Kette  
 $\Rightarrow$  irreflexiv, transitive Hülle wohlfundiert

kleinste  
transitive Hülle  $R^* \subseteq D \times D$  von  $R \subseteq D \times D$

$\forall d_1, d_2 \in D : [d_1 R d_2 \Rightarrow d_1 R^* d_2]$

$\forall d_1, d_2, d_3 \in D : [d_1 R^* d_2 \wedge d_2 R^* d_3 \Rightarrow d_1 R^* d_3]$

### Induktion auf $\mathbb{N}_0$

$m \vdash n \Leftrightarrow m + 1 = n$

$\forall n \in \mathbb{N}_0 : [(\forall n' \in \mathbb{N}_0 : (n' + 1 = n \Rightarrow P(n')) \Rightarrow P(n))] \Rightarrow \forall n \in \mathbb{N}_0 : P(n)$

$$\begin{aligned} & ((\forall n \in \mathbb{N}_0 : (n + 1 = 0 \Rightarrow P(n)) \Rightarrow P(0)) \wedge \forall n \in \mathbb{N}_0 : [n = 0 \Rightarrow ((\forall n' \in \mathbb{N}_0 : (n' + 1 = n) \Rightarrow P(n')) \Rightarrow P(n))]]) \Rightarrow \forall n \in \mathbb{N}_0 : P(n) \\ & \quad \downarrow \\ & \quad \text{true} \Rightarrow P(0) \quad \downarrow \quad \forall n'' \in \mathbb{N}_0 : [n'' + 1 = 0 \Rightarrow ((\forall n' \in \mathbb{N}_0 : (n' + 1 = n'' + 1 \Rightarrow P(n')) \Rightarrow P(n'' + 1))]]) \Rightarrow \forall n \in \mathbb{N}_0 : P(n) \\ & \quad \downarrow \\ & \quad P(0) \quad \downarrow \quad \forall n'' \in \mathbb{N}_0 : [P(n'') \Rightarrow P(n'' + 1)] \Rightarrow \forall n \in \mathbb{N}_0 : P(n) \end{aligned}$$

### Strukturelle Induktion EN

Menge als BNF  $a ::= n \mid a_1 \dots a_n$   
 $a \leq a \Leftrightarrow a$  direkter Teilausdruck von  $a$

$\forall a \in A : [(\forall a' \in A : a' \text{ direkter Teil ausdruck von } a \Rightarrow P(a')) \Rightarrow P(a)] \Rightarrow \forall a \in A : P(a)$

↓ Fallunterscheidung über Struktur

$\forall n \in \mathbb{N} : P(n) \wedge \forall a_1, a_2 \in A : P(a_1), P(a_2) \Rightarrow P(a_1 \dots a_2 \dots)$

$\Rightarrow \forall a \in A : P(a)$

~~Deterministische Auswertung~~ Bsp.  $\langle a, \dots \rangle \wedge \langle a, \dots \rangle \dots$  herleitbar  $\Rightarrow \dots$

Wir verwenden Beweisprinzip der strukturellen Induktion für

$P(a) = \vdash \vdash \langle a, \dots \rangle \wedge \langle a, \dots \rangle \text{ herleitbar} \Rightarrow \dots ]$

Fall  $\forall n \in \mathbb{N} : P(n)$  Die Herleitungen können nur folgende Form haben...  $\Rightarrow P(n)$   
Fall ...

## Termbeschreibung von Regeln

Rterme(r-name) = { r-name( $\xi\eta, (\xi_1\eta, \dots, \xi_n\eta)$ ) |  $\xi\eta, \xi_1\eta, \dots$  enthalten keine Metavariablen  
Repräsentation kalkülig }  $\Phi_1, \eta_1, \dots, \Phi_m, \eta$  sind erfüllt

U von Herleitungen aus  $\xi_1 \dots \xi_k$

r-name( $\xi, (H_1, \dots, H_n)$ ) EDER  $[H] \vdash_{(K)} \xi$  -  $H$  ist  $K$ -Herleitung von  $\xi$

oder

$\xi \in \{\xi_1, \dots, \xi_k\}$

wegelassen  
nur  
Existenzaussage aus Kontext trage optional

DER<sub>K</sub>( $\xi$ ) - Menge Herleitungen von  $\xi$

DER<sub>K</sub> - Menge Herleitungen

IMP

A Kalkül für  $\langle a, \sigma \rangle \Vdash \eta$

B  $\langle b, \sigma \rangle \Vdash t$

C  $\langle c, \sigma \rangle \rightarrow \sigma$

Induktionsprinzip

$H_i \vdash H \Leftrightarrow$  direkte Teilherleitung

Beispiel - deterministische Berechnung IMD

P(H) =

Fall r-name

Es ist zu zeigen

$\forall$

Basisfall

form  $\in \text{DER}_{-}(\dots)$  | P(H<sub>1</sub>), P(H<sub>2</sub>), ...

