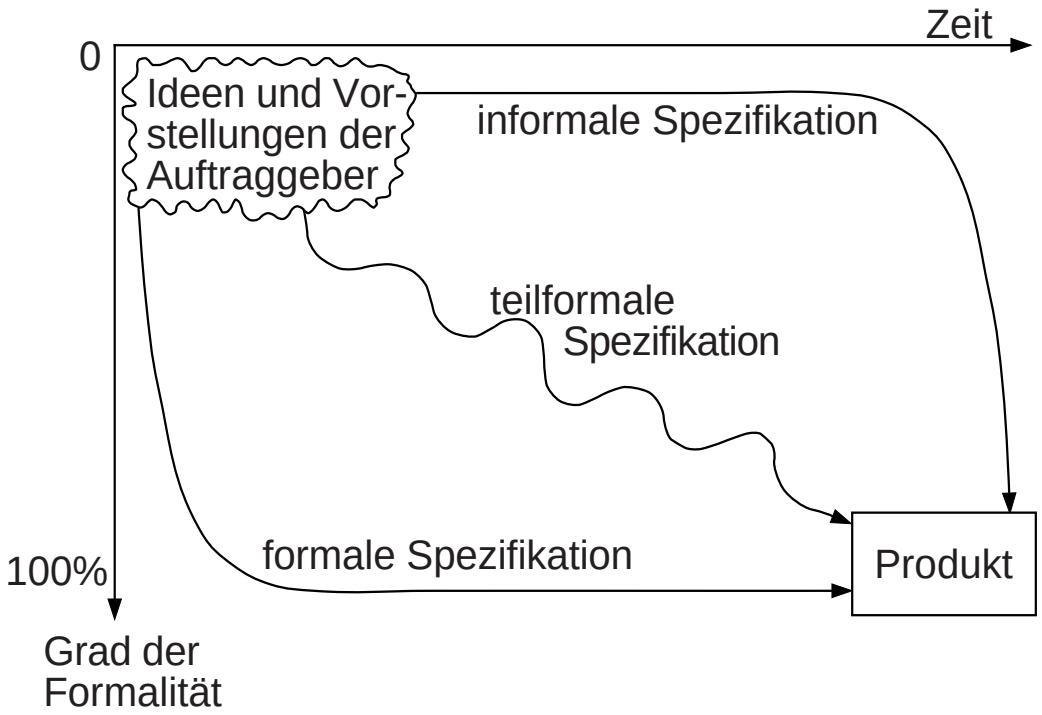
13 Formale Spezifikation von Anforderungen

Mögliche Formalitätsgrade einer Spezifikation



- informal, in der Regel deskriptiv mit natürlicher Sprache
- formal, deskriptive und konstruktive Verfahren möglich
- teilformal mit konstruktiven, anschaulichen Modellen

Formale Spezifikation

- Formaler Kalkül, d.h.
  Verwendung einer Spezifikationssprache mit
  - formal definierter Syntax

    und
  - formal definierter Semantik

- Primär für funktionale Anforderungen
- Verschiedene Arten:
  - Rein deskriptiv, zum Beispiel Algebraische Spezifikationen
  - Rein konstruktiv, zum Beispiel Petrinetze
  - Modellbasierte Mischformen, zum Beispiel VDM und Z

Stärken

- ☆ Immer eindeutig, da Semantik formal definiert
- ☆ Widerspruchsfreiheit formal prüfbar
- ☆ Erfüllung gewisser Eigenschaften beweisbar, zum Beispiel Sicherheitsanforderungen wie Nichterreichbarkeit gefährlicher Zustände
- ☆ Lösungsneutral
- ☆ Formale Verifikation von Programmen möglich
- ☆ Modelle simulierbar/animierbar, z.B. Petrinetze

