

Formale Methoden und Werkzeuge

Vorlesung

Wintersemester 2024–26

Johannes Waldmann, HTWK Leipzig

19. Oktober 2026

Einleitung

Formale Methoden und Werkzeuge ...

- ... zur Spezifikation, Analyse (Verifikation), Synthese von Hard- und Softwaresystemen.
 - für (geplantes) System S (Bsp: CPU, Programm) *formales Modell* M angeben (Bsp: logische Schaltung, endl. Automat)
 - gewünschte System-Eigenschaft E angeben (Bsp: Boolesche Formel, reguläre Sprache)
 - Analyse: mit Werkzeug(unterstützung) feststellen, ob das Modell die Eigenschaft hat (Bsp: $M \Rightarrow E$, $M \subseteq E$)
 - Synthese (*Constraint-Programmierung*): Werkzeug konstruiert aus Constraint E ein passendes Modell M

Die Rolle der Abstraktion

- Schritt von System S zu Modell M ist *Abstraktion* (es werden Details ignoriert), das ist
 - teils *nützlich* (verbessert die Übersicht, Einsicht),
 - teils *notwendig* (nur wenn $M \subseteq E$ entscheidbar oder M aus E berechenbar, kann überhaupt ein Werkzeug zur Analyse bzw. Synthese implementiert werden)
- in den einfachen Beispielen (besonders am Anfang der VL) beginnen wir die Betrachtung bereits bei M (damit wir sehen, welche Theorien angewendet werden sollen)
- das Abstrahieren muß aber auch geübt werden, es ist aber eine anwendungsspezifische Kunst

Industrielle Anwendungen von FMW

- Verifikation von Schaltkreisen
(*bevor* man diese tatsächlich produziert)
 $F = \text{S-Implementierung}(x) \neq \text{S-Spezifikation}(x)$
wenn F unerfüllbar ($\neg \exists x. F$), dann ist Impl. korrekt
- Verifikation von Software durch *model checking*:
Programmzustände abstrahieren durch
Zustandsprädikate, Programmabläufe durch endliche
Automaten.
z. B. Thomas Ball et al. 2004: *Static Driver Verifier*
<https://www.microsoft.com/en-us/research/project/slam/publications/>
benutzt Constraint-Solver Z3 (Nikolaj Björner et al.,
2007–) <https://github.com/Z3Prover/z3/wiki>

Industrielle Anwendungen von FMW

- automatische Analyse des Ressourcenverbrauchs von Programmen
 - Termination (jede Rechnung hält)
 - Komplexität (... nach $O(n^2)$ Schritten)
- mittels *Bewertungen* von Programmzuständen:
 - $W : \text{Zustandsmenge} \rightarrow \mathbb{N}$
 - wenn $z_1 \rightarrow z_2$, dann $W(z_1) > W(z_2)$.
- Parameter der Bewertung werden durch Constraint-System beschrieben.
- vgl. Carsten Fuhs: *Automated Termination Analysis...*, Intl. School on Rewriting, 2022 <https://viam.science.tsu.ge/clas2022/isr/termination.html>

Anwendung: Polynom-Interpretationen

- Berechnungsmodell: Wortersetzung ($\approx$ Turingmaschine)
- Programm: $ab \rightarrow ba$ ($\approx$ Bubble-Sort)

Beispiel-Rechnung: $abab \rightarrow baab \rightarrow baba \rightarrow bbaa$

- Bewertung W durch *Interpretation*: lineare Funktionen
 $f_a(x) = Px + Q, f_b(x) = Rx + S$ mit $P, Q, R, S \in \mathbb{N}$

$$W(abab) = f_a(f_b(f_a(f_b(0)))) \dots$$

- Interp. ist monoton: $x > y \Rightarrow f_a(x) > f_a(y) \wedge f_b(x) > f_b(y)$
- Interp. ist kompatibel mit Programm: $f_a(f_b(x)) > f_b(f_a(x))$
- resultierendes Constraint-System für P, Q, R, S ,
- Lösung mittels Z3