Seien

beliebig,  
sodass gilt

Es ist noch zu zeigen

Sei ...

# Formalisierung einer Semantik

Freiheitsgrade - weitere (einschränkende) Details  
 Urteil angemessen = Intuition entsprechend  
 Kalkül

z.B. vorgegebene Links- vor- Rechts- Auswertung

$$a_1 \models 1 \frac{\langle a, a \rangle \rightarrow_1 n}{\langle a, a \rangle \rightarrow n} \text{ heAtomer}$$

Großschrift  $\leftrightarrow$  Kleinschrift Semantik  
 f.e.g. INT  
 rechteilige Zerlegung

$$\vdash 2 \text{ Urteile } \left[ \begin{array}{l} \text{Großschrift} \\ \Rightarrow \end{array} \right] \frac{\langle a, a \rangle \rightarrow_1 \langle a, a \rangle \wedge \langle a, a \rangle \rightarrow n}{\langle a, a \rangle \rightarrow n} \text{ jaAtomer}$$

Kleinschrift Axiome:

$$\vdash \frac{}{\langle (a_1 \dots), a \rangle \rightarrow_1 n}$$

Schritte

$$\frac{\langle a_1, a \rangle \rightarrow_1 \langle a_1, a \rangle}{\langle a_1 \dots, a \rangle \rightarrow_1 \langle a_1 \dots, a \rangle} \text{ a, f, K}$$

Aquivalenzen Semantiken

$$\text{If Urteil } 1 \models \text{Urteil } 2 (\text{Urteil } 2)$$

Kalküle - was wird spezifiziert

Menge (induktiv definiert)

$$I_K = \{ \xi \mid \vdash_K \xi \}$$

Beweisprinzip der Regelinduktion

$$\text{Regeln } \& \text{ Substitutionen Seitenbedingungen } \Rightarrow [P(\xi_1, n) \wedge \dots \wedge P(\xi_n, n) \Rightarrow P(\xi, n)]$$

$$\Rightarrow \forall \xi \in I_K : P(\xi)$$

Abschluss-eigenschaft - einstellige Relation Pauf  $\mathcal{P}(M)$  mit  $\forall Q \subseteq M : \exists Q' \subseteq M : (Q \subseteq Q' \wedge P(Q'))$

Menge  $Q$  abgeschlossen unter r-name

$$\Leftrightarrow \forall \eta : (\#_1, n, \dots, \#_n, \neg) \subseteq Q \Rightarrow \xi_{1, n} \in Q$$

unter  $K \Leftrightarrow$  unter allen Regeln aus  $K$

$$\Leftrightarrow R_K(Q) \subseteq Q$$

Hüllenoperator - Extensivität, Monotonie, Idempotenz

$$\forall Q \subseteq M : f(f(Q)) = f(Q)$$

$$R_K(Q) = \{ \xi \mid \exists \xi_1, \dots, \xi_n : \text{r-name}(\xi, (\xi_1, \dots)) \in R \text{ Terme(r-name)} \wedge \xi_1, \dots, \xi_n \subseteq Q \}$$

$\vdash$  Menge des herleitbaren direkt

$$R_K(Q) = \overline{R}_K(Q) \cup Q$$

monoton

extensiv

$$R_K^0(Q) = Q$$

$$R_K^{i+1}(Q) = \overline{R}_K(R_K^i(Q))$$

$$R_K^*(Q) = \bigcup_{i \in \mathbb{N}_0} R_K^i(Q)$$

monoton  $f : \mathcal{P}(M) \rightarrow \mathcal{P}(M)$

$$\Leftrightarrow \forall Q, Q' \subseteq M : (Q \subseteq Q' \Rightarrow f(Q) \subseteq f(Q'))$$

extensiv  $f : \mathcal{P}(M) \rightarrow \mathcal{P}(M)$

$$\Leftrightarrow \forall Q \subseteq M : (Q \subseteq f(Q))$$

# Software Engineering

Why? > complexity (schneller als Produktivität)

