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

Johannes Waldmann, HTWK Leipzig

6. Mai 2019

1 Einleitung

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 $\to \mathbb{N}$
- wenn $z_1 \to z_2$, 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 genau wieder? (nein)
- Folge-Aufgabe: entspr. Constraint-System für Bewertungsfunktion für $ab \to bba$ aufstellen und lösen.

2 Erfüllbarkeit aussagenlogischer Formeln (SAT)

Aussagenlogik: Syntax

aussagenlogische Formel:

- elementar: Variable $v_1, \ldots$
- zusammengesetzt: durch Operatoren
  - einstellig: Negation
  - zweistellig: Konjunktion, Disjunktion, Implikation, Äquivalenz

Aussagenlogik: Semantik

- Wertebereich $\mathbb{B} = \{0, 1\}$, Halbring $(\mathbb{B}, \vee, \wedge, 0, 1)$
  Übung: weitere Halbringe mit 2 Elementen?
- *Belegung* ist Abbildung $b : V \to \mathbb{B}$
- *Wert* einer Formel F unter Belegung b: $\mathrm{val}(F, b)$
- wenn $\mathrm{val}(F, b) = 1$, dann ist b ein *Modell* von F, Schreibweise: $b \models F$
- *Modellmenge* $\mathrm{Mod}(F) = \{b \mid b \models F\}$
- F *erfüllbar*, wenn $\mathrm{Mod}(F) \neq \emptyset$
- Modellmenge einer *Formelmenge*: $\mathrm{Mod}(M) = \{b \mid \forall F \in M : b \models F\}$

Modellierung durch SAT: Ramsey

gesucht ist Kanten-2-Färbung des K_5 ohne einfarbigen K_3.

- Aussagenvariablen $f_{i,j}$ = Kante (i, j) ist rot (sonst blau).
- Constraints:

  $\forall p : \forall q : \forall r : (p < q \wedge q < r) \Rightarrow ((f_{p,q} \vee f_{q,r} \vee f_{p,r}) \wedge \dots)$

das ist ein Beispiel für ein Ramsey-Problem

(F. P. Ramsey, 1903–1930) `http://www-groups.dcs.st-and.ac.uk/~history/Biographies/Ramsey.html`

diese sind schwer, z. B. ist bis heute unbekannt: gibt es eine Kanten-2-Färbung des K_{43} ohne einfarbigen K_5?

`http://www1.combinatorics.org/Surveys/ds1/sur.pdf`

Programmbeispiel zu Ramsey

Quelltext in `Ramsey.hs`

```
num p q = 10 * p + q ; n x = negate x
f = do
  p <- [1..5] ; q <- [p+1 .. 5] ; r <- [q+1 .. 5]
  [    [ num p q, num q r, num p r, 0 ]
     , [ n $ num p q, n $ num q r, n $ num p r, 0 ] ]
main = putStrLn $ unlines $ do
  cl <- f ; return $ unwords $ map show cl
```

Ausführen:

```
runghc Ramsey.hs | minisat /dev/stdin /dev/stdout
```

Benutzung von SAT-Solvern

Eingabeformat: SAT-Problem in CNF:

- Variable = positive natürliche Zahl
- Literal = ganze Zahl ($\neq 0$, mit Vorzeichen)
- Klausel = Zeile, abgeschlossen durch 0.
- Programm = Header `p cnf <#Variablen> <#Klauseln>`, dann Klauseln

Beispiel

```
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

Löser: `minisat input.cnf output.text`

Modellierung durch SAT: N Damen

stelle möglichst viele Damen auf $N \times N$-Schachbrett, die sich nicht gegenseitig bedrohen.

- Unbekannte: $q_{x,y}$ für $(x, y) \in F = \{1, \ldots, N\}^2$
  mit Bedeutung: $q_{x,y} \iff$ Feld (x, y) ist belegt
- Constraints: $\bigwedge_{a,b \in F, a \text{ bedroht } b} \neg p_a \vee \neg p_b$.
- „möglichst viele“ läßt sich hier vereinfachen zu:
  „in jeder Zeile genau eine“. (Constraints?)

Normalformen (DNF, CNF)

Definitionen:

- Variable: $v_1, \ldots$
- Literal: v oder $\neg v$
- DNF-Klausel: Konjunktion von Literalen
- DNF-Formel: Disjunktion von DNF-Klauseln
- CNF-Klausel: Disjunktion von Literalen
- CNF-Formel: Konjunktion von CNF-Klauseln

Disjunktion als Implikation: diese Formeln sind äquivalent:

- $(x_1 \wedge \ldots \wedge x_m) \rightarrow (y_1 \vee \ldots \vee y_n)$
- $(\neg x_1 \vee \ldots \vee \neg x_m \vee y_1 \vee \ldots \vee y_n)$

Äquivalenzen

Def: Formeln F und G heißen *äquivalent*, wenn $\mathrm{Mod}(F) = \mathrm{Mod}(G)$.
Satz: zu jeder Formel F existiert äquivalente Formel G in DNF.
Satz: zu jeder Formel F existiert äquivalente Formel G' in CNF.

aber ... wie groß sind diese Normalformen?

Erfüllbarkeits-Äquivalenz

Def: F und G *erfüllbarkeitsäquivalent*, wenn $\mathrm{Mod}(F) \neq \emptyset \iff \mathrm{Mod}(G) \neq \emptyset$.
Satz: es gibt einen Polynomialzeit-Algorithmus, der zu jeder Formel F eine erfüllbarkeitsäquivalente CNF-Formel G berechnet.

(Zeit $\geq$ Platz, also auch $|G| = \mathrm{Poly}(|F|)$)

Beweis (folgt): Tseitin-Transformation

Tseitin-Transformation

Gegeben F, gesucht erfüllbarkeitsäquivalentes G in CNF.
Berechne G mit $\mathrm{Var}(F) \subseteq \mathrm{Var}(G)$ und $\forall b : b \models F \iff \exists b' : b \subseteq b' \wedge b' \models G$.
Plan:

- für jeden nicht-Blatt-Teilbaum T des Syntaxbaumes von F eine zusätzliche Variable n_T einführen,
- wobei gelten soll: $\forall b' : \mathrm{val}(n_T, b') = \mathrm{val}(T, b)$.

Realisierung:

- falls (Bsp.) $T = L \vee R$, dann $n_T \leftrightarrow (n_L \vee n_R)$ als CNF-Constraintsystem
- jedes solche System hat ≤ 8 Klauseln mit 3 Literalen, es sind ingesamt $|F|$ solche Systeme.

Tseitin-Transformation (Übung)

Übungen (Hausaufgabe, ggf. autotool):
für diese Formeln:

- $(x_1 \leftrightarrow x_2) \leftrightarrow (x_3 \leftrightarrow x_4)$
- Halb-Adder (2 Eingänge x, y, 2 Ausgänge r, c)
  $(r \leftrightarrow (\neg(x \leftrightarrow y))) \wedge (c \leftrightarrow (x \wedge y))$

jeweils:

- führe die Tseitin-Transformation durch
- gibt es eine kleinere erfüllbarkeitsäquivalente CNF (deren Modelle Erweiterungen der Original-Modelle sind)

Aufgaben zur SAT-Modellierung

- Rösselsprung (= Hamiltonkreis)
- Norinori `http://nikoli.com/en/puzzles/norinori/`
- ABCEndView `http://www.janko.at/Raetsel/AbcEndView/`

Vorgehen bei Modellierung:

- welches sind die Unbekannten, was ist deren Bedeutung?
  (Wie rekonstruiert man eine Lösung aus der Belegung, die der Solver liefert?)
- welches sind die Constraints? wie stellt man sie in CNF dar?

Formulierung von SAT-Problemen mit Ersatz
`http://hackage.haskell.org/package/ersatz`, Edward Kmett,

```
import Prelude hiding ((&&),(||),not )
import Ersatz
solveWith minisat $ do
  p <- exists ; q <- exists
  assert $ p && not q
  return [p,q::Bit]
```

Unbekannte erzeugen (`exists`), Formel konstruieren (`&&`,...), assertieren, lösen
zu Implementierung vgl. auch `http://www.imn.htwk-leipzig.de/~waldmann/etc/untutorial/ersatz/`

ABC End View
mögliche Kodierung:
$v(x, y, z) \iff$ an Position (x, y) steht Zeichen z,
wobei $0 =$leer, $1 = a, 2 = b, 3 = c$

```
v <- replicateM 4 $ replicateM 4
   $ replicateM 4 exists
forM (entries v) $ \ e -> assert $ exactly_one e
forM (rows v) $ \ r ->    assert $ all_different r
forM (cols v) $ \ c ->    assert $ all_different c
assert $ see c $ left 1 v ; assert $ see b $ top 0 v
```

SAT-Kodierung: Hamiltonkreis

- Def: Graph $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$,

  Permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ bestimmt Hamiltonkreis, wenn und $v_{\pi(1)}v_{\pi(2)} \in E, \ldots, v_{\pi(n-1)}v_{\pi(n)} \in E, v_{\pi(n)}v_{\pi(1)} \in E$.

- SAT-Kodierung: benutzt Variablen $p(i, j) \leftrightarrow \pi(i) = j$.

  Welche Constraints sind dafür nötig?

- Anwendung: Rösselsprung auf Schachbrett

Übungsaufgaben

zum Basteln:

- (noch von voriger Woche) `http://www.nikoli.com/en/puzzles/norinori/`
- OEIS (http://oeisf.org/Poster15a.pdf) Bestimme weitere Werte für `http://oeis.org/A250000`, `http://oeis.org/A227133`

  Kommentar: das ist im Moment noch schwierig, weil man dazu Anzahl-Constraints braucht und wir diese noch nicht behandelt haben.

schriftlich, empfohlen:

- Finde eine gute untere Schranke für die Größe einer äquivalenten CNF (d.h. *ohne* zusätzliche Variablen) für $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ (das XOR über n Eingängen).

  Ansatz: zeige eine untere Schranke für die Anzahl der Literale in jeder Klausel einer solchen CNF.

  (leicht) Finde eine gute obere Schranke für die Größe einer erfüllbarkeitsäquivalenten CNF (d.h. mit zusätzlichen Variablen und rekonstruierbarer Belegung) für diese Formel.

SAT-Kodierungen

- Q: welche Probleme sind (mit polynomiellem Aufwand) SAT-kodierbar?
- A: alle aus der Komplexitätsklasse NP

Beispiele:

- Independent Set (Schach: n-Damen-Problem)
- Vertex Cover (Variante n-Damen-Problem)
- Hamiltonkreis (Schach: Rösselsprung)

damit ist das Thema theoretisch komplett gelöst,
aber praktisch kommt es doch auf kunstvolle Kodierungen an, weitere Einzelheiten dazu später (Bit-Blasting für SMT)

Wiederholung: NP-Vollständigkeit

- nichtdeterministische Turingmaschine (NDTM)
  - Rechnung ist eine Baum,
  - jeder Knoten ist eine Konfiguration
    (Bandinhalt, Kopfzustand, Kopfposition),
  - jede Kante ist ein Rechenschritt
  - Rechnung ist erfolgreich, wenn in wenigstens einem Blatt der Zustand akzeptierend ist
- NP $:=$ die Sprachen, die sich in Polynomialzeit durch NDTM entscheiden lassen
- Reduktion $M \leq_P L \iff \exists f : \forall x : x \in M \iff f(x) \in L$ und f ist P-berechenbar
- L ist NP-vollständig: $L \in$ NP und $\forall M \in$ NP : $M \leq_P L$

Wiederholung: SAT ist NP-vollständig

SAT $\in$ NP ist klar (NDTM rät die Belegung)

Sei $M \in$ NP, zu zeigen $M \leq_P$ SAT. Gegeben ist also das Programm einer NDTM, die M in Polynomialzeit akzeptiert

- übersetze eine Eingabe x für diese Maschine in eine Formel $f(x)$ mit $x \in M \iff f(x) \in$ SAT:
- benutzt Unbekannte $C(t, p, a) : \iff$ zur Zeit t steht an Position p das Zeichen a.
- Klauseln legen fest:

  für $t = 0$ steht die Eingabe auf dem Band,

  $\forall t$: von t nach $t + 1$ richtig gerechnet (lt. Programm)

  schließlich akzeptierender Zustand erreicht

3 SAT-Solver

Überblick

Spezifikation:

- Eingabe: eine Formel in CNF
- Ausgabe:
  - eine erfüllende Belegung
  - *oder* ein Beweis für Nichterfüllbarkeit

Verfahren:

- evolutionär (Genotyp = Belegung)

- lokale Suche (Walksat)
- DPLL (Davis, Putnam, Logeman, Loveland)

Evolutionäre Algorithmen für SAT

- Genotyp: Bitfolge $[x_1, \ldots, x_n]$ fester Länge
- Phänotyp: Belegung $b = \{(v_1, x_1), \ldots, (v_n, x_n)\}$
- Fitness: z. B. Anzahl der von b erfüllten Klauseln
- Operatoren:
  - Mutation: einige Bits ändern
  - Kreuzung: one/two-point crossover?

Problem: starke Abhängigkeit von Variablenreihenfolge

Lokale Suche (GSat, Walksat)

Bart Selman, Cornell University, Henry Kautz, University of Washington
`http://www.cs.rochester.edu/u/kautz/walksat/`
Algorithmus:

- beginne mit zufälliger Belegung
- wiederhole: ändere das Bit, das die Fitness am stärksten erhöht

Problem: lokale Optima — Lösung: Mutationen.

DPLL

Davis, Putnam (1960), Logeman, Loveland (1962), `http://dx.doi.org/10.1145/321033.321034` `http://dx.doi.org/10.1145/368273.368557`
Zustand = partielle Belegung

- *Decide*: eine Variable belegen
- *Propagate*: alle Schlußfolgerungen ziehen
  Beispiel: Klausel $x_1 \vee x_3$, partielle Belegung $x_1 = 0$,
  Folgerung: $x_3 = 1$
- bei *Konflikt* (widersprüchliche Folgerungen)
  - (DPLL original) Backtrack (zu letztem Decide)
  - (DPLL mit CDCL) Backjump (zu früherem Decide)

DPLL-Begriffe

für partielle Belegung b (Bsp: $\{(x_1, 1), (x_3, 0)\}$): Klausel c ist

- *erfüllt*, falls $\exists l \in c : b(l) = 1$, Bsp: $(\neg x_1 \vee x_2 \vee \neg x_3)$
- *Konflikt*, falls $\forall l \in c : b(l) = 0$, Bsp: $(\neg x_1 \vee x_3)$
- *unit*, falls $\exists l \in c : b(l) = \bot \wedge \forall l' \in (c \setminus \{l\}) : b(l') = 0$,
  Bsp: $(\neg x_1 \vee \neg x_2 \vee x_3)$. Dabei ist $l = \neg x_2$ das Unit-Literal.
- *offen*, sonst. Bsp: $(x_2 \vee x_3 \vee x_4)$.

Eigenschaften: für CNF F und partielle Belegung b:

- wenn $\exists c \in F : c$ ist Konflikt für b, dann $\neg\exists b' \supseteq b$ mit $b' \models F$
  (d.h., die Suche kann dort abgebrochen werden)
- wenn $\exists c \in F$: c ist Unit für b mit Literal l, dann $\forall b' \supseteq b : b' \models F \Rightarrow b'(l) = 1$
  (d.h., l kann ohne Suche belegt werden)

DPLL-Algorithmus

Eingabe: CNF F, Ausgabe: Belegung b mit $b \models F$ oder UNSAT.
$\mathrm{DPLL}(b)$ (verwendet Keller für Entscheidungspunkte):

- (success) falls $b \models F$, dann halt (SAT), Ausgabe b.
- (backtrack) falls F eine b-Konfliktklausel enthält, dann:
  - falls Keller leer, dann halt (UNSAT)
  - sonst $v := pop()$ und $\mathrm{DPLL}(b_{<v} \cup \{(v, 1)\}$.

  dabei ist $b_{<v}$ die Belegung *vor* decide(v)
- (propagate) falls F eine b-Unitklausel c mit Unit-Literal l enthält: $\mathrm{DPLL}(b \cup \{(\text{variable}(l), \text{polarity}(l))\})$
- (decide) sonst wähle $v \notin \mathrm{dom}\, b$, push$(v)$, und $\mathrm{DPLL}(b \cup \{(v, 0)\})$.

DPLL: Eigenschaften

- Termination: DPLL hält auf jeder Eingabe
- Korrektheit: wenn DPLL mit SAT hält, dann $b \models F$.
- Vollständigkeit: wenn DPLL: UNSAT, dann $\neg\exists b : b \models F$

wird bewiesen durch Invariante

- $\forall b' : b' \in \mathrm{Mod}(F) \Rightarrow b \leq_{\text{lex}} b'$

  (wenn DPLL derzeit b betrachtet, und wenn F ein Modell b' besitzt, dann ist b' unterhalb oder rechts von b)

- dabei bedeutet: $b \leq_{\text{lex}} b'$:

  $b \subseteq b'$ oder $\exists v : b(v) = 0 \wedge (b_{<v} \cup \{(v, 1)\}) \subseteq b'$

Satz (Ü): für alle endlichen V: $<_{\text{lex}}$ ist eine wohlfundierte Relation auf der Menge der partiellen V-Belegungen:

DPLL-Beispiel