Schwächen

- ☆ Erstellung sehr aufwendig
- ☆ Prüfung/Nachweis der Vollständigkeit wird nicht einfacher
- ☆ Ohne profunde Ausbildung nicht lesbar
  → Prüfung auf Adäquatheit durch den Kunden?
- ☆ Fehlende Mittel zur Dekomposition: Große Spezifikationen auch für Fachleute schwer verstehbar → Fehlererkennung? Prüfung auf Adäquatheit der Anforderungen? Korrekte Umsetzung der Anforderungen im Entwurf?
- ☆ Teilweise einseitig ausgerichtet:
  - ✧ Auf Funktionen (Algebraische Spezifikation)
  - ✧ Auf Verhalten (Petrinetze)
  - ✧ Aspekte wie Benutzerschnittstellen sind praktisch nicht modellierbar
  - ✧ Beschreibung von Ausnahmefällen schwierig
  - ✧ Zum Teil muss perfekte bzw. abstrakte Technologie angenommen werden

Machbarkeit / Wirtschaftlichkeit formaler Spezifikationen?

- ☆ Marginale Rolle in der Praxis
  - ✧ trotz theoretischer Vorteile
  - ✧ trotz intensiver Forschung (zum Beispiel Algebraische Spezifikation seit ca. 1977)
- ☆ Einsatz heute
  - ✧ Punktuell sinnvoll und möglich
  - ✧ Vor allem für sicherheitskritische Komponenten
  - ✧ Einsatz in der Breite
    - nicht möglich (Prüfung auf Adäquatheit!)
    - nicht sinnvoll (unwirtschaftlich)
- ☆ Auch möglich: Teilformale Spezifikation mit gezielter Formalisierung kritischer Teile

Algebraische Spezifikation

- Deskriptive, formale Methode
- Primär für die formale Spezifikation komplexer Datentypen
- Syntax durch Signaturen (Definitions- und Wertebereiche) der Operationen
- Semantik durch Axiome (Ausdrücke, die immer wahr sein müssen)
- Axiome beschreiben im Wesentlichen Invarianten unter der Anwendung von Funktionen

Beispiel: Algebraische Spezifikation einer Keller-Struktur (Stack)

Sei bool der Datentyp mit dem Wertebereich {false, true} und der Booleschen Algebra als Operationen. Sei ferner elem der Datentyp für die Datenelemente in der Keller-Struktur.

TYPE Stack

FUNCTIONS

new: () $\rightarrow$ Stack;	-- neuen (leeren) Stack anlegen
push: (Stack, elem) $\rightarrow$ Stack;	-- Element hinzufügen
pop: Stack $\rightarrow$ Stack;	-- zuletzt hinzugefügtes Element entfernen
top: Stack $\rightarrow$ elem;	-- liefert zuletzt hinzugefügtes Element
empty: Stack $\rightarrow$ bool;	-- wahr, wenn Stack kein Element enthält
full: Stack $\rightarrow$ bool;	-- wahr, wenn Stack voll ist

AXIOMS

$\forall$ s $\in$ Stack, e $\in$ elem

(1) $\neg$ full(s) $\rightarrow$ pop(push(s,e)) = s	-- Pop hebt den Effekt von Push auf
(2) $\neg$ full (s) $\rightarrow$ top(push(s,e)) = e	-- Top liefert das zuletzt gespeicherte Element
(3) empty(new) = true	-- ein neuer Stack ist leer
(4) $\neg$ full (s) $\rightarrow$ empty(push(s,e)) = false	-- nach Push ist ein Stack nicht mehr leer
(5) full(new) = false	-- ein neuer Stack ist nicht voll
(6) $\neg$ emtpy(s) $\rightarrow$ full(pop(s)) = false	-- nach Pop ist ein Stack niemals voll

Modellbasierte Spezifikation

- Mathematisches Modell des Systemzustands
- Basierend auf Mengen, Relationen und prädikatenlogischen Ausdrücken
- Beschreibung von
  - Grundmengen
  - Zusammenhängen zwischen diesen Mengen (Relationen, Funktionen)
  - Invarianten (Prädikate)
  - Zustandsveränderungen (Relationen, Funktionen)
  - Zusicherungen für Zustände
- Bekannte Vertreter:
  - VDM (Vienna Development Method, Björner und Jones 1978)
  - Z (Spivey 1992)
  - OCL