- Ingenieursdisziplin
- Aspekte Softwareentwicklung
  - ↓ mehr Arbeitstechnik
  - Programmierung
  - Dokumentation

~~Software~~

## Softwarequalität

### funktional

z.B. Ein-/Ausgabe

### nicht funktional

z.B. Benutzbarkeit

Effizienz, Sicherheit

Wartbarkeit,

Zuverlässigkeit

## Dokumentation

### Benutzerdokumentation

Anleitung, Beschreibung, Funktionalität  
Systemvoraussetzung

### Entwicklungsdocumentation

#### Systemanforderung

##### ↓ Lastenheft und Pflichtenheft

interne Funktionsweise, Systemarchitektur

Testfälle / -ergebnisse

- Protokollieren Entwurfsentscheidungen

## Software Prozesse

### Menge Aktivitäten

Randbedingungen Entwicklung

- z.B. Spezifikation von Dienst, Rahmenbedingungen Software
- Anforderungsanalyse - Kundengespräche, Prototypen

-definition - → Dokumentieren

Machbarkeitsstudien, -Technologie verfügbar?

Marktanalyse - wirtschaftlich?

präzise miteinander konsistent

vollständig realistisch

### Programmentwicklung

#### Architektur - Spec.

Entwurf Komponenten Komponenten, Schnittstellen,

wie Gesamtsystem Beziehungen

Algorithmen - Umsetzung in Impl.

↓ Datenstrukturen

### Validierung

entspricht Wünschen?

△ subjektiv

### Verifikation

erfüllt Anforderungsspezifikation

#### Techniken

manuell - z.B. Inspektion

statische Analyse - z.B. Typprüfung, formale Verifikation

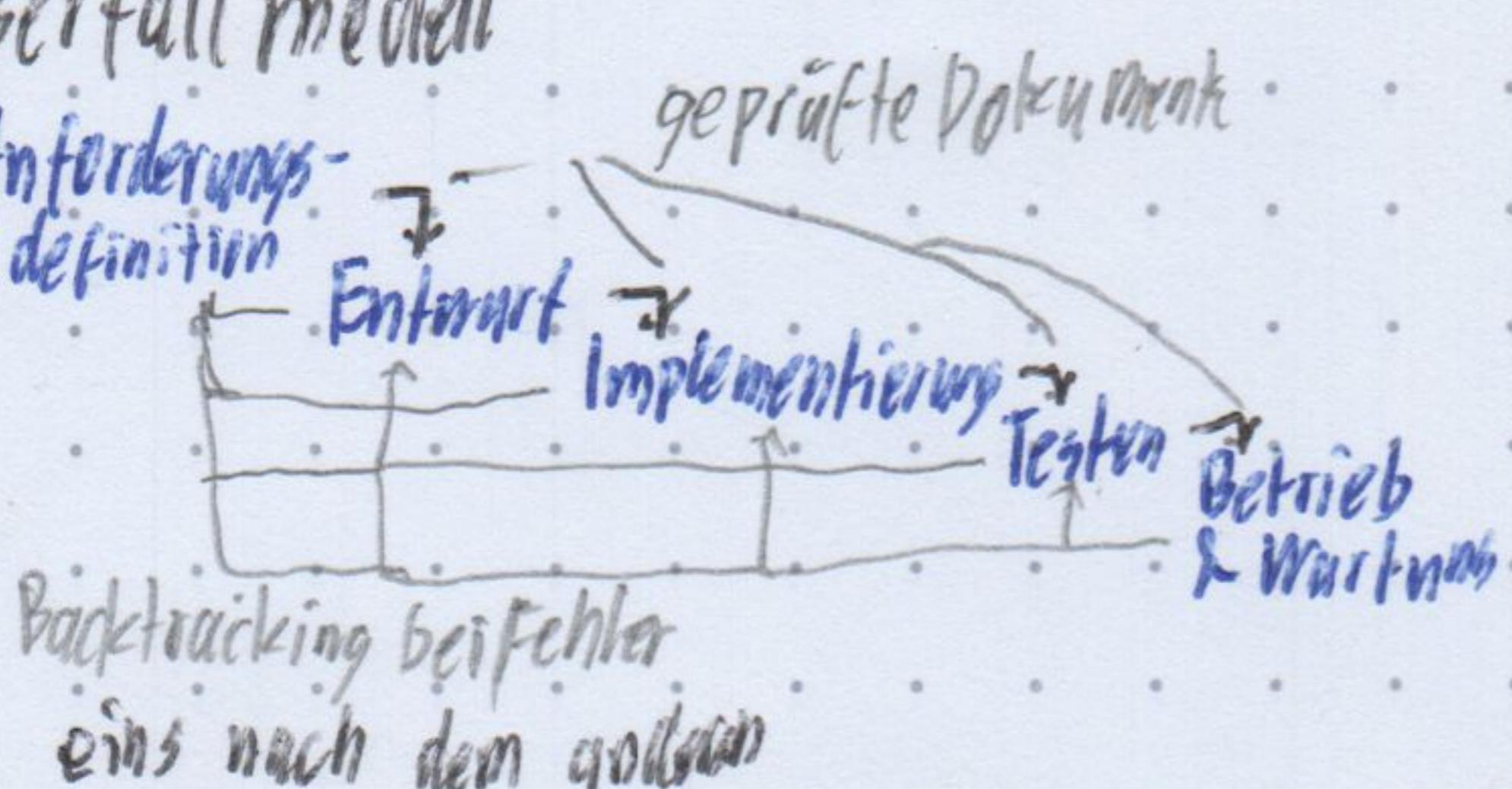
Testen - z.B. kontrollfluss-, Datenfluss-