```
[[2,3],[3,5],[-3,-4],[2,-3,-4]
,[-3,4],[1,-2,-4,-5],[1,-2,4,-5]]
```

decide belegt immer die kleinste freie Variable, immer zunächst negativ

DPLL-Beispiel (Lösung)

```
[[2,3],[3,5],[-3,-4],[2,-3,-4]
,[-3,4],[1,-2,-4,-5],[1,-2,4,-5]]

[Dec (-1),Dec (-2),Prop 3,Prop (-4),Back
,Dec 2,Dec (-3),Prop 5,Prop (-4),Back
,Dec 3,Prop (-4),Back,Back,Back
,Dec 1,Dec (-2),Prop 3,Prop (-4),Back
,Dec 2,Dec (-3),Prop 5]
```

DPLL: Heuristiken, Modifikationen

- Wahl der nächsten Entscheidungsvariablen

  (am häufigsten in aktuellen Konflikten)

- Lernen von Konflikt-Klauseln (erlaubt Backjump)

- Vorverarbeitung (Variablen und Klauseln eliminieren)

alles vorbildlich implementiert und dokumentiert in Minisat `http://minisat.se/` (seit ca. 2005 sehr starker Solver)

Semantisches Folgern

- Def: eine Formel F folgt aus einer Formelmenge M, geschrieben $M \models F$, falls $\mathrm{Mod}(M) \subseteq \mathrm{Mod}(F)$.
- Bsp: $\{x_1 \vee \overline{x}_2, x_2 \vee x_3\} \models (x_1 \vee x_3)$, Beweise (lt. Def.) z.B. durch Vergleich der Wertetabellen (d.h., explizites Aufzählen der Modellmengen)

Eigenschaften (Übungsaufgaben):

- $M \models \mathsf{True}$
- $(M \models \mathsf{False}) \iff (\mathrm{Mod}(M) = \emptyset)$
- $(M \models F) \iff (\mathrm{Mod}(M \cup \{\neg F\}) = \emptyset)$
- wird bei CDCL benutzt: wir lernen nur Klauseln F, die aus der CNF (Klauselmenge) M folgen:
  $(M \models F) \iff (\mathrm{Mod}(M) = \mathrm{Mod}(M \cup \{F\}))$

DPLL mit CDCL (Plan)

conflict driven clause learning –
bei jedem Konflikt eine Klausel C hinzufügen, die

- aus der Formel folgt (d.h. Modellmenge nicht ändert)
- den Konflikt durch Propagation verhindert

Eigenschaften/Anwendung:

- danach *backjump* zur vorletzten Variable in K.

  (die letzte Variable wird dann propagiert, das ergibt die richtige Fortsetzung der Suche)
- K führt auch später zu Propagationen, d.h. Verkürzung der Suche

Der Konflikt-Graph

bei DPLL für CNF F:
bei Konflikt für Variable k mit aktueller Belegung b
bestimme *Konflikt-Graph* G:

- Knoten: $\mathrm{dom}(b) \cup \{k\}$ mit Label $b(v)$

- Kanten: $v \to v'$, falls v' durch eine Propagation belegt wurde mit einer Unit-Klausel, die v enthält

Eigenschaften von G:

- durch Decide belegte Variablen haben keine Vorgänger
- Konfliktvariable hat keinen Nachfolger, (wenigstens) zwei Vorgänger

Lernen aus dem Konflikt-Graphen (Korrektheit)

Satz: für Konflikt-Graphen G mit Konflikt k für CNF F bei Belegung b gilt: für jede Klausel C gilt:

- wenn jeder Pfad in G von einer Decide-Variablen zu k durch einen Knoten v geht mit $\neg b(v) \in C$,
- dann $F \models C$, d.h., $\mathrm{Mod}(F) = \mathrm{Mod}(F \cup \{C\})$, d.h. das Lernen von C ist *korrekt*

Beweis: betrachte $\mathrm{Var}(C) \subseteq V(G)$. Von dort aus kann man Unit-Propagationen ausführen, die zu einem Widerspruch in k führen. (Die Information, die man aus den Decide-Knoten benötigt, ist bereits in $\mathrm{Var}(C)$ enthalten.)

Lernen aus dem Konflikt-Graphen (Termination)

Satz: unter Vorauss. wie eben:

- wenn ein solches C gelernt wird,
- und zum vorletzten Entscheidungslevel der Variablen in C zurückgekehrt wird
- dann wird die Belegung b nie mehr vorkommen

Beweis: . . . denn sie wird durch Unit-Propagation mit C verhindert.

Lernen (Implementierung)

Wie wählt man C? (widersprechende) Ziele sind:

- C ist kurz (führt eher zu Propagationen, schränkt den Suchraum besser ein)
- C enthält Variablen geringer Entscheidungshöhe (dann kann man weiter zurückspringen)

mögliche Implementierung

```
c := Klausel, die zu Konflikt führte,
while (...) {
  l := das in c zuletzt belegte Literal
  d := Klausel, durch die var(l) belegt wurde
  c := resolve (c,d,var(l)); }
```

Korrektheit folgt aus Eigenschaften der Resolution.
verschiedene Heuristiken zum Aufhören.

4 UnSAT-Solver

Beweise für Nichterfüllbarkeit

- bisher: Interesse an erfüllender Belegung $m \in \mathrm{Mod}(F)$ (= Lösung einer Anwendungsaufgabe)
- jetzt: Interesse an $\mathrm{Mod}(F) = \emptyset$.
  Anwendungen: Schaltkreis C erfüllt Spezifikation $S \iff \mathrm{Mod}(C(x) \neq S(x)) = \emptyset$.

Solver rechnet lange, evtl. Hardwarefehler usw.

- $m \in \mathrm{Mod}(F)$ kann man leicht prüfen
  (unabhängig von der Herleitung)
- wie prüft man $\mathrm{Mod}(F) = \emptyset$?
  (wie sieht ein *Zertifikat* dafür aus?)

Resolution

- ein Resolutions-Schritt:
$$\frac{(x_1 \vee \ldots \vee x_m \vee y), (\neg y \vee z_1 \vee \ldots \vee z_n)}{x_1 \vee \ldots \vee x_m \vee z_1 \vee \ldots \vee z_n}$$
- Sprechweise: Klauseln C_1, C_2 werden nach y *resolviert*.
- Schreibweise: $C = C_1 \oplus_y C_2$
- Beispiel:
$$\frac{x \vee y, \neg y \vee \neg z}{x \vee \neg z}$$
- Satz: $\{C_1, C_2\} \models C_1 \oplus_y C_2$. (Die Resolvente folgt aus den Prämissen.)

Resolution als Inferenzsystem

mehrere Schritte:

- Schreibweise: $M \vdash C$
- Klausel C *ist ableitbar* aus Klauselmenge M
- Definition:
  - (Induktionsanfang) wenn $C \in M$, dann $M \vdash C$
  - (Induktionsschrit)
    wenn $M \vdash C_1$ und $M \vdash C_2$, dann $M \vdash C_1 \oplus_y C_2$

Beachte Unterschiede:

- Ableitung $M \vdash C$ ist *syntaktisch* definiert (Term-Umformung)
- Folgerung $M \models C$ ist *semantisch* definiert (Term-Auswertung)

Resolution und Unerfüllbarkeit

Satz: $\mathrm{Mod}(F) = \emptyset \iff F \vdash \emptyset$ (in Worten: F in CNF nicht erfüllbar $\iff$ aus F kann man die leere Klausel ableiten.)

- Korrektheit ($\Leftarrow$): Übung.
- Vollständigkeit ($\Rightarrow$): Induktion nach $|\mathrm{Var}(F)|$

dabei Induktionsschritt:

- betrachte F mit Variablen $\{x_1, \ldots, x_{n+1}\}$.
- Konstruiere F_0 (bzw. F_1) aus F durch „Belegen von x_{n+1} mit 0 (bzw. 1)“ (d. h. Streichen von Literalen und Klauseln)
- Zeige, daß F_0 und F_1 unerfüllbar sind.
- wende Induktionsannahme an: $F_0 \vdash \emptyset, F_1 \vdash \emptyset$
- kombiniere diese Ableitungen

Resolution, Bemerkungen

- Unit Propagation kann man als Resolution auffassen
- moderne SAT-Solver können Resolutions-Beweise für Unerfüllbarkeit ausgeben
- es gibt nicht erfüllbare F mit (exponentiell) großen Resolutionsbeweisen (sonst wäre NP = co-NP, das glaubt niemand)
- komprimiertes Format für solche Beweise (RUP—reverse unit propagation) wird bei “certified unsat track” der SAT-competitions verwendet (evtl. Übung)
- vollständige Resolution einer Variablen y als Preprocessing-Schritt

Vorverarbeitung

Niklas Eén, Armin Biere: *Effective Preprocessing in SAT Through Variable and Clause Elimination.* SAT 2005: 61-75 http://dx.doi.org/10.1007/11499107_5 http://minisat.se/downloads/SatELite.pdf

- clause distribution:
  Elimination einer Variablen y durch vollständige Resolution (Fourier-Motzkin-Verfahren):
  jede Klausel $C \ni y$ resolvieren gegen jede Klausel $C' \ni \overline{y}$,
  Originale löschen
- self-subsumption resolution (evtl. Übung)

Implementierung muß Subsumption von Klauseln sehr schnell feststellen (wenn $C_1 \subseteq C_2$, kann C_2 entfernt werden)

Reverse Unit Propagation

http://www.satcompetition.org/2014/certunsat.shtml *RUP proofs are a sequence of clauses that are redundant with respect to the input formula. To check that a clause C is redundant, all literals C are assigned to false followed by unit propagation. In order to verify redundancy, unit propagation should result in a conflict.*

$\curvearrowright$ Konflikt für $F \wedge \neg C \curvearrowright F \wedge \neg C$ ist nicht erfüllbar $\curvearrowright \neg F \vee C$ ist allgemeingültig $\curvearrowright F \models C$ (aus F folgt C) $\curvearrowright C$ „ist redundant“

siehe auch E.Goldberg, Y.Novikov. *Verification of proofs of unsatisfiability for CNF formulas.* Design, Automation and Test in Europe. 2003, March 3-7,pp.886-891 http://eigold.tripod.com/papers/proof_verif.pdf

DRAT (Deletion Resolution Asymmetric Tautology)

http://www.cs.utexas.edu/~marijn/drat-trim/
dazu ggf. Übungsaufgaben

Variablen-Elimination nach Fourier-Motzkin

- Jean-Baptiste Joseph Fourier, 1768–1830; Theodore Motzkin, 1908–1970
- Methode für Variablenelimination bei linearen Ungleichungen, hier angewendet für SAT.
- Elimination von x durch *vollständige* Resolution:
  - Klauselmenge (CNF) M, Teilmengen $M_x^+ = \{c \in M, x \in c\}$; $M_x^- = \{c \in M, \neg x \in c\}$;

  - $M' = M \setminus (M_x^+ \cup M_x^-) \cup \{c_1 \oplus_x c_2 \mid c_1 \in M_x^+, c_2 \in M_x^-\}$
  - Korrektheit: $\mathrm{Mod}(M) \neq \emptyset \iff \mathrm{Mod}(M') \neq \emptyset$
- Anwendungen:
  - Solver (im Allg. unpraktisch),
  - Präprozessor (sehr nützlich)

Übung Variablen-Elimination

- Korrektheit beweisen (Beweis enthält einen Algorithmus für die Rekonstruktion der Belegung)
- die frühere Erfüllbarkeits-Aufgabe (autotool) durch vollständige Elimination lösen
- Beschreibung der Vorverarbeitung in `minisat`: Niklas Een, Armin Biere, SAT 2005: `http://minisat.se/downloads/SatELite.pdf`

SAT-Kodierung von Anzahl-Constraints

- Anwendungen bei Aufgaben dieser Art:
  - *möglichst viele* Springer auf Schachbrett, *so daß* diese sich nicht bedrohen
  - *genau* ein Bit pro Zeile, pro Spalte (Permutationsmatrix)
  - `http://www2.stetson.edu/~efriedma/mathmagic/0315.html`
- Def: $b \models_k (x_1, \ldots, x_n) :\iff k \geq \#\{i \mid b(x_i)\}$
  entsprechend $_{k,k}$
- Impl. (naiv): $_k(x_1, \ldots, x_n) = \bigwedge_{S \in \binom{\{1,\ldots,n\}}{k+1}} \bigvee_{i \in S} \neg x_i$
  (besser?) mit Hilfsvariablen $a_{k,j} =_k (x_1, \ldots, x_j)$
- Kriterien: Korrektheit und Effizienz:
  - Anzahl der Klauseln, Variablen (Tseitin-Transformation)
  - Verhalten bezüglich Unit-Propagation

Binäre Kodierung

- benutzt Formeln (Schaltkreise) für
  - Halb-Addierer, Voll-Addierer

  - Addition beliebiger Binärzahlen
  - Vergleich von Binärzahl mit Konstante
- dann ist die Implementierung trivial: $_k(x_1, \ldots, x_n) = (\mathsf{binary}(k) \geq \sum \mathsf{binary}(x_i))$
- Verbesserung durch Anpassen der Bitbreite an
  - Bitbreite von k
  - Anzahl der Summanden (bei Teilsummen)

  ergibt lineare Anzahl von Variablen und Klauseln

  (Ü: Nachrechnen! Verbess. von $n \log n$ auf $n \log k$ auf n)

Propagierbarkeit

- Motivation: *semantische* Folgerungen aus einer partiellen Belegung sollen einfach *syntaktisch* ableitbar sein (durch *unit propagation*)
- Definition: eine CNF-Kodierung C einer Aussage A heißt *arc consistent*,

  falls für alle partiellen Belegungen b, Literale l gilt:

  wenn $b \cup A \models l$, dann ist l durch Unit-Propagationen aus b und C ableitbar.
- Wortbedeutung: *arc* = Bogen = Kante, Erklärung später
- Bsp: Binärkodierung von $_k$ ist nicht arc-consistent.

Übung Anzahl-Constraints

Modellierung:

- Schubfach-Problem (pigeon hole)

  das ergibt kleine schwere SAT-Instanzen (DPLL/CDCL-Solver schaffen das nicht)
- Sudoku (ohne Vorgaben)
- All-Interval-Series
- `http://www2.stetson.edu/~efriedma/mathmagic/0916.html`

Lösungsverfahren:

- *commander encoding* für $_1$ (Klieber, Kwon, 2007)
- dafür arc-consistency überprüfen
- Hölldobler and Nguyen, 2013 `http://www.wv.inf.tu-dresden.de/Publications/2013/report-13-04.pdf`
- siehe auch Diskussion und Quellen in `https://github.com/Z3Prover/z3/issues/755`
- autotool-Aufgabe zu arc-cons. entwerfen, implementieren

5 Binäre Entscheidungsgraphen (Grundlagen)

Motivation: aussagenlog. Formeln

- *kanonisch* repräsentieren
  (Def: $\mathrm{Mod}(F) = \mathrm{Mod}(G) \iff \mathrm{rep}(F) = \mathrm{rep}(G)$)
- *effizient* vernüpfen ($\mathrm{rep}(F \vee G) = \mathrm{rep}(F) \oplus \mathrm{rep}(G)$)

Literatur: D. E. Knuth: TAOCP (4A1) 7.1.4; Kroening, Strichman: Decision Procedures, 2.4. naive Ansätze (Ü: bestimme o.g. Eigenschaften):

- Formel F selbst?
- Modellmenge $\mathrm{Mod}(F)$ als Menge von Belegungen?

Darstellung von Modellmengen

Ordnung auf Variablen festlegen: $x_n > \ldots > x_2 > x_1$
und dann binärer Entscheidungsbaum:

- Blätter: beschriftet mit 0, 1
  repräsentiert leere (bzw. volle) Modellmenge $\emptyset$ bzw. $\{\emptyset\}$
- innere Knoten t auf Höhe k
  - Schlüssel: Variable x_k, Kinder: Bäume l, r

repräsentiert Teilmenge von $\{x_1, \ldots, x_k\} \to \mathbb{B}$
$[t] = \{\{(x_k, 0)\} \cup b \mid b \in [l]\} \cup \{\{(x_k, 1)\} \cup b \mid b \in [r]\}$

Für jede Formel F exist. genau ein solcher Baum B mit $[B] = \mathrm{Mod}(F)$.
Jeder Pfad von Wurzel zu 1 repräsentiert ein $b \in \{x_1, \ldots, x_k\} \to \mathbb{B}$ (ein Modell).

Entscheidungsgraphen mit Sharing

DAG (gerichteter kreisfreier Graph) $G = (V, E)$ heißt geordnetes binäres Entscheidungsdiagramm, falls:

- G hat einen Startknoten (ohne Vorgänger)
- die Endknoten (ohne Nachfolger) von G sind $\subseteq \{0, 1\}$.
- Struktur $s : V \setminus \{0, 1\} \to V \times \mathrm{Var} \times V$.

heißt *geordnet*, falls Variablen auf jedem Pfad absteigen
heißt *reduziert* (ROBDD), falls außerdem

- $\forall v \in V : s(v) = (l, x, r) \Rightarrow l \neq r$ (keine gleichen Kinder)
- $\forall v, w \in V : s(v) = s(w) \Rightarrow v = w$ (keine gleichen Knoten)

Randal E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Trans.Comp., C-35(8):677-691, 1986 `http://www.cs.cmu.edu/~bryant/pubdir/ieeetc86.pdf`

ROBDD - Beispiel