Constraint-Programmierung—Beispiel

```
(set-logic QF_NIA) (set-option :produce-models true)
(declare-fun P () Int) (declare-fun Q () Int)
(declare-fun R () Int) (declare-fun S () Int)
(assert (and (< 0 P) (<= 0 Q) (< 0 R) (<= 0 S)))
(assert (> (+ (* P S) Q) (+ (* R Q) S)))
(check-sat) (get-value (P Q R S))
```

- *Constraint-System*: eine prädikatenlogische Formel F
 $0 < P \wedge \dots \wedge P \cdot S + Q > R \times Q + S$
- *Lösung*: Interpretation (Var.-Belegung), für die F wahr ist
- CP ist eine Form der *deklarativen* Programmierung.
- *Vorteil*: Benutzung von allgemeinen Suchverfahren
(bereichs-, aber nicht anwendungsspezifisch).

Constraints in der Unterhaltungsmathematik

- Nikoli (1980–, „the first puzzle magazine in Japan.“)
<https://www.nikoli.co.jp/en/puzzles/>
- Erich Friedman: *Math Magic* 1998–2020
<https://erich-friedman.github.io/mathmagic/>
- Simon Tatham’s Portable Puzzle Collection <https://www.chiark.greenend.org.uk/~sgtatham/puzzles/>
- Angela und Otto Janko: <http://www.janko.at/Raetsel/> ,
- Donald Knuth: *A Potpourri of Puzzles*, 2022,
TAOCP, Band 4, Pre-Faszikel 9b,
<https://cs.stanford.edu/~knuth/taocp.html>,
<https://cs.stanford.edu/~knuth/fasc9b.ps.gz>

Wettbewerbe für Constraint-Solver

- für aussagenlogische Formeln:

<http://www.satcompetition.org/>

(SAT = satisfiability)

- für prädikatenlogische Formeln

<https://smt-comp.github.io/>

(SMT = satisfiability modulo theories)

Theorien: $\mathbb{Z}$ mit $\leq$, Plus, Mal; $\mathbb{R}$ mit $\leq$, Plus; ...

- Termination und Komplexität

[https://www.termination-portal.org/wiki/](https://www.termination-portal.org/wiki/Termination_Competition)

Termination_Competition

Gliederung der Vorlesung

- Aussagenlogik
 - zur Modellierung (SAT-Kodierung) von Systemen mit endlichem Zustandsraum, begrenzter Schritt-Zahl (bounded model checking)
 - Werkzeuge: SAT-Kompiler (Tseitin-Transformation), SAT-Solver (Propagation, Backtracking, Lernen)
- eingeschränkte Prädikatenlogik
 - Unbekannte aus unendlichen Bereichen (Zahlen)
 - Werkzeuge: SMT-Solver (SAT modulo Theory)
- allgemeine Beweis-Systeme
 - zur Modellierung beliebiger (unbeschränkter) Systeme
 - Methode: Typisierung (Curry-Howard-Isomorphie, program = proof), Werkzeuge: interaktive Beweiser

Organisatorisches

- jede Woche 1 Vorlesung + 1 Übung
 - Hausaufgaben (*Haus* bedeutet: zuhause bearbeiten, in der Übung diskutieren)
 - Aufgaben im Skript
 - Aufgaben in `autotool`
- Prüfungszulassung: regelmäßiges und erfolgreiches Bearbeiten von Hausaufgaben
- Klausur (2 h, keine Hilfsmittel).
 - Quelltexte, Planung und Diskussion der Übungsaufgaben
- `https://gitlab.dit.htwk-leipzig.de/johannes.waldmann/fmw-ws24` (Projekt-Mitgliedschaft beantragen, Zugang wird dann auf Mitglieder eingeschränkt)

Literatur

- Cerone, A., Roggenbach, M., Schlingloff, B.-H., Schneider, G., Shaikh, S.A.: *Teaching formal methods for software engineering - ten principles*, Informatica Didactica, Nr. 9, (2015). <https://www.informaticadidactica.de/uploads/Artikel/Schlinghoff2015/Schlinghoff2015.pdf>
- Uwe Schöning, Jacob Toran: *Das Erfüllbarkeitsproblem SAT*, Lehmanns (2012)
- Christel Baier, Joost Katoen: *Principles of Model Checking*. MIT Press, Cambridge (2008)
- Samuel Mimram: *Program = Proof*, <https://www.lix.polytechnique.fr/Labo/Samuel.Mimram/teaching/INF551/course.pdf>

Hausaufgaben

WS 25: 1b, 5, 4a, 2.