orientiert

## Prozessmodell - verfeinerlt, ordnet Aktivitäten

### Wasserfallmodell

nur hier  
Auftraggeber  
beteiligung



### Spiralmodell

#### Zyklen mit 4 Sektoren

↓ Ergebnisse  
Prototypen

↓ Zielsetzung  
Risikoabschaffung,  
↓ Risikoreduktion

Entwicklung und Test  
Planung nächster Zyklus

+ Risiko berücksichtigt

### V-Modell

Anforderungsdef. → Anwendungszenario → Abnahmetest

Grubentwurf → Testfälle → Systemtest

Feinentwurf → Integrationstest

Modultest → Modultest

BRUNNEN

Produkte mit Bearbeitungszustand  
Attribut ändern → Validierung, Verifizierung  
fester Bestandteil

8

## Formelle Modellierung für Dokumentation & Denken

## Modelle aus Perspektive auf Abstraktionsstufe - z.B. Anforderungsicht

z.B. Black-box Sicht, vermutlich  
Architektur, fluss

SpezifikationsSprache - formale Notation  
Syntax  
Semantik - Abbildung  
in Mathematik

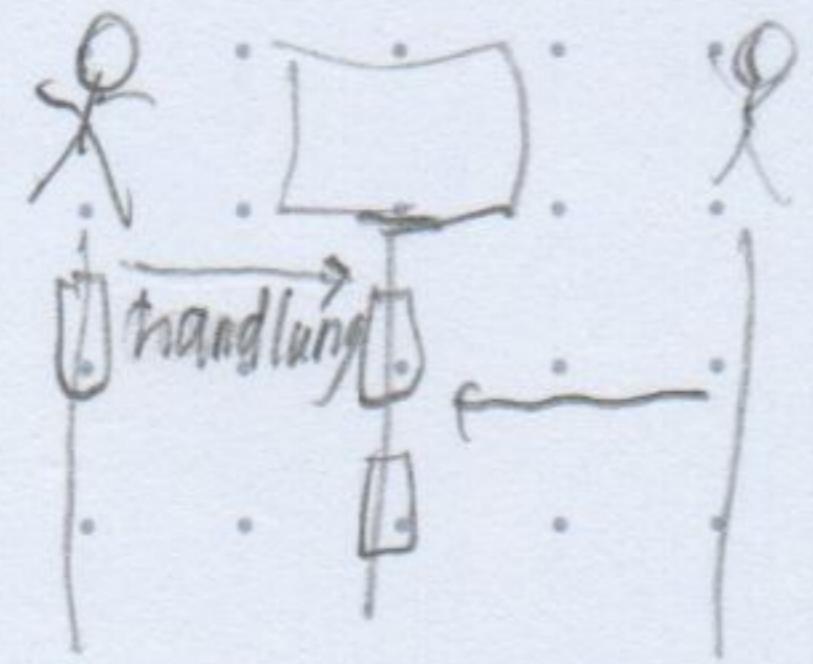
zur Spezifikation von Modellen  
beschreibt Modell  
Menge Modelle

graphische

Use Cases

gerichteter Graph  
Strichblätter - Akteure  
Ellipsen - Klassen möglicher  
AkteurInteraktion

