Constraint-Programmierung
Vorlesung
Sommersemester 2009, 2012, 2015, WS 2016

Johannes Waldmann, HTWK Leipzig

13. Oktober 2016

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
- *Lösung* = Modell von F (= Strukur M, in der F wahr ist)
- CP ist eine Form der *deklarativen* Programmierung.
- *Vorteil*: Benutzung von allgemeinen Suchverfahren (bereichs-, aber nicht anwendungsspezifisch).

Industrielle Anwendungen der CP

- Verifikation von Schaltkreisen
  (*bevor* man diese tatsächlich produziert)
  $F =$ S-Implementierung$(x) \neq$ S-Spezifikation(x)
  wenn F unerfüllbar $(\neg\exists x)$, dann Implementierung korrekt
- Verifikation von Software durch *model checking*:
  Programmzustände abstrahieren durch Zustandsprädikate,
  Programmabläufe durch endliche Automaten.
  z. B. Static Driver Verifier `http://research.microsoft.com/en-us/projects/slam/`
  benutzt Constraint-Solver Z3 `http://research.microsoft.com/en-us/um/redmond/projects/z3/`

Industrielle Anwendungen der CP

automatische Analyse des Resourcenverbrauchs von Programmen

- Termination (jede Rechnung hält)
- Komplexität (...nach $O(n^2)$ Schritten)

mittels *Bewertungen* von Programmzuständen:

- W : 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.

CP-Anwendung: Polynom-Interpretationen

- Berechnungsmodell: Wortersetzung ($\approx$ Turingmaschine)
- Programm: $ab \to ba$ ($\approx$ Bubble-Sort)
  Beispiel-Rechnung: $abab \to baab \to baba \to bbaa$
- Bewertung W durch lineare Funktionen
  $f_a(x) = Px + Q, f_b(x) = Rx + S$ mit $P, Q, R, S \in \mathbb{N}$
  $W(abab) = f_a(f_b(f_a(f_b(0)))), \ldots$
- monoton: $x > y \Rightarrow f_a(x) > f_a(y) \wedge f_b(x) > f_b(y)$
- kompatibel mit Programm: $f_a(f_b(x) > f_b(f_a(x))$
- resultierendes Constraint-System für P, Q, R, S,
- Lösung mittels Z3

Wettbewerbe für Constraint-Solver

- für aussagenlogische Formeln:
  `http://www.satcompetition.org/`
  (SAT = satisfiability)
- für prädikatenlogische Formeln
  `http://smtcomp.sourceforge.net/`
  (SMT = satisfiability modulo theories)
  Theorien: $\mathbb{Z}$ mit $\leq$, Plus, Mal; $\mathbb{R}$ mit $\leq$, Plus; ...
- Termination und Komplexität
  `http://www.termination-portal.org/wiki/Termination_Competition`

Gliederung der Vorlesung

- Aussagenlogik
  - CNF-SAT-Constraints (Normalf., Tseitin-Transformation)
  - DPLL-Solver, Backtracking und Lernen
  - ROBDDs (Entscheidungsdiagramme)
- Prädikatenlogik (konjunktive Constraints)
  - Finite-Domain-Constraints
  - naive Lösungsverfahren, Konsistenzbegriffe
  - lineare Gleichungen, Ungleichungen, Polynomgleichungen
  - Termgleichungen, Unifikation
- Kombinationen
  - Kodierungen nach CNF-SAT (FD, Zahlen)
  - SMT, DPLL(T)

Organisatorisches

- jede Woche 1 Vorlesung + 1 Übung
- Übungsaufgaben
  - „schriftlich“, d.h. Aufgabe im Skript (Folie), Diskussion in Übung an der Tafel
  - online, d.h. Aufgabe in ‖autotool—, Bearbeitung selbständig

  Prüfungszulassung: 50 Prozent der autotool-Pflichtaufgaben
- Klausur (2 h, keine Hilfsmittel)

Literatur

- Krzysztof Apt: *Principles of Constraint Programming*, `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521825832`
- Daniel Kroening, Ofer Strichman: *Decision Procedures*, Springer 2008. `http://www.decision-procedures.org/`
- Petra Hofstedt, Armin Wolf: Einführung in die Constraint-Programmierung, Springer 2007. `http://www.springerlink.com/content/978-3-540-23184-4/`
- Uwe Schöning: Logik für Informatiker, Spektrum Akad. Verlag, 2000.

Ausblick

ich betreue gern
Masterprojekte/-Arbeiten zur Constraint-Programmierung, z.B.

- Tests/Verbesserung für
  `https://github.com/ekmett/ersatz`
- Ersatz-ähnliche Schnittstelle für
  `https://github.com/adamwalker/haskell_cudd`
- autotool-Aufgaben zu CP, vgl. `https://gitlab.imn.htwk-leipzig.de/autotool/all/tree/master/collection/src/DPLLT`
- Anwendungen (auch: nichtlineare bzw. diskrete Optimierung, data mining)

Übung KW41 (Aufgaben)

- Constraint-Optimierungsprobleme: https://www.nada.kth.se/~viggo/problemlist/compendium.html
  Beispiel: https://www.nada.kth.se/~viggo/wwwcompendium/node195.html
- Beispiele für Constraints aus der Unterhaltungsmathematik:
  http://www.janko.at/Raetsel/,
  http://www.nikoli.co.jp/en/puzzles/
- formales Modell für
  - http://www.janko.at/Raetsel/Sternenhaufen/
  - http://www.janko.at/Raetsel/Wolkenkratzer/
- Constraint-Solver im Pool ausprobieren (Z3, minisat)
  allgemeine Hinweise: http://www.imn.htwk-leipzig.de/~waldmann/etc/pool/ http://www.imn.htwk-leipzig.de/~waldmann/etc/beam/
- autotool: einschreiben und ausprobieren
  https://autotool.imn.htwk-leipzig.de/new/vorlesung/234/aufgaben

Übung KW41 (Diskussion)

Constraint-System für Hochhaus-Rätsel:

- Unbekannte: $h_{x,y} \in \{0, \ldots, n-1\}$ für $x, y \in \{0, \ldots, n-1\}$
- Constraint für eine Zeile x:
  $\bigvee_{p \in \text{Permutationen}(0,\ldots,n-1), p \text{kompatibel mit Vorgaben}} \bigwedge_{y \in \{0,\ldots,n-1\}} (h_{x,y} = p(y))$
  Bsp: $n = 4$, Vorgabe links 2, rechts 1, kompatibel sind $[0,2,1,3], [2,0,1,3], [2,1,0,3], [2,1,0,3]$.
  entspr. für Spalten
- diese Formel wird exponentiell groß (wg. Anzahl Permutationen),
  Folge-Aufgabe: *geht das auch polynomiell*?

Constraint für monotone kompatible Bewertungsfunktion:

- mit Z3 lösen
- eine kleinste Lösung finden (Summe von P, Q, R, S möglichst klein) — dafür Assert(s) hinzufügen.
- Abstieg der so gefundenen Bewertungsfunktion nachrechnen für $abab \to baab \to baba \to bbaa$
- gibt diese Bewertungsfunktion die maximale Schrittzahl