Beispiel: Spezifikation eines Bibliothekssystems mit Z (Ausschnitt)

Bibliothek

$$\begin{array}{l} Bestand: \mathbb{P}\ Buch \\ Benutzer: \mathbb{P}\ Person \\ ausgeliehen: Buch \mapsto Person \\ \hline \textbf{dom}\ ausgeliehen \subseteq Bestand \\ \textbf{ran}\ ausgeliehen \subseteq Benutzer \end{array}$$

Ausleihen

Δ Bibliothek

auszuleihendesBuch?: Buch

Ausleiher?: Person

---

auszuleihendesBuch? $\in$ Bestand \ **dom** ausgeliehen

Ausleiher? $\in$ Benutzer

ausgeliehen' = ausgeliehen $\cup$ {(auszuleihendesBuch?, Ausleiher?)}

Bestand' = Bestand

Benutzer' = Benutzer

AnfrageAusleihbar

Ξ Bibliothek

angefragtesBuch?: Buch

istAusleihbar!: {ja, nein}

---

angefragtesBuch? $\in$ Bestand

((angefragtesBuch? $\notin$ **dom** ausgeliehen $\wedge$ istAusleihbar! = ja) $\vee$

(angefragtesBuch? $\in$ **dom** ausgeliehen $\wedge$ istAusleihbar! = nein))

Nachweis geforderter Eigenschaften

Werden Anforderungen durch Modelle beschrieben, so möchte man häufig wissen, ob das Modell gewisse geforderte Eigenschaften aufweist.

Formale Spezifikationen erlauben

- die Gültigkeit von Eigenschaften zu beweisen

  Verfahren: mathematisch-logisches Schließen, unterstützt durch Theorembeweiser-Software

- die Gültigkeit von Invarianten automatisiert zu testen und ggf. Gegenbeispiele zu finden

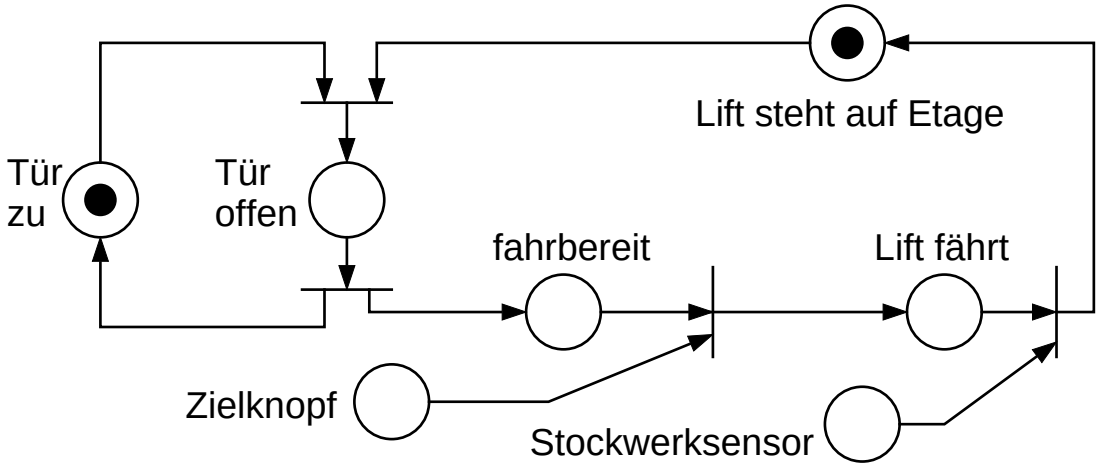
  Verfahren: Systematisches, automatisiertes Explorieren des gesamten Zustandsraums der Spezifikation und Prüfen der gewünschten Eigenschaft in jedem Zustand (Model Checking)

Eigenschaften, deren Gültigkeit man beweisen oder testen möchte, sind zum Beispiel sicherheitskritische Invarianten, etwa für eine Liftsteuerung, dass sie nie einen Lift mit offenen Türen fahren lässt.