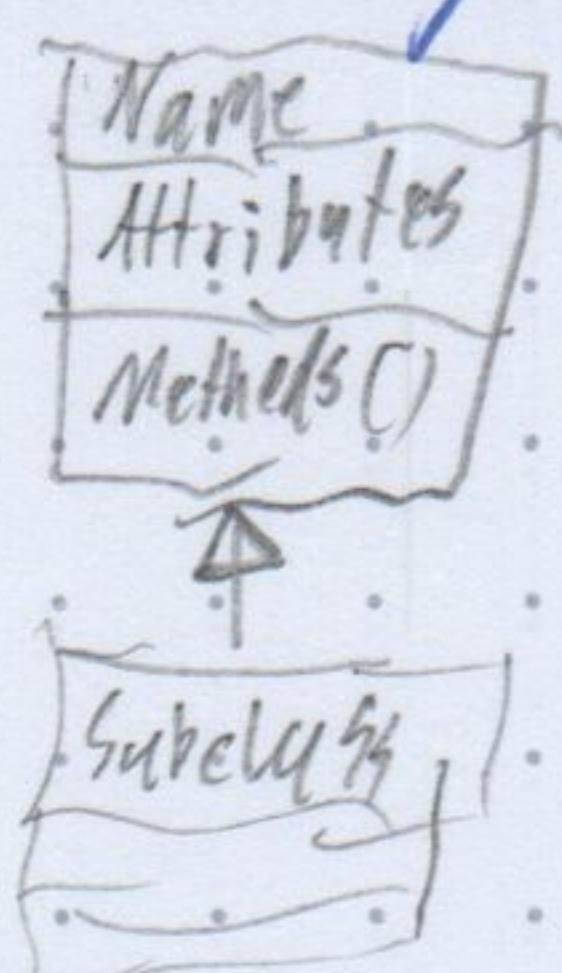
Aktionsfolgen



Datenflussdiagramm

beschriftete Pfeile - Datenfluss  
Rechtecke - Datenspeicher / Quelle  
abgerundete " - Berechnungsschritte

Klassendiagramm



Evaluierung Softwarequalität

Kriterien

Spec ~ Realität?  
Spec ~ Programm?

Häufige Kriterienwerte + Vergleichbarkeit  
fordern entl. formale Methoden

## Modelle aus Perspektive auf Abstraktionsstufe - z.B. Anforderungsicht

Architektur-  
Implementierung

informell  
semi-formal - semantik nicht formal  
formal - eindeutige Semantik

entl. (alle)  
Teil-  
Entwickler-  
dokumentation

Selectiv eingesetzt  
je nach kritischkeit  
z.B. CSP, CCS, PL-Kalkül,  
TLA, CTL, LTL/Z, CASL

Einsatz  
Anforderungen

vermeidet Mehrdeutigkeit  
Kommunikation mit Auftraggeber

Während Entwicklung  
zwischen Entwicklern

Verifikation auf Code/Modell Ebene  
z.B. Hoare Logik: EPL C EQL

- anspruchsvoll  
- zeitintensiv

Nachbelegung  
Programm (Imperativ) → Aussagenlogik  
Verbedingung

Beweise mit Kalkülen  
Werkzeugunterstützung  
erleichtert, "Qualität"  
editieren Spec/Beweis  
Überprüfen Syntax/Beweis  
Suche Beweissucht  
Vermutung Spec/Beweise  
z.B. B-tool, Coq, ISABELLE, RCF  
PVS, VSE

formelle Softwareentwicklung - Start: Spec Anforderung  
Ziel: Programm

Transformationsbasiert  
schrittweise Transformationen aus Objekt  
behalten Eigenschaften