```
import qualified OBDD as O
import qualified OBDD.Data as O

let d = O.or [ O.unit 3 True, O.unit 5 False ]

import System.Process
readProcess "dot" [ "-Tx11" ] $ O.toDot $ d
```

ROBDD - Anwendung

- Variablen $x_{1,1} \ldots, x_{n,n}$,
- Formel Q_n mit $\mathrm{Mod}(Q_n) =$ alle konfliktfreien Anordnungen von n Damen auf dem $n \times b$-Brett

- $\mathrm{Mod}(Q_n)$ als ROBDD repräsentieren

  `https://github.com/jwaldmann/haskell-obdd/blob/master/examples/Queens.hs`

Eigenschaften von ROBDDs

- jedes ROBDD repräsentiert eine Menge von partiellen Belegungen
  (alle Pfade von Wurzel zu 1)
  Wert $[n]$ für Knoten mit $s(n) = (l, x, r)$ ist $\neg x \wedge [l] \vee x \wedge [r]$.
- Erfüllbarkeit, Allgemeingültigkeit trivial
- Satz: ROBDD repräsentiert Formeln *kanonisch*
  (zu gegebener Formel F und Variablenordung $>$
  gibt es genau ein ROBDD)
  Beweis: Konstruktion aus vollständigem Entscheidungsbaum
- Größe der ROBDDS?
- effiziente Operationen? (folgende Folien)

Operationen mit ROBDDs (Plan)

binäre boolesche Operation $f(n_1, n_2)$:
mit $s(n_i) = (l_i, x_i, r_i)$

- auf Blättern 0,1 Wert ausrechnen
- auf Knoten mit gleichen Variablen $(x_1 = x = x_2)$
  $f(n_1, n_2) = f(\neg x \wedge l_1 \vee x \wedge r_1, \neg x \wedge l_2 \vee x \wedge r_2) = \neg x \wedge (f(l_1, l_2)) \vee x \wedge (f(r_1, r_2))$
- auf Knoten mit versch. Variablen $(x_1 > x_2)$
  $f(n_1, n_2) = f(\neg x \wedge l_1 \vee x \wedge r_1, n_2) = \neg x \wedge (f(l_1, n_2)) \vee x \wedge (f(r_1, n_2))$.

Operationen mit ROBDDs (Implementierung)

- dynamische Optimierung (d. h. von unten nach oben ausrechnen und Ergebnisse merken).

  ergibt Laufzeit für $f(s, t)$ von $O(|s| \cdot |t|)$.

  $\Rightarrow$ worst-case-Laufzeit für Konstruktion eines ROBDD zu einer Formel: exponentiell

- memoization für Knoten-Konstruktion (“hash consing”)

  garantiert Reduktion

- sharing zwischen verschiedenen BDD-Graphen (⇒ “BDD base”, d.h. ein DAG mit mehreren Startknoten) erlaubt Nachnutzung von Arbeit

- *memoization* aller Konstruktor-Aufrufe *und Operationen*

ROBDD-Anwendungen: Modelle zählen

typische Anwendungen von ROBDD ...

- (*nicht* Erfüllbarkeit, denn ...)

- Äquivalenz

- Zählen von Modellen:

  $|\operatorname{Mod}(0)| = 0, |\operatorname{Mod}(1)| = 1,\ |\operatorname{Mod}(l, x, r)| = |\operatorname{Mod}(l)| + |\operatorname{Mod}(r)|$

diese Zahlen bottom-up dranschreiben: Linearzeit
dabei beachten, daß bei der Konstruktion evtl. Variablen verschwunden sind
Beispiel (Übung) Anzahl der Lösungen des n-Damen-Problems

ROBBD-Implementierungen

- `http://hackage.haskell.org/package/obdd`

