Formale Spezifikation mit TLA$^+$

Fachseminar: *Software Engineering*

Alexander Weigl, Vortagsdatum

Motivation

TLA$^+$ represents the only effective **methodology** I've seen for visualizing and quantifying algorithmic complexity in a way that is meaningful to engineers.

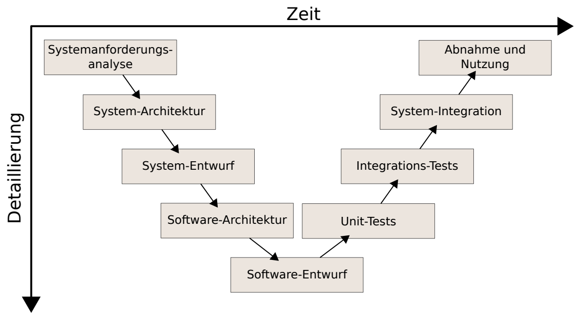
Brannon Barson, Processor Architect, Intel Corparation

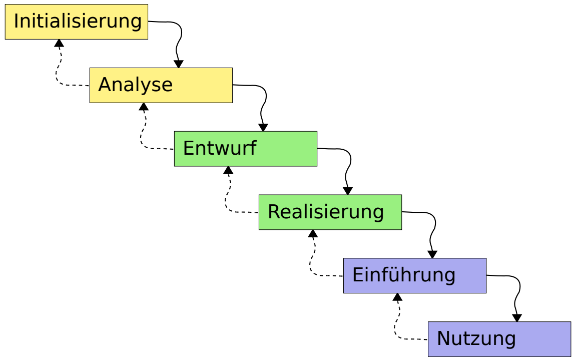
Überblick

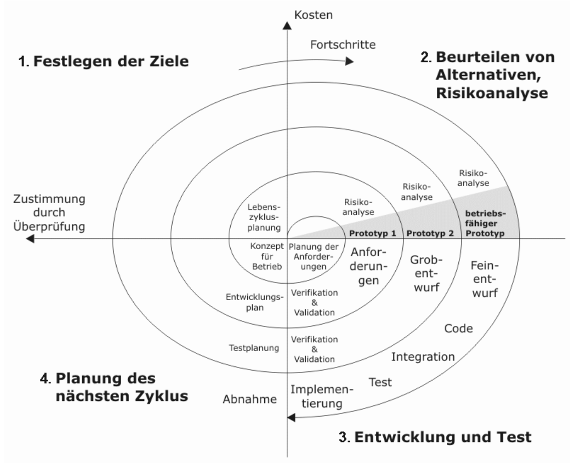
1. Formale Spezifikation
2. Prädikatenlogik (PL)
3. Lineare Temporale Logik (LTL)
4. TLA$^+$
5. Modelchecker
6. Zusammenfassung

Formale Spezifikation

- Anforderungen in formaler Sprache
- standardisiert
- Einsatz in der SWE?







früh oder spät?

[Lam02] S. 83, Bilder: wikipedia.org

Formale Spezifikation

- Vorteil
  - Validierung der Spezifikation (Konkretisierung, Machbarkeit)
  - Beweisbarkeit von Eigenschaften
  - Grundlage der Kommunikation
  - Modelchecker (ausführbar), Beweissystem
- Nachteil
  - Erlernen der Sprache, Tools
  - Erstellungsaufwand (Dokumentation!)
  - Abweichungen zw. Spezifikation/Implementierung
  - Wann verhält sich das System korrekt?

Wieso keine agile Methoden (Prototyp, TDD)?

[Lam02] S. 75

Grundlagen: Prädikatenlogik

- Erweiterung Aussagenlogik
- Aussagen über Mengen
- Bestandteile der Prädikatioenlogik:

$\wedge, \vee, \neg$	UND, ODER, NICHT
$\Rightarrow, \Leftrightarrow$	Impliziert, Äquivalenz
$P(x)$	Prädikate ($f: A^n \to \{wahr, falsch\}$)
$\forall$	Allquantor: $\forall x \in S: P(x) \equiv \bigwedge_{x' \in S} P(x')$
$\exists$	Existenzquantor: $\exists x \in S: P(x) \equiv \bigvee_{x' \in S} P(x')$

Beispiel: *greatest common divisor*

$$
\begin{aligned}
GCD(x,y,i) \equiv \quad & x \bmod i = 0 \\
& \wedge\, y \bmod i = 0 \\
& \wedge\, \forall\, 1 \le j \le i : \quad x \bmod j = 0 \\
& \qquad\qquad\qquad\quad \wedge\, y \bmod j = 0 \\
& \qquad\qquad\qquad\quad \Rightarrow i \le j
\end{aligned}
$$

$$GCD(x,y,i) \Leftrightarrow \gcd(x,y) = i$$

- i teilt x, y
- Wenn $1 \le j \le i$ Teiler von x, y ist, dann muss $j \ge i$.