Erfinde- und Verifiziere  
erfinde neue Spec.  
Beweise erhält Eigenschaften

# verhaltensorientierte Modellierung

Alternative Aspekte: Daten, Architektur

Bsp.  
S = Var  $\rightarrow$  Val

## Transitionssysteme $(S, S_0, E \rightarrow)$

$S =$  Menge der Zustände (Momentaufnahmen eines Systems)

$S_0 \subseteq S$ , Menge der Anfangszustände

$E =$  Menge der Ereignisse □ Menge Detail oft Designentscheidung

↳ Gegebenheiten, die Zustandsübergang auslösen können

Definition

↳ unterspezifiziert  
↳ Symbole  
↳ mathematische Konzepte  
(andere)

$S \times E \times S \ni \rightarrow =$  Transition relation

Notation:  $(s, e, s') \in \rightarrow$  auch  $s \xrightarrow{e} s'$

Spur - endliche Folge von Ereignissen & Zuständen  $\leftrightarrow$  Historien - unendliche Folgen

Ereignis- Spur - nur Ereignisse  
Zustands- Zustände

↳ z.B. für nicht/nach endlich Schritte terminierende Ausführungen

Operationen

Konkatenation

$$t_1 \cdot t_2 = \begin{cases} t_1, & \text{wenn } t_2 = () \\ (t_1, t_2), q, & \text{wenn } t_2 = (t_2, q) \end{cases}$$

Prefix

$t \cdot t' \Leftrightarrow tt' : t = t \cdot t'$

Projektion nach  $Q \subseteq SVE$

$$t \upharpoonright Q = \begin{cases} () & \text{wenn } t = () \\ (t \upharpoonright Q, q) & \text{wenn } t = (t, q), q \in Q \\ t \upharpoonright Q & " \quad " \quad q \notin Q \end{cases}$$

Einführung VFE für Terminierung

$$\exists n : \text{hist}(n) = \checkmark \rightarrow \forall n \geq n \text{ hist}(n) = \checkmark$$

Traces: Transitionssysteme  $\vdash P(\text{Spuren})$

$$\text{Hist}(ts) = \{ h : \mathbb{N}_0 \rightarrow (SVE)$$

$\exists t \in \text{traces}(ts)$ , wobei  $t \in S_0$

$$| h(0) \in S_0$$

$t, (s, e, s') \in \text{traces}(ts)$ , wobei  $t, (s, e, s') \in \text{traces}(ts)$

$$\forall n \in \mathbb{N}_0 : h(2+n) = h(2n+1)$$

$$h(2n+2)$$

$$S\text{-Traces}(ts) = \{ t \upharpoonright S \mid t \in \text{traces}(ts) \}$$

$$S\text{-Hist}(ts) = \{ h : \mathbb{N}_0 \rightarrow S \mid \exists h' \in \text{Hist}(ts) :$$

$$h \circ h' = h$$

$$h(n) = h'(2n+1)$$

$$E\text{-Hist}(ts) = \{ h : \mathbb{N}_0 \rightarrow E \mid$$

$$h'(2n+1)$$

## Modulare Modellierung

Komposition von Systemen üblich  $\Rightarrow$  für Modell auch

nebenläufige Ausführung ohne Kommunikation

$$S = S_1 \times S_2$$

$S_0 = S_0^1 \times S_0^2$  Synchron - gleichzeitige Berechnungsschritte

$$S \models E = E_1 \times E_2$$

$$\rightarrow = \{ ((s_1, s_2), (e_1, e_2), (s_1', s_2')) \mid (s_1, e_1, s_1') \in E_1 \times S, (s_2, e_2, s_2') \in E_2 \times S$$

$$\quad | s_1 \xrightarrow{e_1} s_1' \wedge s_2 \xrightarrow{e_2} s_2' \}$$

Theorem:

$$\tau_2 = \emptyset \Rightarrow \rightarrow_1 = f(s_1, e_1, s_1') \models S_1 \times E_1 \times S_1$$

$$\exists s_2, s_2' \in S_2 : \exists e_2 \in E_2 : (s_1, s_1') \xrightarrow{(e_1, e_2)} (s_1', s_2')$$