```
import qualified Prelude ; import OBDD
import qualified Data.Set as S
let t = variable "x" || not (variable "y")
display t
number_of_models (S.fromList ["x", "y"]) t
```

- `http://vlsi.colorado.edu/~fabio/CUDD/` (F. Somenzi)

- `http://sourceforge.net/projects/buddy/` (J. Lind-Nielsen)

Übung BDD

- Konstruieren Sie das ROBDD für die Zählfunktion

  $\mathrm{count}_{\geq k}(x_1, \ldots, x_n) :=$ let $s = x_1 + \cdots + x_n$ in $s \geq k$

  a) für $k = 2, n = 4$, b) allgemein

- Bestimmen Sie die Anzahl der Modelle (für a)

- entspr. für die Funktion $x_1 \oplus \ldots \oplus x_n$

- Bestimmen Sie die Anzahl der n-Bit-Vektoren ohne benachbarte 1, indem Sie zu einer geeigneten Formel die BDDs berechnen und deren Modelle zählen.

- Anzahl der Überdeckungen eines Schachbretts $(w \times h)$ durch Dominos (2×1):

  Modellierung (Variablen? Constraints?), Implementierung

6 Binäre Entscheidungsgraphen (Ergänzungen)

ROBDDs und Automaten

- für gegebene Variablenordnung $x_1 < x_2 < \ldots < x_n$

  Belegung $\iff$ Wort über $\{0, 1\}^n$

- das ROBDD für Formel F *ist* der minimale vollständige deterministische Automat für $\mathrm{Mod}(F)$ (fast)

- Ü: bestimme diesen Automaten für $F = x_1 \oplus (x_2 \wedge \neg x_3)$

- Jedes $L \in \mathsf{REG}$ hat genau einen min. vollst. det. Aut.

  entspricht: Jede Formel F hat eindeutig bestimmtes ROBDD.

Modelle zählen und würfeln

- Anzahl der Modelle eines BDD kann bottom-up durch eine Rechenoperation pro Knoten bestimmt werden

- diese Anzahlen können groß werden (`int` reicht nicht, `double` ist ungenau,...)

- wenn man die Anzahlen in jedem Knoten hat, dann kann man eine *Gleichverteilung* auf allen Modellen realisieren:

  in der Wurzel beginnend, würfle in jedem Knoten die Variablenbelegung gewichtet nach den Modell-Anzahlen der beiden Kinder.

Binäre lineare Optimierung (Ü)

Gib einen Algorithmus an, der das folgende Problem mit Hilfe eines BDD für F effizient löst:

- gegeben:
  - aussagenlog. Formel F über Variablen $x_1, \ldots, x_n$
  - Gewichte $c_1, \ldots, c_n \in \mathbb{Z}$
- gesucht: eine Belegung b der Variablen, die
  - F erfüllt (d.h., $b \models F$)
  - und $\sum_i c_i \cdot b(x_i)$ maximiert.

Implementierung: `http://hackage.haskell.org/package/obdd-0.5.0/docs/OBDD-Linopt.html`

Anwendungen: z.B. „. . . möglichst viele, so daß. . .“

Die Größe von BDDs

- ein BDD mit k Knoten kann mit $\leq k \cdot (\log_2 k + \log_2 n + \log_2 k)$ Bit notiert werden
- es gibt 2^{2^n} Boolesche Funktionen von n Variablen,
- man braucht 2^n Bit, um *eine* Funktion zu notieren (der Werteverlauf)
- $\Rightarrow$ es gibt Funktionen auf n Variablen, deren BDD $\gtrsim 2^n$ Knoten benötigt.
- dieser Beweis ist nicht konstruktiv.
- einige konkrete Beispiele solcher Funktionen sind bekannt.

Beispiele, Einfluß der Variablenordnung

Bestimme das ROBDD für $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$

- für $a_1 > b_1 > a_2 > b_2 > a_3 > b_3$
- für $a_1 > a_2 > a_3 > b_1 > b_2 > b_3$

Funktionen mit exponentieller ROBDD-Größe für *jede* Variablenordnung: Bryant 1991

- hidden weighted bit function $f(x_1, \ldots, x_n) =$
  let $s = x_1 + \ldots + x_n$ in if $s = 0$ then 0 else x_s
- das mittlere Resultat-Bit bei integer multiplication
- $(x_{11}, x_{1n}, \ldots, x_{nn}) \mapsto$ (x ist Permutationsmatrix)

(`http://www.cs.cmu.edu/~bryant/pubdir/ieeetc91.pdf`)

Das Umordnen von Variablen in BDDs

- aus dem BDD der Funktion f zu Ordnung $\ldots > x > y > \ldots$

  kann man das BDD von f zu Ordnung $\ldots > y > x > \ldots$ (d.h., x tauscht mit y) bestimmen

- dabei werden nur die benachbarten Schichten von x und y betrachtet und geändert

- Ansatz zur Verkleinerung von BDDs:

  solange benachbarte Variablen vertauschen, bis Größe nicht mehr abnimmt

Anwendung BDD: Optimales Sortieren

- betrachten vergleichsbasierte Sortierverfahren

  (d.h., binäre Entscheidungsbäume: Verzweigungsknoten = Vergleich, Blatt = Permutation)

- informationstheoretische Schranke: Baum der Höhe $\leq h$ hat $\leq 2^h$ Blätter, also $2^h \geq n!$, Bsp. $\log_2 12! \approx 28.835$

- gibt es ein Sortierverfahren für 12 Elemente mit Tiefe 29?

- für jeden Knoten k gilt:

  - Pfad von Wurzel zu k bestimmt eine Halbordnung H,

- Anzahl der Permutationen, die mit H verträglich sind, muß $\leq 2^h$ sein, $h =$ Anzahl noch möglicher Vergleiche

Ansatz: bestimme diese Anzahlen mittels BDDs

BDDs zum Abzählen von Permutationen

- Bsp: wieviele Permutationen von $[1, \ldots, 5]$ sind verträglich mit der durch $1 < 2, 2 < 3, 2 < 4$ erzeugten Halbordnung? (Antw: 10 Stück)
- Boolesche Kodierung von:
  - $[x_1, \ldots, x_n]$ ist Perm. von $[1, \ldots, n]$
    one-hot encoding für Zahlen, d.h., *exactly-one* für jede Zeile und jede Spalte
  - $[x_1, \ldots, x_n]$ ist kompatibel mit $x_i < x_j$
    order encoding, d.h., Zahl als schwach monoton steigende Bitfolge

  Quelltexte: `https://gitlab.imn.htwk-leipzig.de/waldmann/min-comp-sort`

Übung BDD

- welche Funktionen zum Zählen von Modellen sind in der API von CUDD, BuDDy?
- ändere das Programm `https://github.com/jwaldmann/haskell-obdd/blob/master/examples/Weight.hs`, so daß tatsächlich eine maximale dominierende Menge bestimmt wird (wie der Kommentar schon behauptet).
- bestimme mittels der Funktion `fold`

  die Wahrscheinlichkeit dafür, daß $x_1 \oplus \ldots \oplus x_{10}$ wahr ist, wenn jedes x_i (unabhängig von den anderen) mit Wahrscheinlichkeit $1/i$ wahr ist.

  („fehlende“ Knoten sind hier kein Problem, warum?)

Forschungsaufgaben BDD

- Kann man 16 Elemente mit 45 Vergleichen sortieren?

  Vgl. Peczarski, 2011: `https://arxiv.org/abs/1108.0866`

- (Kislitsin 196?, zitiert in Knuth: TAOCP Vol 3 Sect 5.3.1) Bezeichne mit $T(P)$ die Anzahl der Permutationen, die mit der von P erzeugten Halbordnung verträglich sind.

  Beweise (oder widerlege): Wenn P nicht total ist, gibt es Elemente x, y, so daß $1/3 \leq T(P \cup (x, y))/T(P) \leq 2/3$.

- Benutze BDDs zur Bestimmung kleiner Sortiernetze. Bestätige und erweitere Ergebnisse aus Bundala et al., 2014: `https://arxiv.org/abs/1412.5302`

7 Prädikatenlogik

Plan

(für den Rest der Vorlesung)

- Prädikatenlogik (Syntax, Semantik)
- existentielle konjunktive Constraints

  in verschiedenen Bereichen, z. B.

  Gleichungen und Ungleichungen auf Zahlen $(\mathbb{Z}, \mathbb{Q}, \mathbb{R})$
- beliebige Boolesche Verknüpfungen

  SAT modulo T (= SMT), DPLL(T)
- Bit-blasting (SMT $\rightarrow$ SAT)

Syntax der Prädikatenlogik

- Signatur: Name und Stelligkeit für
  - Funktions-
  - und Relationssymbole

- Term:
  - Funktionssymbol mit Argumenten (Terme)
  - Variable
- Formel
  - atomar: Relationssymbol mit Argumenten (Terme)
  - Boolesche Verknüpfungen (von Formeln)
  - Quantor Variable Formel
-
  - gebundenes und freies Vorkommen von Variablen
  - Sätze (= geschlossene Formeln)

Semantik der Prädikatenlogik

- Universum, Funktion, Relation,
- Struktur, die zu einer Signatur paßt
- Belegung, Interpretation
- Wert
  - eines Terms
  - einer Formel

  in einer Struktur, unter einer Belegung

die Modell-Relation $(S, b) \models F$ sowie $S \models F$
Erfüllbarkeit, Allgemeingültigkeit (Def, Bsp)

Theorien

Def: $\mathrm{Th}(S) := \{F \mid S \models F\}$
(Die Theorie einer Struktur S ist die Menge der Sätze, die in S wahr sind.)

Bsp: „$\forall x : \forall y : x \cdot y = y \cdot x$“ $\in \mathrm{Th}(\mathbb{N}, 1, \cdot)$

Für K eine Menge von Strukturen:
Def: $\mathrm{Th}(K) := \bigcap_{S \in K} \mathrm{Th}(S)$
(die Sätze, die in jeder Struktur aus K wahr sind)

Bsp: „$\forall x : \forall y : x \cdot y = y \cdot x$“ $\notin \mathrm{Th}(\text{Gruppen})$
. . . denn es gibt nicht kommutative Gruppen, z.B. $\mathrm{SL}(2, \mathbb{Z})$

Unentscheidbarkeit

(Alonzo Church 1938, Alan Turing 1937)

Das folgende Problem ist nicht entscheidbar:

- Eingabe: eine PL-Formel F
- Ausgabe: *Ja*, gdw. F allgemeingültig ist.

Beweis: man kodiert das Halteproblem für ein universelles Berechnungsmodell als eine logische Formel.

Für Turingmaschinen braucht man dafür „nur“eine zweistellige Funktion

$f(i,t) =$ der Inhalt von Zelle i zur Zeit t.

Beachte: durch diese mathematische Fragestellung (von David Hilbert, 1928) wurde die Wissenschaft der Informatik begründet.

Folgerungen aus Unentscheidbarkeit

Suche nach (effizienten) Algorithmen für Spezialfälle
(die trotzdem ausreichen, um interessante Anwendungsprobleme zu modellieren)

- Einschränkung der Signatur (Bsp: keine F.-S., nur einstellige F.-S, nur einstellige Rel.-S.)
- Einschränkung der Formelsyntax
  - nur bestimmte Quantoren, nur an bestimmten Stellen
    (im einfachsten Fall: ganz außen existentiell)
  - nur bestimmte Verknüpfungen (Bsp: nur durch $\wedge$)
- Einschränkung auf Theorien von gegebenen Strukturen

  Bsp: $F \in \mathrm{Th}(\mathbb{N}, 0, +)$? Theorie der ganzen Zahlen mit Addition

8 Lineare Gleichungen und Ungleichungen

Syntax, Semantik

- lin. (Un-)Gleichungssystem $\rightarrow$ (Constraint $\wedge$)* Constraint
- Constraint $\rightarrow$ Ausdruck Relsym Ausdruck
- Relsym $\rightarrow \quad = \quad | \quad \leq \quad | \quad \geq$
- Ausdruck $\rightarrow$ (Zahl $\cdot$ Unbekannte $+$)* Zahl

Beispiel: $4y \leq x \wedge 4x \leq y - 3 \wedge x + y \geq 1 \wedge x - y \geq 2$

Semantik: Wertebereich für Unbekannte (und Ausdrücke) ist $\mathbb{Q}$ oder $\mathbb{Z}$

Normalformen

- Beispiel:
  $4y \leq x \wedge 4x \leq y - 3 \wedge x + y \geq 1 \wedge x - y \geq 2$
- Normalform: $\bigwedge_i \sum_j a_{i,j} x_j \geq b_i$

$$x - 4y \quad \geq \quad 0$$
$$\ldots$$

- Matrixform: $Ax^T \geq b^T$
  A ist linearer Operator.

Lösung von linearen (Un-)Gl.-Sys. mit Methoden der linearen Algebra

Hintergründe

Warum funktioniert das alles?

- lineares Gleichungssystem:
  Lösungsmenge ist (verschobener) *Unterraum*, endliche Dimension
- lineares Ungleichungssystem:
  Lösungsmenge ist *Simplex* (Durchschnitt von Halbräumen, konvex), endlich viele Seitenflächen

Wann funktioniert es nicht mehr?

- nicht linear: keine Ebenen
- nicht rational, sondern ganzzahlig: Lücken

Lineare Gleichungssysteme

Lösung nach Gauß-Verfahren:

- eine Gleichung nach einer Variablen umstellen,
- diese Variable aus den anderen Gleichungen eliminieren (= Dimension des Lösungsraumes verkleinern)

vgl. mit Elimination einer Variablen im Unifikations-Algorithmus

Lineare Ungleichungen und Optimierung

Entscheidungsproblem:

- Eingabe: Constraintsystem,
- gesucht: eine erfüllende Belegung

Optimierungsproblem:

- Eingabe: Constraintsystem und *Zielfunktion* (linearer Ausdruck in Unbekannten)
- gesucht: eine optimale erfüllende Belegung (d. h. mit größtmöglichem Wert der Zielfunktion)

Standard-Form des Opt.-Problems: $A \cdot x^T = b, x^T \geq 0$, minimiere $c \cdot x^T$.

Ü: reduziere OP auf Standard-OP, reduziere EP auf OP

Lösungsverfahren für lin. Ungl.-Sys.

- Simplex-Verfahren (für OP)

  Schritte wie bei Gauß-Verfahren für Gleichungssysteme (= entlang einer Randfläche des Simplex zu einer besseren Lösung laufen)

  Einzelheiten siehe Vorlesung Numerik/Optimierung

  exponentielle Laufzeit im schlechtesten Fall (selten)
- Ellipsoid-Verfahren (für OP): polynomiell
- Fourier-Motzkin-Verfahren (für EP)

  vgl. mit Elimination durch vollständige Resolution

  exponentielle Laufzeit (häufig)

Beispiel LP: monotone Interpretation

- Beispiel: das Wortersetzungssystem $R = \{aa \to bbb, bb \to a\}$ terminiert.
- Beweis: definiere $h : \Sigma \to \mathbb{N} : a \mapsto 5, b \mapsto 3$
  und setze fort zu $h^* : \Sigma^* \to \mathbb{N} : h(c_1 \dots c_n) = \sum h(c_i)$.
  Dann gilt $u \to_R v \Rightarrow h^*(u) > h^*(v)$ wegen $\forall (l \to r) \in R : h^*(l) > h^*(r)$.
- Die Gewichtsfunktion h erhalt man als Lösung des linearen Ungleichungssystems
  $2a > 3b \wedge 2b > a \wedge a \geq 0 \wedge b \geq 0$.

Beispiel LP-Solver

- Aufgabenstellung im LP-Format (`http://lpsolve.sourceforge.net/5.0/CPLEX-format.htm`)

```
Minimize
 obj: a + b
Subject To
 c1: 2 a - 3 b >= 1
 c2: 2 b -   a >= 1
End
```

- mit `https://projects.coin-or.org/Clp` lösen:

```
clp check.lp solve solu /dev/stdout
```

Fourier-Motzkin-Verfahren

Def.: eine Ungls. ist in x-Normalform, wenn jede Ungl.

- die Form „x ($\leq \mid \geq$) (Ausdruck ohne x)“ hat
- oder x nicht enthält.

Satz: jedes Ungls. besitzt äquivalente x-Normalform.

Def: für Ungls. U in x-Normalform:
$U_x^{\downarrow} := \{A \mid (x \geq A) \in U\}$, $U_x^{\uparrow} := \{B \mid (x \leq B) \in U\}$,
$U_x^{-} = \{C \mid C \in U, C \text{ enthält } x \text{ nicht}\}$.

Def: (x-Eliminations-Schritt) für U in x-Normalform:
$U \to_x \{A \leq B \mid A \in U_x^{\downarrow}, B \in U_x^{\uparrow}\} \cup U_x^{-}$

Satz: (U erfüllbar und $U \to_x V$) $\iff$ (V erfüllbar).

FM-Verfahren: Variablen nacheinander eliminieren.

(Mixed) Integer Programming

- "linear program": lineares Ungleichungssystem mit Unbekannten aus $\mathbb{Q}$
- "integer program": lineares Ungleichungssystem, mit Unbekannten aus $\mathbb{Z}$
- "mixed integer program": lineares Ungleichungssystem, mit Unbekannten aus $\mathbb{Q}$ und $\mathbb{Z}$

MIP-Beispiel

LP-Format mit Abschnitten

- `General` für ganzzahlige Unbekannte
- `Binary` für Unbekannte in $\{0, 1\}$

```
Minimize  obj: y
Subject To
 c1: 2 x  <= 1
 c2: - 2 x + 2 y <= 1
 c3:  2 x + 2 y >= 1
General  x y
End
```

Lösen mit `https://projects.coin-or.org/Cbc`:

```
cbc check.lp solve solu /dev/stdout
```

Ü: ausprobieren und erklären: nur x, nur y ganzzahlig

MIP-Lösungsverfahren

- Ansatz: ein MIP M wird gelöst,

  indem eine Folge von LP $L_1, \ldots$ gelöst wird.

- Def: *Relaxation* $R(M)$: wie M, alle Unbekannten reell.

- *Einschränkung*: für eine ganze Unbekannte x_i

  falls $\max\{x_i \mid \vec{x} \in \mathrm{Mod}(R(M))\} = B < \infty$,

  füge Constraint $x_i \leq \lfloor B \rfloor$ hinzu

- *Fallunterscheidung (Verzweigung)*:

  wähle eine ganze Unbekannte x_i und $B \in \mathbb{R}$ beliebig:

  $\mathrm{Mod}(M) = \mathrm{Mod}(M \cup \{x_i \leq \lfloor B \rfloor\}) \cup \mathrm{Mod}(M \cup \{x_i \geq \lceil B \rceil\})$

  entspricht *decide* in DPLL — aber es gibt kein CDCL

SAT als IP

gesucht ist Funktion T : CNF $\to$ IP mit

- T ist in Polynomialzeit berechenbar
- $\forall F \in$ CNF : F erfüllbar $\iff$ $T(F)$ lösbar

Lösungsidee:

- Variablen von $T(F) =$ Variablen von F
- Wertebereich der Variablen ist $\{0, 1\}$
- Negation durch Subtraktion, Oder durch Addition, Wahrheit durch ≥ 1

Komplexität von MIP

- LP ist in P (Ellipsoid-Verfahren) $\Rightarrow$ LP sind effizient lösbar.
  Wie schwer ist (M)IP?
- bekannt ist: SAT ist NP-vollständig,
  d.h., SAT $\in$ NP und $\forall L \in$ NP : $L \leq_P$ SAT.
  Beweis-Idee: SAT-Kodierung der Rechnung einer TM, die L akzeptiert.
  Benutze Matrix $A[t, p] =$ Inhalt von Zelle p zum Zeitpunkt t
- SAT $\leq_P$ IP $\quad \Rightarrow$ IP ist NP-hart
  deswegen gibt es keinen effizienten Algorithmus für IP (falls P $\neq$ NP)

[fragile,environment=slide]

Travelling Salesman als MIP

(dieses Bsp. aus Papadimitriou und Steiglitz: *Combinatorial Optimization*, Prentice Hall 1982)

Travelling Salesman:

- Instanz: Gewichte $w : \{1, \ldots, n\}^2 \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ und Schranke $s \in R_{\geq 0}$
- Lösung: Rundreise mit Gesamtkosten $\leq s$

Ansatz zur Modellierung:

- Variablen $x_{i,j} \in \{0, 1\}$, Bedeutung: $x_{i,j} = 1 \iff$ Kante (i, j) kommt in Rundreise vor
- Zielfunktion?
- Constraints — reicht das: $\sum_i x_{i,j} = 1, \sum_j x_{i,j} = 1$?

[fragile,environment=slide]

Travelling Salesman als MIP (II)

Miller, Tucker, Zemlin: *Integer Programming Formulation and Travelling Salesman Problem* JACM 7(1960) 326–329

- zusätzliche Variablen $u_1, \ldots, u_n \in \mathbb{R}$
- Constraints C: $\forall 1 \leq i \neq j \leq n : u_i - u_j + nx_{i,j} \leq n - 1$

Übung: beweise

- für jede Rundreise gibt es eine Belegung der u_i, die C erfüllt.
- aus jeder Lösung von C kann man eine Rundreise rekonstruieren.

Was ist die anschauliche Bedeutung der u_i?

[fragile,environment=slide]

min und max als MIP

- kann man den Max-Operator durch lin. Ungln simulieren?
  (gibt es äq. Formulierung zu $\max(x, y) = z$?)
- Ansatz: $x \leq z \wedge y \leq z \wedge (x = z \vee y = z)$,
  aber das *oder* ist verboten.

Idee zur Simulation von $A \leq B \vee C \leq D$:

- neue Variable $f \in \{0, 1\}$
- Constraint $A \leq B + \ldots \wedge C \leq D + \ldots$
- funktioniert aber nur ...

[fragile,environment=slide]

Übungen zu lin. Gl./Ungl.

- LP für Gewichtsfunktion für `http://termcomp.imn.htwk-leipzig.de/pairs/238238287`
- Beispiel Fourier-Motzkin-Solver: `https://gitlab.imn.htwk-leipzig.de/waldmann/pure-matchbox/blob/master/src/FM.hs`
- Formulierung eines SAT-Problems als IP, Lösung mit CBC.
- Überdeckungsproblem (möglichst wenige Damen, die das gesamte Schachbrett beherrschen) as IP,

  Constraint-System programmatisch erzeugen, z.B. `https://hackage.haskell.org/package/limp`, Lösen mit `https://hackage.haskell.org/package/limp-cbc`
- (Zusatz) lin. Gl. und Gauß-Verfahren in GF(2)

  (= der Körper mit Grundbereich $\{0, 1\}$ und Addition XOR und Multiplikation AND)

[fragile,environment=slide]

[fragile,environment=slide]

[fragile,environment=slide]

[fragile,environment=slide]

9 (Integer/Real) Difference Logic

[fragile,environment=slide]

Motivation, Definition

viele Scheduling-Probleme enthalten:

- Tätigkeit i dauert d_i Stunden
- i muß beendet sein, bevor j beginnt.

das führt zu Constraintsystem:

- Unbekannte: $t_i =$ Beginn von i
- Constraints: $t_j \geq t_i + d_i$

STM-LIB-Logik `QF_IDL, QF_RDL`:
boolesche Kombination von Unbekannte $\geq$ Unbekannte $+$ Konstante
[fragile,environment=slide]

Lösung von Differenz-Constraints

- (später:) Boolesche Kombinationen werden durch DPLL(T) behandelt,
- (jetzt:) der Theorie-Löser behandelt Konjunktionen von Differenz-Konstraints.
- deren Lösbarkeit ist in Polynomialzeit entscheidbar,
- Hilfsmittel: Graphentheorie (kürzeste Wege), schon lange bekannt (Bellman 1958, Ford 1960),

[fragile,environment=slide]

Constraint-Graphen für IDL

Für gegebenes IDL-System S konstruiere gerichteten kantenbewerteten Graphen G

- Knoten $i =$ Unbekannte t_i
- gewichtete Kante $i \xrightarrow{d} j$, falls Constraint $t_i + d \geq t_j$

beachte: Gewichte d können negativ sein. (wenn nicht: Problem ist trivial lösbar)

Satz: S lösbar $\iff$ G besitzt keinen gerichteten Kreis mit negativem Gewicht.

Implementierung: Information über Existenz eines solchen Kreises fällt bei einem anderen Algorithmus mit ab.

[fragile,environment=slide]

Kürzeste Wege in Graphen
(single-source shortest paths)

- Eingabe:
  - gerichteter Graph $G = (V, E)$
  - Kantengewichte $w : E \to \mathbb{R}$
  - Startknoten $s \in V$
- Ausgabe: Funktion $D : V \to \mathbb{R}$ mit $\forall x \in V : D(x) =$ minimales Gewicht eines Pfades von s nach x

äquivalent: Eingabe ist Matrix $w : V \times V \to \mathbb{R} \cup \{+\infty\}$
bei (von s erreichbaren) negativen Kreisen gibt es x mit $D(x) = -\infty$
[fragile,environment=slide]

Lösungsidee
iterativer Algorithmus mit *Zustand* $d : V \to \mathbb{R} \cup \{+\infty\}$.

$d(s) := 0, \forall x \neq s : d(x) := +\infty$
while es gibt eine Kante $i \overset{w_{i,j}}{\to} j$ mit $d(i) + w_{i,j} < d(j)$
$d(j) := d(i) + w_{i,j}$

jederzeit gilt die *Invariante*:

- $\forall x \in V$: es gibt einen Weg von s nach x mit Gewicht $d(x)$
- $\forall x \in V : D(x) \leq d(x)$.

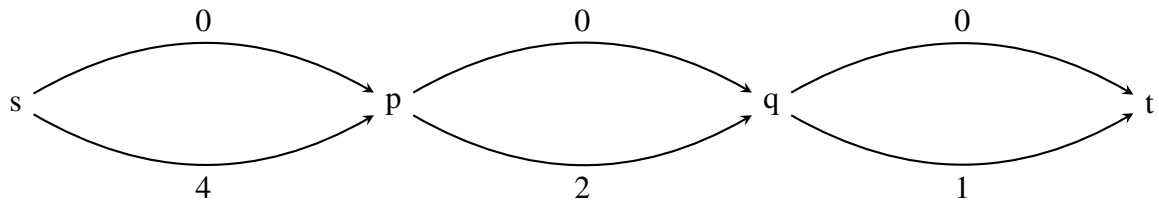
verbleibende Fragen:

- Korrektheit (falls Termination)
- Auswahl der Kante (aus mehreren Kandidaten)
- Termination, Laufzeit

[fragile,environment=slide]

Laufzeit

exponentiell viele Relaxations-Schritte:



besser:

- Bellman-Ford (polynomiell)

  for i from 1 to $|V|$: jede Kante $e \in E$ einmal entspannen

  dann testen, ob alle Kanten entspannt sind. (d. h. $d(j) \geq d(i) + w_{i,j}$)

  Wenn nein, dann existiert negativer Kreis. (Beweis?)
- Dijkstra (schneller, aber nur bei Gewichten ≥ 0 korrekt)

[fragile,environment=slide]

[fragile,environment=slide]

10 SMT, DPLL(T)

[fragile,environment=slide]

SMT = Satisfiability modulo Theory

- Aufgabenstellung:

  Erfüllbarkeitsproblem für beliebige boolesche Kombinationen von atomaren Formeln aus einer Theorie

  Beispiel: $x \geq 3 \vee x + y \leq 4 \leftrightarrow x > y$

- Lösungsplan:
  - Umformung in erfüllbarkeitsäquivalente CNF (Tseitin)
  - Lösung durch Variante von DPLL (“DPLL modulo Theory”)

Literatur: Kroening/Strichman, Kap. 11

[fragile,environment=slide]

SMT-{LIB,COMP}

- Standard-Modellierungssprache, Syntax/Semantik-Def: `http://smtlib.cs.uiowa.edu/standard.shtml`
- Aufgabensammlung: `http://smtlib.cs.uiowa.edu/benchmarks.shtml` Modelle aus Kombinatorik, Scheduling, Hard- und Software-Verifikation, ... Herkunft: crafted, industrial, (random)
- Wettbewerb: `http://www.smtcomp.org/`
- typische Solver:
  - Z3 `http://z3.codeplex.com/` (Microsoft Research)
  - Yices `http://yices.csl.sri.com/` (SRI, ehem. Stanford Research Inst.)

[fragile,environment=slide]

Beispiel `queen10-1.smt2` aus SMT-LIB

```
(set-logic QF_IDL) (declare-fun x0 () Int)
(declare-fun x1 () Int) (declare-fun x2 () Int)
(declare-fun x3 () Int) (declare-fun x4 () Int)
(assert (let ((?v_0 (- x0 x4)) (?v_1 (- x1 x4))
(?v_2 (- x2 x4)) (?v_3 (- x3 x4)) (?v_4 (- x0 x1))
(?v_5 (- x0 x2)) (?v_6 (- x0 x3)) (?v_7 (- x1 x2))
(?v_8 (- x1 x3)) (?v_9 (- x2 x3))) (and (<= ?v_0 3)
(>= ?v_0 0) (<= ?v_1 3) (>= ?v_1 0) (<= ?v_2 3) (>=
?v_2 0) (<= ?v_3 3) (>= ?v_3 0) (not (= x0 x1))
(not (= x0 x2)) (not (= x0 x3)) (not (= x1 x2))
(not (= x1 x3)) (not (= x2 x3)) (not (= ?v_4 1))
(not (= ?v_4 (- 1))) (not (= ?v_5 2)) (not (= ?v_5
(- 2))) (not (= ?v_6 3)) (not (= ?v_6 (- 3))) (not
(= ?v_7 1)) (not (= ?v_7 (- 1))) (not (= ?v_8 2))
(not (= ?v_8 (- 2))) (not (= ?v_9 1)) (not (= ?v_9
(- 1)))))) (check-sat) (exit)
```

[fragile,environment=slide]

Umfang der Benchmarks (2014)

`http://www.cs.nyu.edu/~barrett/smtlib/?C=S;O=D`