Grundlagen: Lineare Temporale Logik

- Logik über zeitliche Ablauf von Zuständen
- Jeder Zustand: einen Nachfolger/Vorgänger

$$\cdots \rightarrow \left|\begin{array}{lcr} a & = & 130 \\ b & = & 31 \end{array}\right| \rightarrow \left|\begin{array}{lcr} a & = & 31 \\ b & = & 27 \end{array}\right| \rightarrow \left|\begin{array}{lcr} a & = & 27 \\ b & = & 4 \end{array}\right| \rightarrow \left|\begin{array}{lcr} a & = & 4 \\ b & = & 3 \end{array}\right| \rightarrow \left|\begin{array}{lcr} a & = & 3 \\ b & = & 1 \end{array}\right| \rightarrow \left|\begin{array}{lcr} a & = & 1 \\ b & = & 0 \end{array}\right| \rightarrow \cdots$$

- Zustand: σ_i eine Variablenbelegung (im Sinne TLA$^+$)
- Verhalten: $\sigma = \sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \cdots$
- Keine Vorschrift zur Erzeugung von Zuständen (Spezifikation).
- Spezifikation erzeugt Menge von Verhalten

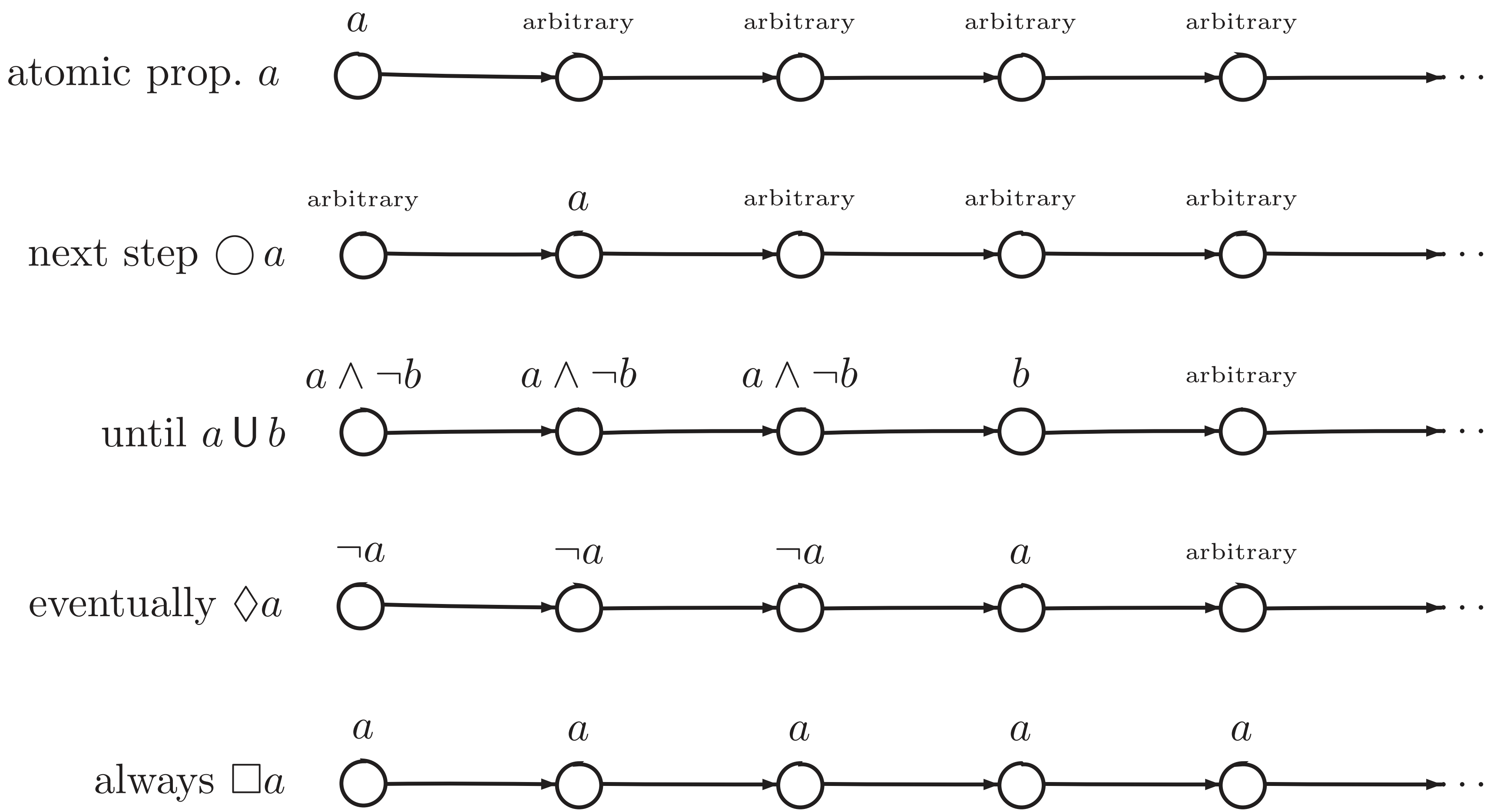
[BK08] Kapitel 5

Grundlagen: Lineare Temporale Logik

- Temporale Quantoren, Operatoren
  - $\bigcirc$ Next
  - $\Diamond$ Diamond
  - $\Box$ Box
  - $\leadsto$ Leads-to
  - weitere bei Bedarf $\mathsf{U}, \bigcirc^k, \Diamond^{\leq k}, \bigcirc^{-1}$
- Auswertung von Formeln: $\sigma \models F$
- hier nur informell:

[BK08] Kapitel 5

Grundlagen: Lineare Temporale Logik



aus [BK08] Kapitel 5

Grundlagen: Lineare Temporale Logik

Zusammensetzung:

$$\begin{aligned} \Box\Diamond F &\quad \text{infinitely often} \\ \Diamond\Box F &\quad \text{eventually forever} \\ F \leadsto G &\Leftrightarrow \Box(F \Rightarrow \Diamond G) \end{aligned}$$

Grundlagen: Lineare Temporale Logik

Eine Verkehrsampel hat die Phasen *grün*, *rot* und *gelb*. Dabei soll die Ampel folgende Zustandsfolge einhalten:

$$rot \longrightarrow gelb \longrightarrow gr\ddot{u}n \longrightarrow gelb \longrightarrow rot \longrightarrow \ldots$$

$$\Box\Diamond gr\ddot{u}n$$

$$\Box(rot \Rightarrow \neg \bigcirc gr\ddot{u}n)$$

$$\Box(rot \Rightarrow (rot\,\mathsf{U}(gelb \wedge \bigcirc(gelb\,\mathsf{U}gr\ddot{u}n)))).$$