1. Pierluigi Crescenzi, Viggo Kann *A compendium of NP optimization problems*

<https://web.archive.org/web/20200227195925/http://www.nada.kth.se/~viggo/problemlist/compendium.html>

Beispiele:

- (a) Minimum File Transfer Scheduling (node195).
- (b) Minimum Dynamic Storage Allocation (node163).

Erläutern Sie die Spezifikation an einem Beispiel. Geben Sie eine lösbare Instanz sowie dafür zwei Lösungen mit unterschiedlichem Maß an.

2. Aufgabe: formalisieren Sie *Math Magic* Februar 2007.

Was ist dabei für Springer und König einfacher als für Dame, Läufer, Turm?

(allgemein für Math Magic: offene Fragen <https://erich-friedman.github.io/mathmagic/unsolved.html>)

3. Aufgabe: formalisieren Sie Wolkenkratzer oder Towers (was ist der Unterschied?)

<https://www.janko.at/Raetsel/Wolkenkratzer>

<https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/towers.html>

- $B = \{0, 1, \dots, n - 1\}$, Unbekannte: $h_{x,y} \in B$ für $x, y \in B$
- Constraint für eine Zeile x : (entspr. für Spalte)

$\bigvee_{p \in \text{Permutationen}(0, \dots, n-1), p \text{ kompatibel mit Vorgaben}} \bigwedge_{y \in \{0, \dots, n-1\}} (h_{x,y} = p(y))$

Bsp: $n = 4$, Vorgabe links 2, rechts 1, kompatibel sind
 $[0, 2, 1, 3], [2, 0, 1, 3], [2, 1, 0, 3], [2, 1, 0, 3]$.

- diese Formel wird exponentiell groß (wg. Anzahl Permutationen),

Folge-Aufgabe: *geht das auch polynomiell?*

- Geben Sie eine Aufgabenstellung der Größe $w \times w$ an mit $4 \cdot w$ Vorgaben (d.h., *alle* Vorgaben), die mehr als eine Lösung hat.

... für $w = 4$, für größere w (einige, alle)

- (offene Frage?) Geben Sie eine eindeutig lösbare Instanz mit möglichst wenigen Vorgaben an. (Sind $w - 1$ Vorgaben für $w \times w$ immer möglich?)

4. Modellierung von Puzzles (aus Tatham-Collection)

- (a) geben Sie ein Modell an für *Pegs* (Solitaire). Hinweise:
- Zustand = Boolesche Matrix,
 - Schritt = Relation zwischen Matrizen,
 - Lösung = Schrittfolge ($\Rightarrow$ Zustandsfolge). Wieviele Schritte?
- (b) Modell für *Unruly*. (keine Zustandsfolge, sondern direkt die Lösung. Welches sind die Unbekannten, welches sind ihre Beziehungen, untereinander und zur Vorgabe)
- (c) modellieren Sie *Untangle*

Vergleichen Sie mit den tatsächlichen Quelltexten

<https://git.tartarus.org/?p=simon/puzzles.git>

5. Constraint für monotone kompatible Bewertungsfunktion:
- lösen Sie mit Z3 (ist im Pool installiert, vgl. <https://www.imn.htwk-leipzig.de/~waldmann/etc/pool/>)
 - eine *kleinste* Lösung finden (Summe von P, Q, R, S möglichst klein) — dafür Assert(s) hinzufügen.
 - Abstieg der so gefundenen Bewertungsfunktion nachrechnen für $abab \rightarrow baab \rightarrow baba \rightarrow bbaa$
 - gibt diese Bewertungsfunktion die maximale Schrittzahl genau wieder? (nein)
 - Folge-Aufgabe: entspr. Constraint-System für Bewertungsfunktion für $ab \rightarrow bba$ aufstellen und lösen.