```
QF_BV_DisjunctiveScheduling.zip           2.7G
QF_IDL_DisjunctiveScheduling.zip          2.4G
incremental_Hierarchy.zip                 2.1G
QF_BV_except_DisjunctiveScheduling.zip    1.6G
QF_IDL_except_DisjunctiveScheduling.zip   417M
QF_LIA_Hierarchy.zip                      294M
QF_UFLRA_Hierarchy.zip                    217M
QF_NRA_Hierarchy.zip                      170M
QF_LRA_Hierarchy.zip                      160M
```

- QF: quantifier free,
- I: integer, R: real, BV: bitvector
- D: difference, L: linear, N: polynomial

[fragile,environment=slide]

DPLL(T), Prinzip

Ansatz: für jedes Atom $A = P(t_1, \ldots, t_k)$ eine neue boolesche Unbekannte $p_A \leftrightarrow A$.
naives Vorgehen:

- für jede Lösung des SAT-Problem für diese Variablen p_*:
- feststellen, ob die dadurch beschriebene Konjunktion von Atomen in der Theorie T erfüllbar ist

Realisierung mit DPLL(T):

- decide, T-solve (Konjunktion von T-Atomen)
- Konflikte (logische und T-Konfl.): backtrack
- logische Propagationen, Lernen
- T-Propagation (T-Deduktion)

[fragile,environment=slide]

DPLL(T), Beispiel QF_LRA

- T-Solver für Konjunktion von Atomen

  z. B. Simplex, Fourier-Motzkin

- T-Konfliktanalyse:

  bei Nichterfüllbarkeit liefert T-Solver eine „Begründung“

  = (kleine) nicht erfüllbare Teilmenge (von Atomen $\{a_1, \ldots, a_k\}$), dann Klausel $\neg a_1 \vee \ldots \vee \neg a_k$ lernen

- T-Deduktion, Bsp: aus $x \leq y \wedge y \leq z$ folgt $x \leq z$

  neues (!) Atom $x \leq z$ entsteht durch Umformungen während Simplex oder Fourier-Motzkin

  betrachte $\neg x \leq y \vee \neg y \leq z \vee x \leq z$ als Konfliktklausel, damit CDCL

[fragile,environment=slide]

DPLL(T): Einzelheiten, Beispiele

Literatur: Robert Nievenhuis et al.: `http://www.lsi.upc.edu/~roberto/RTA07-slides.pdf`

Univ. Barcelona, Spin-Off: Barcelogic,

Anwendung: `http://barcelogic.com/en/sports`

. . . software for professional sports scheduling. It has been successfully applied during the last five years in the Dutch professional football (the main KNVB Ere- and Eerste Divisies).

An adequate schedule is not only important for sportive and economical fairness among teams and for public order. It also plays a very important role reducing costs and increasing revenues, e.g., the value of TV rights.

[fragile,environment=slide]

Übung DPLL(T)

- ein DPLL(T)-Beispiel für T= Differenzlogik und lineare Ungl. durchrechnen. (evtl. auch autotool)
- dabei ggf. T-Entscheidungsverfahren wiederholen.
- welche Möglichkeiten bestehen dabei für T-Propagation? T-Lernen?

[fragile,environment=slide]

Übung SMT-LIB

- einige Beispiele aus SMT-LIB mit Z3 lösen
- Beispiele aus SMT-LIB verstehen (warum modelliert die Formel das Anwendungsproblem?)
- selbst ein Anwendungsproblem in SMT-LIB-Sprache modellieren

[fragile,environment=slide]

Aufgaben passend zur Jahreszeit

`https://www.mathekalender.de/index.php?page=calendar`
Löse die Aufgabe mit der jeweils passenden Methode der Constraint-Programmierung:

- 2. Dezember: Benni saß rechts vom Weihnachtsmann und links vom Cola-Trinker, ...
- 4. Dezember: Unsere Lebkuchenfirma möchte die Großhändler in Aspels, Bummerang und Cesaria beliefern. ...
- 5. Dezember: Es ist unmöglich, dass ein Rentier nach Wettkampfende 31 Punkte hat, ...

  beachte Verweis auf `https://www.matheon.de/research/projects?projectID=17&applicationAreaID=2`

[fragile,environment=slide]

Übung SMT-LIB und -Solver

- in Haskell eingebettete DSL für SMT-Kodierungen: `https://hackage.haskell.org/package/smtlib2`
- vervollständigen Sie (Aufgabe 15. Dezember) `https://gitlab.imn.htwk-leipzig.de/waldmann/cp-ws16/blob/master/kw50/A15.hs`
- damit dann die Aufgabe 5. Dezember

[fragile,environment=slide]

[fragile,environment=slide]

[fragile,environment=slide]

[fragile,environment=slide]

[fragile,environment=slide]

11 Polynomgleichungen

[fragile,environment=slide]

Hilberts 10. Problem

Entscheidung der Lösbarkeit einer diophantischen Gleichung.

Eine diophantische Gleichung mit irgendwelchen Unbekannten und mit ganzen rationalen Zahlkoefficienten sei vorgelegt: man soll ein Verfahren angeben, nach welchem sich mittels einer endlichen Anzahl von Operationen entscheiden lässt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist.

(Vortrag vor dem Intl. Mathematikerkongreß 1900, Paris)

`http://www.mathematik.uni-bielefeld.de/~kersten/hilbert/rede.html`

[fragile,environment=slide]

Probleme als Motor und Indikator des Fortschritts in der Wissenschaft

- Solange ein Wissenszweig Ueberfluß an Problemen bietet, ist er lebenskräftig; Mangel an Problemen bedeutet Absterben oder Aufhören der selbstständigen Entwickelung.
- . . . denn das Klare und leicht Faßliche zieht uns an, das Verwickelte schreckt uns ab.

- Ein mathematisches Problem sei ferner schwierig, damit es uns reizt, und dennoch nicht völlig unzugänglich, damit es unserer Anstrengung nicht spotte ...

Hilbert 1900. — (vgl. auch *Millenium Problems* `http://www.claymath.org/millennium/`)

[fragile,environment=slide]

Beispiele f. Polynomgleichungen

Einzelbeispiele:

- $x^2 - 3x + 2 = 0$
- $(x - 1) \cdot (x - 2) = 0$

Verknüpfungen:

- Kodierung von $P = 0 \vee Q = 0 \vee R = 0$ durch *eine* Gleichung: ...
- Kodierung von $P = 0 \wedge Q = 0 \wedge R = 0$ durch *eine* Gleichung: ...

Teilbereiche:

- Kodierung von $x \in \mathbb{R} \wedge x \geq 0$
- Kodierung von $x \in \mathbb{Z} \wedge x \geq 0$

[fragile,environment=slide]

Geschichte

Lösbarkeit in reellen Zahlen: ist entscheidbar

- Polynom in einer Variablen: Descartes 1637, Fourier 1831, Sturm 1835, Sylvester 1853
- Polynom in mehreren Variablen:
  - Tarski $\approx$ 1930
  - Collins 1975 (cylindrical algebraic decomposition)

Lösbarbeit in ganzen Zahlen: ist nicht entscheidbar

- Fragestellung: Hilbert 1900
- Beweis: Davis, Robinson, Matiyasevich 1970

[fragile,environment=slide]

Polynomgleichungen über $\mathbb{R}$

- jedes Polynom von ungeradem Grad besitzt wenigstens eine reelle Nullstelle (folgt aus Stetigkeit)
- ... diese ist für Grad > 4 nicht immer durch Wurzelausdrücke darstellbar (folgt aus Galois-Theorie)

... trotzdem kann man reelle Nullstellen *zählen*!

- Sturmsche Kette: $p_0 = P, p_1 = P', p_{i+2} = -\operatorname{rem}(p_i, p_{i+1})$
- $\sigma(P, x) :=$ Anzahl der Vorzeichenwechsel in $[p_0(x), p_1(x), \ldots]$
- Satz: $|\{x \mid P(x) = 0 \wedge a < x \leq b\}| = \sigma(P, a) - \sigma(P, b)$

[fragile,environment=slide]

Quantoren-Elimination über $\mathbb{R}$

- Sturmsche Ketten verallgemeinern für Polynome in mehreren Variablen
- Variablen der Reihe nach entfernen
- Realisierung durch
  - Tarski (1930): Laufzeit $\underbrace{\exp(\ldots(\exp}_{|V|}(1)\ldots))$
  - Collins (1975): QEPCAD (cyclindrical algebraic decomposition)
    Laufzeit $\exp(\exp(|V|))$

implementiert in modernen Computeralgebra-Systemen, vgl.`http://www.singular.uni-kl.de/Manual/3-1-3/sing_1815.htm`

[fragile,environment=slide]

Polynomgleichungen über $\mathbb{Z}$

Def: eine Menge $M \subseteq \mathbb{Z}^k$ heißt *diophantisch*,
wenn ein Polynom P mit Koeffizienten in $\mathbb{Z}$ existiert,
so daß $M =$
$\{(x_1, \ldots, x_i) \mid \exists x_{i+1} \in \mathbb{Z}, \ldots, x_k \in \mathbb{Z} : P(x_1, \ldots, x_k) = 0\}$
Beispiele:

- Menge der Quadratzahlen (leicht)
- Menge der natürlichen Zahlen (schwer)
- Menge der Zweierpotenzen (sehr schwer)
- Menge der Primzahlen?

Abschluß-Eigenschaften? (Durchschnitt, Vereinigung, ...)
[fragile,environment=slide]

Diophantische Mengen

Satz (Davis, Matiyasevich, Putnam, Robinson; $\approx$ 1970)

$$M \text{ diophantisch} \iff M \text{ ist rekursiv aufzählbar}$$

`http://logic.pdmi.ras.ru/~yumat/H10Pbook/`

positive Werte dieses Polynoms = Menge der Primzahlen
$(k+2)(1-(wz+h+j-q)^2-((gk+2g+k+1)(h+j)+h-z)^2-(2n+p+q+z-e)^2-(16(k+1)^3(k+2)(n+1)^2+1-f^2)^2-(e^3(e+2)(a+1)^2+1-o^2)^2-((a^2-1)y^2+1-x^2)^2-(16r^2y^4(a^2-1)+1-u^2)^2-(((a+u^2(u^2-a))^2-1)(n+4dy)^2+1-(x+cu)^2)^2-(n+l+v-y)^2-((a^2-1)l^2+1-m^2)^2-(ai+k+1-l-i)^2-(p+l(a-n-1)+b(2an+2a-n^2-2n-2)-m)^2-(q+y(a-p-1)+s(2ap+2a-p^2-2p-2)-x)^2-(z+pl(a-p)+t(2ap-p^2-1)-pm)^2)$
(challenge: ... find some) `http://primes.utm.edu/glossary/xpage/MatijasevicPoly.html`

12 Presburger-Arithmetik

[fragile,environment=slide]

Definition, Resultate

(nach Mojżesz Presburger, 1904–1943)

- Prädikatenlogik (d. h. alle Quantoren $\forall, \exists$, alle Booleschen Verknüpfungen)
- Signatur: Fun.-Symbole $0, 1, +$, Rel.-Symbole $=, <, \leq$
- interpretiert in der Struktur der natürlichen Zahlen

Resultate:

- Presburger 1929: Allgemeingültigkeit und Erfüllbarkeit solcher Formeln sind entscheidbar
- Fischer und Rabin 1974: Entscheidungsproblem hat Komplexität $\in \Omega(2^{2^n})$ (untere Schranke! selten!)

[fragile,environment=slide]

Beispiele f. Presburger-Formeln

Beispiele:

- es gibt eine gerade Zahl: $\exists x : \exists y : x = y + y$
- jede Zahl ist gerade oder ungerade: ...

definierbare Funktionen und Relationen:

- Minimum, Maximum
- Differenz? Produkt?
- kleiner-als ($<$) nur mit Gleichheit ($=$)?
- 0, 1, 2, 4, 8, ... $x = 2^k$ durch eine Formel der Größe $O(k)$

kann man noch größere Zahlen durch kleine Formeln definieren?

[fragile,environment=slide]

Entscheidungsverfahren (Ansatz)

Def.: Menge $M \subset \mathbb{N}^k$ heißt ***P-definierbar***
$\iff$ es gibt eine P-Formel F so daß $M = \mathrm{Mod}(F)$
wobei $\mathrm{Mod}(F) := \{m \in \mathbb{N}^k \mid \{x_1 \mapsto m_1, \ldots, x_k \mapsto m_k\} \models F\}$
für F mit freien Var. $x_1, \ldots, x_k$,

Satz: jede solche Modellmenge $\mathrm{Mod}(F)$ ist *effektiv regulär*:

- es gibt einen Algorithmus, der zu jeder P-Formel F ...
- ... einen endl. Automaten A konstruiert mit $\mathrm{Lang}(A) =$ Kodierung von $\mathrm{Mod}(F)$

Folgerung: Allgemeingültigkeit ist entscheidbar:
$\mathrm{Lang}(A) = \emptyset$ gdw. $\mathrm{Mod}(F) = \emptyset$ gdw. F ist widersprüchlich gdw. $\neg F$ ist allgemeingültig.
[fragile,environment=slide]

Entscheidungsverfahren (Kodierung)

Kodierung ist nötig,
denn $\mathrm{Mod}(F) \subseteq \mathbb{N}^k$, aber $\mathrm{Lang}(A) \subseteq \Sigma^*$.

wählen $\Sigma = \{0,1\}^k$, benutze Ideen:

- Kodierung einer Zahl: binär (LSB links)
  $c(3) = 11, c(13) = 1011$
- Kodierung eines Tupels: durch Stapeln
  $c(3,13) = (1,1)(1,0)(0,1)(0,1)$

Beispiele: Automat oder reg. Ausdruck für

- $\{c(x) \mid x$ ist gerade$\}$, $\{c(x) \mid x$ ist durch 3 teilbar$\}$,
- $\{c(x,y) \mid x + x = y\}$, $\{c(x,y,z) \mid x + y = z\}$,
- $\{c(x,y) \mid x \leq y\}$, $\{c(x,y) \mid x < y\}$

[fragile,environment=slide]

Formeln und Modellmengen

Idee: logische Verknüpfungen $\Rightarrow$ passende Operationen auf (kodierten) Modellmengen

- $\mathrm{Mod}(\mathsf{False}) = \emptyset$, $\mathrm{Mod}(\neg F) = \ldots$
- $\mathrm{Mod}(F_1 \wedge F_2) = \mathrm{Mod}(F_1) \cap \mathrm{Mod}(F_2)$
- $\mathrm{Mod}(\exists x_i.F) = \mathrm{proj}_i(\mathrm{Mod}(F))$
  Projektion entlang i-ter Komponente: $\mathrm{proj}_i : \mathbb{N}^k \to \mathbb{N}^{k-1}$:
  $(x_1, \ldots, x_k) \mapsto (x_1, \ldots, x_{i-1}, x_{x+1}, \ldots, x_k)$

zu zeigen ist, daß sich diese Operationen effektiv realisieren lassen (wobei Ein- und Ausgabe durch endl. Automaten dargestellt werden)
Ü: warum werden andere Verknüpfungen nicht benötigt?
[fragile,environment=slide]

Automaten (Definition)

Def: $A = (\Sigma, Q, I, \delta, F)$ mit

- Alphabet Σ, Zustandsmenge Q,
- Initialzustände $I \subseteq Q$, Finalzustände $F \subseteq Q$,
- Übergangsrelationen (f. Buchstaben) $\delta : \Sigma \to Q \times Q$.

daraus abgeleitet:

- Übergangsrelation f. Wort $w = w_1 \dots w_n \in \Sigma^*$:
  $\delta'(w) = \delta(w_1) \circ \dots \circ \delta(w_n)$
- A akzeptiert w gdw. $\exists p \in I : \exists q \in F : \delta'(w)(p, q)$
- Menge (Sprache) der akzeptierten Wörter:
  $\mathrm{Lang}(A) = \{w \mid A \text{ akzeptiert } w\} = \{w \mid I \circ \delta'(w) \circ F^{\mathrm{T}} \neq \emptyset\}$

[fragile,environment=slide]

Automaten (Operationen: Durchschnitt)

Eingabe: $A_1 = (\Sigma, Q_1, I_1, \delta_1, F_1), A_2 = (\Sigma, Q_2, I_2, \delta_2, F_2)$,
Ausgabe: A mit $\mathrm{Lang}(A) = \mathrm{Lang}(A_1) \cap \mathrm{Lang}(A_2)$
Lösung durch Kreuzprodukt-Konstruktion: $A = (\Sigma, Q, I, \delta, F)$ mit

- $Q = Q_1 \times Q_2$ (daher der Name)
- $I = I_1 \times I_2, F = F_1 \times F_2$
- $\delta(c)((p_1, p_2), (q_1, q_2)) = \dots$

Korrektheit: $\forall w \in \Sigma^* : w \in \mathrm{Lang}(A) \iff w \in \mathrm{Lang}(A_1) \wedge w \in \mathrm{Lang}(A_2)$

Komplexität: $|Q| = |Q_1| \cdot |Q_2|$ (hier: Zeit = Platz)
[fragile,environment=slide]

Automaten (Operationen: Komplement)

Eingabe: $A_1 = (\Sigma, Q_1, I_1, \delta_1, F_1)$,
Ausgabe: A mit $\mathrm{Lang}(A) = \Sigma^* \setminus \mathrm{Lang}(A_1)$
Lösung durch Potenzmengen-Konstruktion: $A = (\Sigma, Q, I, \delta, F)$ mit

- $Q = 2^{Q_1}$ (daher der Name)
- $I = \{I_1\}$, $F = 2^{Q_1 \setminus F_1}$
- $\delta(c)(M) = \ldots$

Korrektheit: $\forall w \in \Sigma^* : w \in \mathrm{Lang}(A) \iff w \notin \mathrm{Lang}(A_1)$

Komplexität: $|Q| = 2^{|Q_1|}$ (hier: Zeit = Platz)
[fragile,environment=slide]

Automaten (Operationen: Projektion)

Eingabe: Automat $A_1 = (\Sigma^k, Q_1, I_1, \delta_1, F_1)$, Zahl i
Ausgabe: A mit $\mathrm{Lang}(A) = \mathrm{Lang}(\mathrm{proj}_i(A_1))$
Lösung: $A = (\Sigma^{k-1}, Q, I, \delta, F)$ mit

- $Q = Q_1, I = I_1, F = F_1$
- $\delta(c)(p, q) = \ldots$

Korrektheit: $\forall w \in \Sigma^* : w \in \mathrm{Lang}(A) \iff w\,\mathrm{Lang}(\mathrm{proj}_i(A_1))$

Komplexität: $|Q| = |Q_1|$ (hier: Zeit = Platz)
[fragile,environment=slide]

Zusammenfassung Entscheidbarkeit

durch Automaten-Operationen sind realisierbar:

- elementare Relationen ($x + y = z$)
- Quantoren und logische Verknüpfungen

Folgerungen

- zu jeder Presburger-Formel F kann ein Automat A konstruiert werden mit $\mathrm{Mod}(F) = \mathrm{Lang}(A)$.

- Allgemeingültigkeit, Erfüllbarkeit, Widersprüchlichkeit von Presburger-Formel ist entscheidbar.

die Komplexität des hier angegeben Entscheidungsverfahrens ist hoch, geht das besser?

[fragile,environment=slide]

Eine untere Schranke

Fischer und Rabin 1974: `http://www.lcs.mit.edu/publications/pubs/ps/MIT-LCS-TM-043.ps`

Für *jedes* Entscheidungsverfahren E für Presburger-Arithmetik existiert eine Formel F,

so daß $E(F)$ wenigstens $2^{2^{|F|}}$ Rechenschritte benötigt.

Beweis-Plan: Diagonalisierung (vgl. Halteproblem):
wende solch ein Entscheidungsverfahren „auf sich selbst“ an.
Dazu ist Kodierung nötig (Turing-Programm $\leftrightarrow$ Zahl)

[fragile,environment=slide]

Untere Schranke

Für Maschine M und Eingabe x sowie Funktion $f : \mathbb{N} \to \mathbb{N}$ (z. B. $n \mapsto 2^{2^n}$) konstruiere Formel $D(M, x)$ mit

- $D(M, x) \iff$ Maschine M hält bei Eingabe x in $\leq f(|x|)$ Schritten.
- $D(M, x)$ ist klein und kann schnell berechnet werden.

Für jedes Entscheidungsverfahren E für Presburger-Arithmetik:

- konstruiere Programm $E_0(x)$: if $E(\neg D(x, x))$ then stop else Endlosschleife.
- Beweise: Rechnung $E_0(E_0)$ hält nach $> f(|E_0|)$ Schritten.

bleibt zu zeigen, daß man solche D konstruieren kann.

[fragile,environment=slide]

Presburger-Arithmetik und große Zahlen

(folgende Konstruktion ist ähnlich zu der, die im tatsächlichen Beweis verwendet wird)
es gibt eine Folge von P-Formeln $F_0, F_1, \ldots$ mit

- $|F_n| \in O(n)$

- $F_n(x) \iff x = 2^{2^n}$

Realisierung

- verwende $G_n(x, y, z)$ mit Spezifikation $x = 2^{2^n} \wedge xy = z$
  (die naive Implementierung ist aber zu groß)
- und Trick ("the device preceding Theorem 8"):
  $H(x_1) \wedge H(x_2)$ ist äquivalent zu
  $\forall x \forall y : (x = x_1 \vee \dots) \to \dots$ (nur ein Vorkommen von H)

[fragile,environment=slide]

Semilineare Mengen

- Def: $M \subseteq \mathbb{N}^k$ ist *linear*, falls
  $\exists \vec{a}, \vec{b_1}, \dots, \vec{b_n} \in \mathbb{N}^k : M = \{\vec{a} + \sum c_i \cdot \vec{b_i} \mid c_i \in \mathbb{N}\}$.
- Def: M ist *semi-linear*: M ist endliche Vereinigung von linearen Mengen.
- Satz (Ginsburg and Spanier, 1966, `http://projecteuclid.org/euclid.pjm/1102994974`) M ist Presburger-definierbar $\iff$ M ist semi-linear.

Dieser Satz ist effektiv:

- für „$\Rightarrow$": man kann die semi-lineare Darstellung durch Induktion über den Formelaufbau ausrechnen
- Übung: begründe „$\Leftarrow$".

[fragile,environment=slide]

13 Gleichheits-Constraints

[fragile,environment=slide]

(Dis)Equality Constraints (Bsp, Def)

$$\begin{aligned} & (a_0 = b_0 \wedge b_0 = a_1 \vee a_0 = c_0 \wedge c_0 = a_1) \wedge \ldots \\ \wedge \quad & (a_n = b_n \wedge b_n = a_{n+1} \vee a_n = c_n \wedge c_n = a_{n+1}) \\ \wedge \quad & a_{n+1} \neq a_0 \end{aligned}$$

(Quelle: Strichman, Rozanov, SMT Workshop 2007, http://www.lsi.upc.edu/~oliveras/espai/smtSlides/ofer.ppt, http://smtexec.org/exec/smtlib-portal-benchmarks.php?download&inline&b=QF_UF/eq_diamond/Feq_diamond10.smt2

Formel $\exists x_1, \ldots, x_n : M$ mit M in negationstechnischer Normalform über Signatur:

- Prädikatsymbole $=$ und $\neq$, keine Funktionssymbole

[fragile,environment=slide]

Lösungsplan: SAT-Kodierung

Ansatz:

- jede elementare Formel $(x = y)$ durch eine boolesche Unbekannte $(e_{x,y})$ ersetzen
- und Transitivität der Gleichheitsrelation kodieren

EQUALITY_CONSTRAINTS ist NP-vollständig, denn:

- E.C. $\in$ NP (laut Ansatz)
- SAT $\leq_P$ E.C. (Übung)

Verbesserungen (durch Analyse des *Gleichheitsgraphen*):

- Verkleinerung der Formeln (Entfernen von Variablen)
- Verringerung der Anzahl der Constraints für Transitivität

[fragile,environment=slide]

Gleichheitsgraph (equality graph)

- Knoten: Variablen
- Kanten xy, falls $x = y$ oder $x \neq y$ in M vorkommt (Gleichheitskante, Ungleichheitskante)

(logische Verknüpfungen werden ignoriert!)

Widerspruchskreis:

- geschlossene Kantenfolge
- genau eine Ungleichheitskante
- einfach (simple): kein Knoten doppelt

falls die Kanten eines solchen Kreise mit *und* verknüpft sind, dann ist die Formel nicht erfüllbar.

[fragile,environment=slide]

Vereinfachen von Formeln

Satz: Falls M eine Teilformel T (also $x = y$ oder $x \neq y$) enthält, die auf keinem Widerspruchskreis vorkommt,

dann ist $M' := M[T/\text{True}]$ erfüllbarkeitsäquivalent zu M.

Beweis: eine Richtung ist trivial,

für die andere: konstruiere aus erfüllender Belegung von M' eine erfüllende Belegung von M.

Algorithmus: solange wie möglich

- eine solche Ersetzung ausführen
- Formel vereinfachen (durch $(\text{True} \wedge x) = x$ usw.)

[fragile,environment=slide]

Reduktion zu Aussagenlogik (I)

- Plan: jede elementater Formel ($x = y$) durch eine boolesche Variable ersetzen, dann SAT-Solver anwenden.
- Widerspruchskreise sind auszuschließen: in jedem Kreis darf nicht genau eine Kante falsch sein.

  dadurch wird die Transitivität der Gleichheitsrelation ausgedrückt.
- es gibt aber exponentiell viele Kreise.

[fragile,environment=slide]

Reduktion zu Aussagenlogik (II)

Satz (Bryant und Velev, CAV-12, LNCS 1855, 2000):
es genügt, Transitivitäts-Constraints für sehnenlose Kreise hinzuzufügen.

Satz (Graphentheorie/Folklore):
zu jedem Graphen G kann man in Polynomialzeit einen chordalen Obergraphen G' konstruieren (durch Hinzufügen von Kanten).

Graph G heißt *chordal*, wenn er keinen sehnenlosen Kreis einer Länge ≥ 4 enthält.

Vorsicht: chordal $\neq$ trianguliert, Beispiel.

Algorithmus: wiederholt: einen Knoten entfernen, dessen Nachbarn zu Clique vervollständigen.
[fragile,environment=slide]

Ausflug Graphentheorie

Chordale Graphen G sind *perfekt*: für jeden induzierten Teilgraphen $H \subseteq G$ ist die *Cliquenzahl* $\omega(H)$ ist gleich der chromatischen Zahl $\chi(H)$.
Weiter Klassen perfekter Graphen sind:

- bipartite Graphen, z. B. Bäume
- split-Graphen, Intervallgraphen
- (C_5 ist nicht perfekt)

Strong Perfect Graph Theorem (Vermutung: Claude Berge 1960, Beweis: Maria Chudnovsky und Paul Seymor 2006):
G perfekt $\iff$ G enthält kein C_{2k+1} oder $\overline{C_{2k+1}}$ für $k \geq 2$.
`http://users.encs.concordia.ca/~chvatal/perfect/spgt.html`
[fragile,environment=slide]

Kleine Welt

für Gleichheits-Constraints mit n Variablen $x_1, \ldots, x_n$:
die hier gezeigte Kodierung benutzt n^2 boolesche Variablen $b_{i,j} \iff (x_i = x_j)$
das kann man reduzieren:
Wenn ein solches Gleichheits-System erfüllbar ist, dann besitzt es auch ein Modell mit $\leq n$ Elementen.
(Beweis-Idee: schlimmstenfalls sind alle Variablenwerte verschieden)
Den Wert jeder Variablen kann man also mit $\log n$ Bits kodieren.

Erweiterung: man kann für jede Variable einen passenden (evtl. kleineren) Bereich bestimmen.

14 Uninterpretierte Funktionen (UF)

[fragile,environment=slide]

Motivation, Definition

Interpretation $\models$ Formel,

Interpretation = (Struktur, Belegung)

Die *Theorie* $\mathrm{Th}(S)$ einer Struktur S ist Menge aller in S wahren Formeln: $\mathrm{Th}(S) = \{F \mid \forall b : (S, b) \models F\}$

Beispiel 1: Formel $a{\cdot}b = b{\cdot}a$ gehört zur $\mathrm{Th}(\mathbb{N}$ mit Multipl.), aber nicht zu $\mathrm{Th}($Matrizen über $\mathbb{N}$ mit Multipl

Beispiel 2: Formel

$$(\forall x : g(g(x)) = x \wedge g(h(x)) = x) \rightarrow (\forall x : h(g(x)) = x)$$

gehört zu *jeder* Theorie (mit passender Signatur),

weil sie zur Theorie der *freien Termalgebra* gehört. [fragile,environment=slide]

Anwendungen

Für jede Σ-Algebra S gilt:

- Formel F ist allgemeingültig in der *freien* Σ-Algebra
- $\Rightarrow$ Formel F ist allgemeingültig in S.

Vorteil: kein Entscheidungsverfahren für S nötig

Nachteil: Umkehrung gilt nicht.

Anwendung bei Analyse von Programmen, Schaltkreisen: Unterprogramme (Teilschaltungen) als *black box*,

Roope Kaivola et al.: *Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation*, Conf. Computer Aided Verification 2009, `http://dx.doi.org/10.1007/978-3-642-02658-4_32`

[fragile,environment=slide]

Terme

- Signatur Σ:

  Menge von Funktionssymbolen mit Stelligkeiten
- $t \in (\Sigma)$: geordneter gerichteter Baum mit zu Σ passender Knotenbeschriftung
- Σ-Algebra:
  - Trägermenge D
  - zu jedem k-stelligen Symbol $f \in \Sigma$ eine Funktion $[f] : D^k \to D$.

  damit wird jedem $t \in (\Sigma)$ ein Wert $[t] \in D$ zugeordnet
- (Σ) ist selbst eine Σ-Algebra

[fragile,environment=slide]

Gleichheit von Termen

In jeder Algebra gelten diese Formeln:

$$(t_1 = s_1) \wedge \ldots \wedge (t_k = s_k) \to f(t_1, \ldots, t_k) = f(s_1, \ldots, s_k)$$

(Leibniz-Axiom für die Gleichheit, *functional consistency*)

- Definition: eine Σ-Algebra heißt *frei*, wenn keine Gleichheiten gelten, die nicht aus o.g. Axiomen folgen.
- Beispiel: jede Termalgebra ist frei.
- Nicht-Beispiel: $\Sigma = \{+, 1\}, D = \mathbb{N}$ ist nicht frei.
- Ü: eine freie Algebra auf $\mathbb{N}$ zur Signatur $\{f/2\}$?
- Ü: die von $F(x) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} x$; $G(x) = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} x$; $I() = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ erzeugte Algebra ist frei?

[fragile,environment=slide]

Die Logik QF_UF in SMT-LIB

Bsp: die Formel $(x = y) \wedge (f(f(g(x))) \neq f(f(g(y))))$