(vgl. [BK08] Kapitel 5

TLA$^+$- *Temporal Logic for Actions*

Grundlagen: TLA$^+$

- Erweiterung der LTL, PL
- *built-in* Prädikatenlogik, Mengen, Funktionen
- Spezifikation von Aktionen: Übergang zw. Zustände
- ungetypte Variablen

Beispiel: DieHard

- Bestandsaufnahme
  - Objekte (Womit wird gearbeitet?)
  - Aktionen (Was darf gemacht werden?)
  - Ziel (Endzustand)

On the fountain their should be two jars - five gallon and three gallon.
Fill one of the jags with four gallons.

DieHard

- Zwei Gefäße
  - zwei Variablen $smallJar, bigJar$
  - begrenztes Fassungsvermögen
  - $smallJar \in \{1, 2, 3\}$
  - $bigJar \in \{1, 2, 3, 4, 5\}$
- Aktionen:
  - Entleeren
  - Umfüllen
  - Auffüllen
- Ziel: $bigJar = 4$

Spezfikation DieHard.tla

Zusammenfassung

- EXTENDS - Einbinden von Modulen
- VARIABLES - Variablen deklaration
- Deklaration von Operatoren ($\triangleq$)
- *Prime*-Variablen (Variablen des nächsten Zustandes
- IF THEN ELSE
- TLA$^+$-Normalform

TLA$^+$-Normalform: $Init \land \Box[Next]_v$

$$[Next]_v \stackrel{\text{def}}{=} Next \lor (v' = v) \tag{1}$$

- $Next$ - Disjunktion von Aktionen
- Einführung von *suttering steps*
- Schritte ohne Änderung
- Wichtig für Wiederverwendung

Liveness and Fairness

- **Bisher:** Vorschrift der erlaubten Aktionen **(Safety-Properties)**
- *Stuttering steps* erlauben „Nichts-Tuen“
- Liveness: Das System muss etwas tun!
- Ausschluss von Deadlocks
- Alle Aktionen werden ausgeführt (Fairness)

Liveness and Fairness

Unconditional Fairness Jeder Teilnehmer bekommt seinen Zug unendlich oft.

Strong Fairness Jeder Prozess, der unendlich oft aktiviert wird, bekommt seinen Zug unendlich oft.

Weak Fairness Jeder Prozess, der für eine bestimmte Zeitspanne aktiviert ist, bekommt seinen Zug unendlich oft.

- Weak Fairness ist der Normalfall
- Weitere Einschränkungen möglich

Beispiel (Ampelanlage): Aktion $rot \to grün$

Unconditional Fairness Jeder Ampel wird unendlich oft grün.
Was passiert, wenn kein Auto auf der Bahn steht?