Analoges Bei  $\tau_1 = \emptyset \Rightarrow \tau_2 = \emptyset$

asynchron - unabhängige Berechnungsschritte

$$E = E_1 \vee E_2$$

$$\rightarrow = f(s_1, s_2) \xrightarrow{e} (s_1', s_2')$$

$$\quad | s_1 \xrightarrow{e} s_1' \wedge s_2 = s_2$$

$$\quad | s_2 \xrightarrow{e} s_2' \wedge s_1 = s_1' \}$$

Theorem:

$$E_1 \cap E_2 = \emptyset \Rightarrow \rightarrow_1 = \{ ((s_1, e_1, s_1'), (s_1, e_2, s_2)) \mid (s_1, e_1, s_1') \in E_1 \times S, (s_1, e_2, s_2) \in E_2 \times S \}$$

$\rightarrow_2$  analog

# Kommunikation zwischen Komponenten

Shared Memory ⚡ Schreibkonflikte

$TS_1, TS_2$  speicherkomponierbar  $\Leftrightarrow \exists SL_1, SL_2, S6$

$$S_1 = SL_1 \times SG$$

$$S_2 = SL_2 \times SG$$

## Asynchron

# Shared Memory Komposition TS

$$S := SL_1 \times SL_2 \times SG$$

$$S_0 := \{ (sl_1, sl_2, sg) \in SL_1 \times SL_2 \times SG$$

$$\{(sl_1, sg) \in S_0^1 \cdot 1 \mid (sl_2, sg) \in S_0^2\}$$

$$E := E_1 \vee E_2$$

$$\rightarrow := \{ (sl_1, sl_2, sg), e, (sl_1^{-1}, sl_2^{-1}, sg^{-1}) \} \in S \times E \times S$$

$$| (sl_1, sg) \rightarrow (sl_1', sg') \wedge e \in E_1 \wedge sl_2' = sl_2$$

$$V = 2 \cdot \dots \cdot 2 \cdot 2 \cdot \dots \cdot \dots \cdot 2 \cdot \dots \cdot 1 \cdot \dots \cdot 1$$

Message Passing - senden und empfangen gleichzeitig  
und Kanal modellieren

sonst. Kanal modellieren

L z.B. mit Slack Nachrichten

! gleiche Sender, Empfangsreihenfolge ?

# message-passing Komposition

$$S = S_1 \times S_2$$

$$S_0 = S_0^1 \times S_0^2$$

$$F = E_1 \vee E_2$$

$$\rightarrow \vdash \{((s_1, s_2), e, (s_1', s_2'))\} \in S \times E \times S$$

$$1 s_1 \rightarrow_1 s_2 \wedge_{\text{EF}} E_1 \backslash E_2 \quad 1 s_2 = s_2$$

$v^u$  7 2 2 2 1 1 1

$\text{V. } \text{Sg} \xrightarrow{\text{Sg}} \text{A. } \text{Sg} \xrightarrow{\text{Sg}} \text{MeetE}_1 \text{ of E}_1$

• • • • • • • • • •

$E \cap E_1 \wedge E_2$  gemeinsames Ereignis  
k. lokales

K. . . . Lekules

Prozess  $P = (E, Tr)$

$\alpha(P) = E$  Menge Ereignisse / Alphabet

$\text{traces}(P) = Tr \subseteq E^* \cup (E^* \times \{v\})$  - Menge Ereignisspann  
unter Präfixbildung abgeschlossen

Spezifikationssprache für Prozesse

Prozessausdrücke

Teilsprache CSP

$\text{STOP}_E = (E, \{\lambda\})$  - Spur, dass gar nichts passiert

$\text{SKIP}_E = (E \cup \{v\}, \{\lambda\}, \{v\})$  - Spur, dass System terminiert

$(x \rightarrow P) = (\alpha(P), \{\lambda\} \cup \{t(x), f\} \cup \text{traces}(P))$  - zuerst  $x$   
 $x \in \alpha(P)$   
 $x \neq v$

$(P \sqcap Q) = (\alpha(P), \text{traces}(P) \cup \text{traces}(Q))$  - P oder Q =  
 $\alpha(P) = \alpha(Q)$

Ausdrucks möglichkeit Sprache BNF  $P ::= \text{STOP}_E \mid \text{SKIP}_E \mid (x \rightarrow P) \mid (P \sqcap Q)$

utility function  $f(E, t, v)$   
 $\text{process}(E, t) = \begin{cases} \text{STOP}_E & \text{wenn } t = () \\ \text{SKIP}_E & \text{wenn } t = (v) \\ (x \rightarrow \text{process}(E, t')) & t = (x, t') \wedge x \neq v \end{cases}$

\* kann jeden Prozess mit  
endlicher Tr spezifizieren

unendliche Mengen von Spuren - unendlich lang laufende Systeme?