```
(set-logic QF_UF) (declare-sort U 0)
(declare-fun f (U) U) (declare-fun g (U) U)
(declare-fun x () U) (declare-fun y () U)
(assert (and (= x y)
          (not (= (f (f (g x))) (f (f (g y)))))))
(check-sat)
```

ist nicht erfüllbar, d. h. das Gegenteil ist allgemeingültig:

$$\forall f, g, x, y : ((x = y) \rightarrow (f(f(g(x))) = f(f(g(y)))))$$

[fragile,environment=slide]

Ackermann-Transformation

... von Formel F in `QF_UF` nach Formel F' in *equalitiy logic* (= `QF_UF` mit nur nullstelligen Symbolen)

- ersetze jeden vorkommenden Teilterm F_i durch ein neues nullstelliges Symbol f_i
- füge *functional consistency constraints* hinzu:
  für alle $F_i = g(F_{i1}, \ldots, F_{ik}), F_j = g(F_{j1}, \ldots, F_{jk})$ mit $i \neq j$:
  $(f_{i1} = f_{j1} \wedge \ldots \wedge f_{ik} = f_{jk}) \rightarrow f_i = f_j$

Satz: F erfüllbar $\iff$ F' erfüllbar.

[fragile,environment=slide]

[fragile,environment=slide]

15 Programme mit Arrays

[fragile,environment=slide]

Definition

- Signatur mit Sorten: Array, Index, Element
- Symbole: Gleichheit, select : Array $\times$ Index $\rightarrow$ Element, store : Array $\times$ Index $\times$ Element $\rightarrow$ Array
- Axiome: $\forall a \in$ Array, $i, j \in$ Index, $e \in$ Element : $\mathrm{select}(\mathrm{store}(a, i, e), j) = \ldots$

John McCarthy and James Painter: *Correctness of a Compiler for Arithmetic Expressions*, AMS 1967, `http://www-formal.stanford.edu/jmc/mcpain.html`, `http://goedel.cs.uiowa.edu/smtlib/theories/ArraysEx.smt2`

[fragile,environment=slide]

Array-Logik (Beispiel mit Quantoren)

(aus Kroening/Strichman, Kap. 7)

Gültigkeit der Invariante $\forall x : 0 \leq x < i \Rightarrow a[x] = 0$ für die Schleife

```
for (int i = 0; i<N; i++) {
  a[i] = 0;
}
```

folgt aus Gültigkeit der Formel

$$\begin{array}{rl} & (\forall x : (0 \leq x \wedge x < i) \Rightarrow \mathrm{select}(a, x) = 0) \\ \wedge & (a' = \mathrm{store}(a, i, 0)) \\ \Rightarrow & \ldots \end{array}$$

[fragile,environment=slide]

Array-Logik (Beispiel QF_AX)

```
(set-logic QF_AX)
(declare-sort Index 0)
(declare-sort Element 0)
(declare-fun a () (Array Index Element))
(declare-fun i () Index)
(declare-fun x () Element)
(declare-fun y () Element)
(assert (not (= (store (store a i x) i y)
                (store a i y))))
(check-sat)
```

[fragile,environment=slide]

Array-Logik (Beispiel QF_AUFLIA)