Beispiel 1

Eine einfache Liftsteuerung sei wie folgt mit einem einfachen Petrinetz* modelliert:

*anonyme Marken, pro Stelle höchstens eine Marke, Transitionen ohne Bedingungen



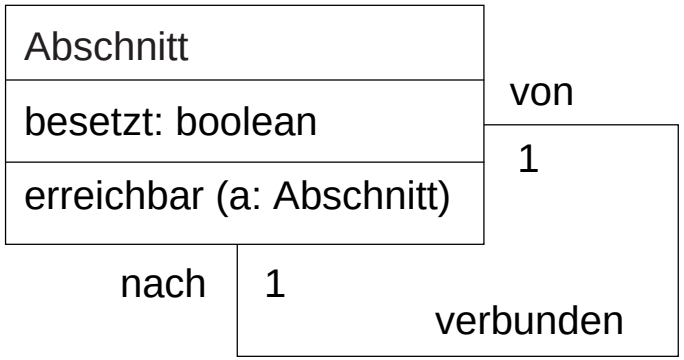
Die Sicherheitseigenschaft, dass der Lift bei offener Tür nicht fahren kann, ist für dieses Modell mit folgender Überlegung beweisbar:

In der rechten Schleife des Netzes kann nie mehr als eine Marke zirkulieren

- $\rightarrow$ Wenn Tür offen markiert ist, kann Lift fährt nicht markiert sein
- $\rightarrow$ Wenn Lift fährt markiert ist, kann Tür offen nicht markiert sein

Beispiel 2

Gegeben sei eine ringförmige S-Bahn-Strecke, die sich aus 10 aufeinanderfolgenden Streckenabschnitten zusammensetzt. Diese sei mit UML und OCL wie folgt modelliert:



context Abschnitt::erreichbar (a: Abschnitt): boolean
post:
result = (self.nach = a) **or** (self.nach.erreichbar (a))

context Abschnitt **inv**:
Abschnitt.allInstances->size = 10

Die Eigenschaft, dass in einem Ring jeder Streckenabschnitt vorn und hinten mit genau einem anderen Streckenabschnitt verbunden ist, wird im Modell durch eine Assoziation mit entsprechenden Kardinalitäten spezifiziert.

Da die Strecke einen Ring bildet, muss jeder Abschnitt von jedem anderen Abschnitt und von sich selbst aus erreichbar sein. Es muss also folgende Invariante gelten:

context Abschnitt **inv**:
Abschnitt.allInstances->forAll (x, y | x.erreichbar (y))

- Der Versuch, diese Invariante zu beweisen, misslingt

  [Warum? Finden Sie ein Gegenbeispiel]

- Lediglich folgende triviale Invariante ist beweisbar

  context Abschnitt **inv**:
  Abschnitt.allInstances->forAll (x | x.erreichbar (x))

  [Wie? Führen Sie den Beweis mit Hilfe der Definition der Operation erreichbar]

- ⇨ Das Modell der S-Bahn-Strecke ist falsch. Es bildet die Eigenschaft, ringförmig zu sein (und damit die Erreichbarkeit jedes Abschnitts von jedem Abschnitt aus) nicht korrekt ab.
  [Wie müsste das Modell verändert werden, damit die Erreichbarkeits-Invariante korrekt wird?]

Literatur

VDM: Björner, D., C. Jones (1978). The Vienna Development Method. Berlin, etc.: Springer.

Z: Spivey, J.M. (1992). *The Z Notation: A Reference Manual*. Second Edition. Hemel Hempstead: Prentice Hall International.

OCL: OMG (2001). OMG Unified Modeling Language Specification. Version 1.4.