Strong Fairness Ampel muss grün werden, wenn ein Auto für **einen** Zustand Farbahn steht.
Was passiert bei Messfehler?

Weak Fairness Ampel wird grün, wenn ein Auto für eine **bestimmte** Zeit auf Farbahn steht.

TLA$^+$-Umsetzung:

- *built-in:*

$$
\begin{aligned}
WF_{vars}(A) &\stackrel{\text{def}}{=} \Diamond\Box(\text{ENABLED } \langle A\rangle_{vars}) \Rightarrow \Box\Diamond\langle A\rangle_{vars} \\
&\equiv \neg\Diamond\Box(\text{ENABLED } \langle A\rangle_{vars}) \vee \Box\Diamond\langle A\rangle_{vars} \\
SF_{vars}(A) &\stackrel{\text{def}}{=} \Box\Diamond(\text{ENABLED } \langle A\rangle_{vars}) \Rightarrow \Box\Diamond\langle A\rangle_{vars} \\
&\equiv \neg\Box\Diamond(\text{ENABLED } \langle A\rangle_{vars}) \vee \Box\Diamond\langle A\rangle_{vars}
\end{aligned}
$$

- Liveness: $\Box\Diamond\langle A\rangle_{vars}$
- ENABLED - Aktion ausführbar
- $\langle A\rangle_{vars} \stackrel{\text{def}}{=} A \wedge v' \neq v$

PlusCal

eventuell

TLC - Modelchecker

Modelchecker

- *Ausführbarkeit* von Spezifikation
  - Aufzählbare, endliche Mengen
  - gebundene Variablen
  - Unterstütze Datentypen
  - Vergleichbarkeit von Werten
- Endlichkeit der Zustandsmenge

$$\begin{array}{rrr}
\exists x : p & \textsc{choose}\ x : p & \\
\exists x \in S : p & \forall x \in S : p & \textsc{choose}\ x \in S : p \\
\{x \in S : p\} & \{e : x \in S\} & [x \in S \mapsto e] \\
\textsc{subset}\ S & \textsc{union}\ S &
\end{array}$$

Modelchecking Mode
Idee: Erzeugen aller erreichbar Zustände

- Erzeugen von Verhalten
- Zustandsgraph

Zustandserzeugnis

- Auswertung von $\mathit{Next} \equiv A_1 \lor A_2 \lor \cdots$
- parallele Ausführung der Disjunktionsterme A_i
- Variablen erhalten Zustandswerte
- *prime*-Variablen $x' = e$ sind TRUE wenn x' nicht gesetzt
- $x' \in S$?
- Breitensuche
- Graph $\mathcal{G}$
- Warteschlange $\mathcal{U}$

TLC starten den folgenden Algorithmus mit leeren $\mathcal{G}$ und $\mathcal{U}$:

1. Überprüfe Annahmen (ASSUME)
2. Berechne initialen Zustände *Init* berechnet. Jeden Zustand s überprüfen,
   (a) ob die Invarianten eingehalten werden; sonst Abbruch.
   (b) ob das *state constraint* eingehalten wird (wenn TRUElege s in $\mathcal{U}$ ab)

3. Solange $\mathcal{U}$ ist nicht leer
   (a) Entnehme den ersten Zustand s aus $\mathcal{U}$.
   (b) Berechne die Menge T der Nachfolgezustände
   (c) $T = \emptyset \Rightarrow$ Deadlock.
   (d) Für jeden Zustand $t \in T$:
      i. Erfüllung der Invarianten in t
      ii. Erfüllung der *state constraint* und *action constraint*
         A. nehme t und (t, t) in $\mathcal{G}$ $(\mathcal{U})$ auf, falls es nicht vorhanden ist.
         B. nehme (s, t) in $\mathcal{G}$ auf.

Einstellungen im Modelchecker

Weiteres

- TLA^+-Syntax und Semantik (Funktionen, Module, ...)
- TLA^+-Modulwiederverwendung, $\exists\!\!\exists$, $\forall\!\!\forall$
- Echtzeit in TLA^+
- TLA-Toolset (TLPS, PlusCal, TLATEX, SANY)
- LTL (Formelle Definition, Positive Normalform, weitere Operatoren)
- ...

Fazit

- Formelle Spezifikation
- Lineare Temporale Logik
  - Logik über Zustandsabfolge
  - Temporale Quantoren
- Temporale Logic for Actions
- Modelchecker
  - Grenzen
  - Algorithmus
  - Anwendung

Vielen Dank für ihre Aufmerksamkeit

Quellen, Dokumente und Spezifikationen

`http://fh-trier.de/~weigla/tla`

Literatur

[BK08] Baier, Christel und Joost-Pieter Katoen: *Principles of Model Checking*. MIT Press, 2008.

[Lam02] Lamport, Leslie: *Specifying Systems*. 1 Auflage, June 2002.