Rekursive Gleichungssysteme

speziell Prozess  $P ::= \text{STOP}_E \mid \text{SKIP}_E \mid (x \rightarrow P) \mid (P \sqcap Q) \mid id$   
der Form  $\{id\} = E P / id F / D$   
 $E(id) \quad \text{Menge Prozessbezeichner}$   
 $V_E = \{\text{STOP}_E, \text{SKIP}_E, \dots, M\} \wedge D = \emptyset$   
 $F(id) \quad \text{id} = E P / id F / D$   
 $V_F = \{\text{STOP}_E, \text{SKIP}_E, \dots, M\} \wedge D = \emptyset$   
 $D(id) \quad \exists id' \text{ mit } id = id' \wedge id' \in V_D$   
Semantik:  $id = (E, \text{traces}(P))$

Alternativ Fixpunktoperator

$\mu X : E, F(X) = (E, \text{Lösung der Gleichung})$   
 $X = E F(X)$   
 $\uparrow$  kleinster Fixpunkt  
 $\uparrow$  nicht  
 $id$  id + P

\* verhindert keine Lösung

Jede id darf nur einmal auf linker Seite auftreten

\* rechten Seiten bemächtigt - für jede id' rechts  
 $id = E P$

$\exists id' \text{ mit } id = id' \wedge id' \in V_D$   
Teilausdruck von  $P: (x \rightarrow P)$ ,  
sodass id' in  $P$  Wächter

\* verhindert mehrere Lösungen

Syntactic Sugar

$(P; Q) = (\alpha(P), \{s \in \text{traces}(P) \mid s \sqcap \{v\} = ()\} \cup \{s \in \text{traces}(P) \mid \forall s'. t \mid s. (v) \in \text{traces}(P) \wedge t \in \text{traces}(Q)\})$  - erst, dann

$(P || Q) = (\alpha(P) \cup \alpha(Q), \{t \in \alpha(P) \cup \alpha(Q)\}^* \mid (t \sqcap \alpha(P)) \sqcap (t \sqcap \alpha(Q)) \in \text{traces}(P) \wedge (t \sqcap \alpha(Q)) \in \text{traces}(Q)\})$  - nebenläufig  
sync gemeinsamer Ereignisz

$(P ||| Q) = (\alpha(P), \{s \in \alpha(P)^* \mid \exists t \in \text{traces}(P). \exists u \in \text{traces}(Q). s = t \sqcap u\})$  ohne sync  
SF interleaving (t, u)

interleaving(t, u) =  $\begin{cases} \{t\} & \text{für } u = () \\ \{t u\} & t = () \\ \{t(x). s \mid s \in \text{interleaving}(t', u)\} & t = (x), t' \\ \{t(x). s \mid s \in \text{interleaving}(t, u')\} & u = (u'), u' \end{cases}$

# Modellierung von Eigenschaften

formel: Systemmodell  $\xrightarrow{\text{erfüllt}}$  Anforderungsmodell

Wenn Systemmodell noch nicht feststeht  
 $\Rightarrow$  Annahmen treffen

P  $\square$  Q

wie P  $\sqcap$  Q, aber Systemumgebung kann beeinflussen ob wie R oder S unterschieden bei komplexerer Semantik

Beispiel Prädikat für Transitionssystem

$K_1(TS) = \forall h \in S \text{-Hist} : \exists n \in \mathbb{N} : \dots h(n) \dots$  trifft Aussage über mögliche Systemläufe

Beziehungen zwischen Anforderungen Beweisen

z.B. welche ist stärker z.B. bei Streit über Formalisierung

Hilfsfunktionen definieren

P sat. S ( $\Leftrightarrow$  Prozess P erfüllt Eigenschaft S)

$\Leftrightarrow \forall tr \in traces(P) : S(tr)$

für unter Prefixbildung abgeschlossen

$BEFORE_{FG}(t_1) = (\gamma(t_1 \upharpoonright G) = () \Rightarrow \gamma(t_1 \upharpoonright F = ()))$

Sinn

von Anforderungen

Auftraggeber, Entwickler

während Entwicklungs

zwischen Entwicklern

Vermeidet

Missverständnisse