```
(set-logic QF_AUFLIA)
(declare-fun a () (Array Int Int))
(declare-fun b () (Array Int Int))
(declare-fun i () Int)
(declare-fun j () Int)
(declare-fun k () Int)
(assert (< i j))
(assert (< (select a i) (select a j)))
(assert (= b (store a k 0)))
(assert (> i k))
(assert (> (select b i) (select b j)))
(check-sat)
```

[fragile,environment=slide]

Definition von Array-Formeln

(nach Kroening/Strichman Kap. 7.3)
wir betrachten boolesche Kombinationen in negationstechnischer Normalform (NNF) von Formeln der Gestalt $\forall i_1, \ldots, i_k : (I \Rightarrow V)$, wobei

- I (index guard): boolesche Kombination von linearen Gln. und Ungln. über $i_1, \ldots, i_k$,
- V (value constraint): boolesche Komb. von Gleichheitsconstraints für Array-Elemente.
  Index-Ausdrücke in select und store $\in \{i_1, \ldots, i_k\}$.

NNF: $\wedge/\vee$ beliebig, $\neg$ nur ganz unten, keine anderen Operatoren.
[fragile,environment=slide]

Vereinfachung von Array-Formeln:

- store entfernen:
  ersetze $\mathrm{store}(a, i, e)$ durch a' und füge Constraints hinzu:
  $\mathrm{select}(a', i) = e \wedge (\forall j : j \neq i \rightarrow \mathrm{select}(a', j) = \mathrm{select}(a', j))$.
- ersetze $\neg\forall j : P(j)$ durch $\neg P(z)$ mit neuer Variablen z

- ersetze $\forall j : P(j)$ durch $\bigwedge_{j \in J} P(j)$
  mit $J =$ alle vorkommenden Index-Ausdrücke (ohne die quantifizierten Variablen)
- select entfernen: ersetze $\mathrm{select}(a, i)$ durch $f_a(i)$

Für Startformel F der Array-Logik in NNF:
Verfahren erzeugt erfüllbarkeitsäquivalente Formel F'
mit Index-Prädikaten und uninterpretierten Funktionen.
$\Rightarrow$ Betrachtung von Theorie-Kombinationen [fragile,environment=slide]

Übung Array-Formeln/NNF

- $P \rightarrow Q$ in NNF?
- $\neg \forall i : P(i) \Rightarrow Q(i)$ auf ***Pränexform*** bringen (Quantor(en) außen)
- Regel „ersetze $\neg \forall i : \ldots$ “ ist nur korrekt für NNF (Gegenbeispiel?)

Sind die folgenden Eigenschaften durch Array-Formeln beschreibbar?

- das Array a ist monoton steigend
- b enthält wenigstens zwei verschiedene Einträge
- für jedes Element $= x$ gibt es rechts davon ein Element $\neq x$

[fragile,environment=slide]

16 Kombination von Theorien

[fragile,environment=slide]

Motivation

(Kroening/Strichman: Kapitel 10)

- Lineare Arithmetik + uninterpretierte Funktionen:
  $(x_2 \geq x_1) \wedge (x_1 - x_3 \geq x_2) \wedge (x_3 \geq 0) \wedge f(f(x_1) - f(x_2)) \neq f(x_3)$
- Bitfolgen + uninterpretierte Funktionen:
  $f(a[32], b[1]) = f(b[32], a[1]) \wedge a[32] = b[32]$

- Arrays + lineare Arithmetik

  $x = \text{select}(\text{store}(v, i, e), j) \wedge y = \text{select}(v, j) \wedge x > e \wedge x > y$

[fragile,environment=slide]

Definition

(Wdhlg) Signatur Σ, Theorie T, Gültigkeit $T \models \phi$
Kombination von Theorien:

- ... mit disjunkten Signaturen: $\Sigma_1 \cap \Sigma_2 = \emptyset$
- Theorie T_1 über Σ_1, Theorie T_2 über Σ_2.
- Theorie $T_1 \oplus T_2$ über $\Sigma_1 \cup \Sigma_2$

Aufgabenstellung:

- gegeben: Constraint-Solver (Entscheidungsverfahren) für T_1, Constraint-Solver für T_2;
- gesucht: Constraint-Solver für $T_1 \oplus T_2$

[fragile,environment=slide]

Konvexe Theorien

eine Theorie T heißt *konvex*, wenn für jede Konjunktion ϕ von Atomen, Zahl n, Variablen $x_1, \ldots, x_n, y_1, \ldots, y_n$ gilt:

- aus $T \models \forall x_1, \ldots, y_1, \ldots : (\phi \rightarrow (x_1 = y_1 \vee \ldots \vee x_n = y_n))$
- folgt: es gibt ein i mit $T \models \forall x_1, \ldots, y_1, \ldots : (\phi \rightarrow (x_i = y_i))$

Ü: warum heißt das *konvex*?

Beispiele: konvex oder nicht?

- lineare Ungleichungen über $\mathbb{R}$ (ja)
- lineare Ungleichungen über $\mathbb{Z}$ (nein)
- Konjunktionen von (Un)gleichungen (ja)

[fragile,environment=slide]

Das Nelson-Oppen-Verfahren (Quellen)

- Greg Nelson, Derek C. Oppen: *Simplification by Cooperating Decision Procedures*, ACM TOPLAS 1979, `http://doi.acm.org/10.1145/357073.357079`
- Oliveras und Rogriguez-Carbonell: *Combining Decisions Procedures: The Nelson-Oppen Approach*,

  als Teil der Vorlesung *Deduction and Verification Techniques*, U Barcelona, 2009 `https://www.cs.upc.edu/~oliveras/teaching.html`
- Tinelli und Harandi: *A new Correctness Proof of the Nelson-Oppen Combination Procedure*, FROCOS 1996.

  `http://homepage.cs.uiowa.edu/~tinelli/papers/TinHar-FROCOS-96.pdf`

[fragile,environment=slide]

Das Nelson-Oppen-Verfahren: purification

purification (Reinigung):

- durch Einführen neuer Variablen:
- alle atomaren Formeln enthalten nur Ausdrücke *einer* Theorie

Beispiel:

- vorher: $\phi := x_1 \leq f(x_1)$
- nachher: $\phi' := x_1 \leq a \wedge a = f(x_1)$

im Allg. $\phi \iff \exists a, \ldots : \phi'$
d. h. ϕ erfüllbar $\iff$ ϕ' erfüllbar.

[fragile,environment=slide]

Nelson-Oppen für konvexe Theorien

für entscheidbare Theorien $T_1, \ldots, T_n$ (jeweils T_i über Σ_i)

- Eingabe: gereinigte Formel $\phi = \phi_1 \wedge \ldots \wedge \phi_n$
  (mit ϕ_i über Σ_i)
- wenn ein ϕ_i nicht erfüllbar, dann ist ϕ nicht erfüllbar
- wenn $T_i \models (\phi_i \rightarrow x_i = y_i)$, dann Gleichung $x_i = y_i$ zu allen ϕ_j hinzufügen,...
- bis sich nichts mehr ändert, dann ϕ erfüllbar

(Beispiele)
(Beweis)
[fragile,environment=slide]

Nelson-Oppen, Beispiel

(Kroening/Strichman, Ex. 10.8)
NO-Verfahren anwenden auf:
$x_2 \geq x_1 \wedge x_1 - x_3 \geq x_2 \wedge x_3 \geq 0 \wedge f(f(x_1) - f(x_2)) \neq f(x_3)$

Diese Beispiel als Constraint-Problem in der geeigneten SMT-LIB-Teilsprache formulieren und mit Z3 Erfüllbarkeit bestimmen.
[fragile,environment=slide]

17 Finite Domain Constraints

[fragile,environment=slide]

Einleitung, Definition

- endliches Universum U (z.B. $[0, 1, 2, 3]$)
- (Funktionen und) Relationen auf U, z.B.
  - $G(x, y) := (x > y)$
  - $P(x, y, z) := (x + y = z)$

- $\exists$-quantifizierte Konjunktion von Atomen, z.B.
  $\exists x, y, z : P(x, y, z) \wedge P(x, x, y) \wedge G(y, x)$

das ist die *historische Form* der Constraint-Programmierung,
modern dargestellt in Krzysztof R. Apt: *Principles of Constraint Progr.*, Cambridge Univ. Press 2003
[fragile,environment=slide]

CNF-SAT als FD-Constraint-Problem

- Universum $U = \{0, 1\}$
- Funktion $N(x) := (1 - x)$,
- Relation $O(x_1, \ldots, x_k) = (x_1 + \ldots + x_k \geq 1)$

übersetze CNF $(p \vee \neg q) \wedge (q \vee r)$
in äquivalentes FD-Problem $O(p, N(q)) \wedge O(q, r)$

- FD-Constraint-Lösen ist wenigstens so schwer wie SAT-Lösen (also NP-vollständig)
- Algorithmen sind auch ähnlich (zu DPLL), Unterschiede:
  - SAT: CDCL
  - FD: Verfeinerung des Konzeptes *Konflikt*

[fragile,environment=slide]

Algorithmen für FD-Solver (Ansatz)
zum Lösen eines Constraint-Systems F: — DPLL für SAT:

- Zustand (Knoten im Suchbaum) ist partielle Belegung b
- Def.: $b_1 \leq b_2$ falls $b_1 \supseteq b_2$,
  (F, b) ist *konsistent* (erfüllbar): $\exists b' \leq b : b' \models F$
- Invariante:
  für jeden Knoten b mit Kindern $b_1(, b_2)$ gilt: $b_i \leq b$ und
  ((F, b) konsistent $\iff$ $b \models F \vee \exists i : (F, b_i)$ konsistent)

grundsätzliches Vorgehen bei FD:

- Zustand ist *Bereichs-Abbildung*: $\mathrm{dom} : V \to 2^U$
  ordnet jeder Variablen eine Teilmenge (*domain*) von U zu
- Inv.: für dom mit Kindern $\mathrm{dom}_1, \ldots$ gilt: $\mathrm{dom}_i \leq \mathrm{dom}$ und
  ((F, dom) kons. $\Leftrightarrow$ (F, dom) gelöst $\vee \exists i : (F, \mathrm{dom}_i)$ kons.).

[fragile,environment=slide]

Lösungen, Lösbarkeit

FD-Constraint F über Universum U, Bereichs-Abbildung $\mathrm{dom} : V(F) \to 2^U$, Belegung $b : V(F) \to U$, Bezeichungen:

- $b \in \mathrm{dom}$ falls $\forall v : b(v) \in \mathrm{dom}(v)$
- $\mathrm{dom}_1 \leq \mathrm{dom}_2$ falls $\forall v : \mathrm{dom}_1(v) \subseteq \mathrm{dom}_2(v)$
- (F, dom) heißt *konsistent*, falls $\exists b : V \to U$ mit $b \in \mathrm{dom}$ und $b \models F$, sonst *inkonsistent*.
- (F, dom) heißt *gelöst* (solved), wenn $\forall b \in \mathrm{dom} : b \models F$
- (F, dom) heißt *fehlgeschlagen* (failed), wenn $\exists v : \mathrm{dom}(v) = \emptyset$ oder V leer und F falsch

Satz:

- (F, dom) gelöst $\Rightarrow$ (F, dom) konsistent
- (F, dom) fehlgeschlagen $\Rightarrow$ (F, dom) inkonsistent

[fragile,environment=slide]

Algorithmen für FD-Solver

DPLL für SAT:

- Zustand (Knoten im Suchbaum) ist partielle Belegung
- Schritte (Kanten): Decide, Propagate, Conflict, Backtrack

grundsätzliches Vorgehen bei FD:

- Zustand ist *Bereichs-Abbildung*: $\mathrm{dom} : V \to 2^U$
- Decide: wähle $d \in \mathrm{dom}(v)$, Kind-Knoten $\mathrm{dom}'(v) = \{d\}$
- Backtrack: wähle ein anderes $d' \in \mathrm{dom}(v)$
- Conflict: dom fehlgeschlagen

- Propagate: verkleinere die Domains von Variablen
  (schließe Werte aus, die zu Inkonsistenzen führen)
  dafür gibt es (im Unterschied zu DPLL) viele Varianten

[fragile,environment=slide]

Globale und lokale Konsistenz

bei den Schritten während der Lösungssuche:

- die Invariante ist global

  (F, dom) kons. $\iff$ (F, dom) gelöst $\vee \exists i : (F, \mathrm{dom}_i)$ kons.

  ... und deswegen bei der Lösungssuche nicht unmittelbar nützlich

- die tatsächliche Auswahl des Schrittes benutzt *lokale* Konsistenz-Kriterien

  (Bsp. Kanten-Konsistenz, Hyperkanten-K., Pfad-K.)

  ... bei denen man nicht alle Belegungen aller Variablen durchprobieren muß

[fragile,environment=slide]

Kantenkonsistenz (Definition)

- Eine Bereichszuordnung dom

  für ein binäres atomares Constraint $C(x, y)$

  heißt *kantenkonsistent* (arc-consistent), wenn gilt:

  $\forall p \in \mathrm{dom}(x) \exists q \in \mathrm{dom}(y) : C(p, q)$

  und $\forall q \in \mathrm{dom}(y) \exists p \in \mathrm{dom}(x) : C(p, q)$

- Eine Bereichszuordnung dom für ein FD-Constraint F heißt kantenkonsistent, wenn dom für jedes binäre Atom in F kantenkonsistent ist.

Beispiele, Zusammenhang zw. Kantenkonsistenz und globaler Konsistenz?

[fragile,environment=slide]

Kantenkonsistenz (Herstellung)

- für ein binäres Constraint $C(x, y)$

  und Bereichszuordnung $\mathrm{dom}(x) = P, \mathrm{dom}(y) = Q$:

- definiere die *Projektionen*

  $P' = \{p \mid \exists q \in Q : C(p, q)\}, \quad Q' = \{q \mid \exists p \in P : C(p, q)\}.$

- und Inferenzregeln $\mathrm{Arc}_1 : (P, Q) \mapsto (P', Q), \quad \mathrm{Arc}_2 : (P, Q) \mapsto (P, Q')$.

- Satz: eine Zuordnung ist kantenkonsistent, wenn sie unter Arc_1 und Arc_2 abgeschlossen ist

- Algorithmus: solange dom nicht kantenkonsistent ist, wende Arc_1 und Arc_2 in beliebiger Reihenfolge an.

[fragile,environment=slide]

Hyperkantenkonsistenz

- Ein Atom $C(x_1, \ldots, x_n)$ mit Bereichszuordnung $\mathrm{dom} : x_i \mapsto D_i$ heißt n-(hyperkanten-)konsistent,

  wenn $\forall p_i \in D_i \exists q_1 \in D_1, \ldots, q_n \in D_n : C(q_1, \ldots, q_{i-1}, p_i, q_{i+1}, \ldots, q_n)$.

  Eine Formel ist n-konsistent, wenn alle Atome mit $\leq n$ Variablen n-konsistent sind

- Inferenzregel?

- Aufwand? (Antwort: $O(|U|^n)$)

- Nutzen? (Antwort: Einsparung von Decide-Knoten)

- beachte: Erweiterungen dieser Def. auf folgenden Folien

[fragile,environment=slide]

Hyperkantenkonsistenz und Konflikte

durch Hyperkanten-Inferenz kann man Konflikte feststellen:
Bsp. $(x_1 + x_2 < x_3)$ mit $D_1 = \{1, 2\}, D_2 = \{1, 2\}, D_3 = \{0, 1\}$

```
Arc_Consistency_Deduction
   { atom = x1 +x2 < x3
   , variable = x3, restrict_to = [  ]
   }
```

es wird ein *failed*-state erreicht ($\mathrm{dom}(x_3) = \emptyset$)

danach ist `Backtrack` möglich.
(das ist die *einzige* Möglichkeit der Konflikt-Feststellung in autotool-Aufgabe)
[fragile,environment=slide]

Hyperkantenkonsistenz: Erweiterungen

Bsp. $(x_1 + x_2 < x_3)$ mit $D_1 = \{0, 1\}, D_2 = \{0, 1\}, D_3 = \{1\}$

- nach Definition: das ist trivial 2-konsistent (es gibt keine 2-Atome), aber nicht 3-konsistent
- Erweiterung: da x_3 eindeutig ist: setze ein, erhalte 2-Atom $(x_1 + x_2 < 1)$, nicht 2-konsistent (betrachte $x_1 = 1$)

Bsp. $(x_1 \leq x_2 \wedge x_1 \neq x_2)$ mit $D_1 = \{0, 1, 2\}, D_2 = \{0, 1, 2\}$

- nach Definition: das ist 2-konsistent (jedes 2-Atom wird dabei einzeln betrachtet)
- Erweiterung: betrachte Konjunktion der Atome, dann nicht 2-konsistent, (betrachte $x_1 = 2$)

[fragile,environment=slide]

FD und SAT

- für alle konsistenz-basierten Methoden gilt
  wenn man nichts mehr propagieren kann, muß man entscheiden
- dabei wählt man einen aus evtl. sehr vielen Werten, d.h. man muß später oft zurückkehren (backtrack)

- man könnte sich zunächst nur für die linke oder rechte Hälfte eines Intervalls entscheiden (dann nur ein Backtrack)
- das entspricht einer Binärdarstellung
  dann kann man gleich ein aussagenlogisches Constraintsystem benutzen: der Löser kann dann Klauseln lernen usw.

[fragile,environment=slide]

Constr.-Programmierung mit Gecode

- ge neric co nstraint prog. d evelopment e nvironment
  `http://www.gecode.org/`
- Guido Tack: *Constraint Propagation - Models, Techniques, Implementation*, Diss., Uni Saarbrücken, `http://www.gecode.org/paper.html?id=Tack:PhD:2009`
- programmatische Benutzung (eingebettete DSL):
  - Constraints durch C++-Programm bauen
  - Lösungsstrategie durch C++-Programm beschreiben
  - ... und aufrufen — dabei sind auch Zustände während der Lösungs-Suche beobachtbar
- alternativ: externe DSL (z.B. minizinc) und Interpreter

[fragile,environment=slide]

Constraints in Mini/Flat-Zinc

- Peter Stuckey et al., *MiniZinc: Towards a standard CP modelling language.*, 2007
  `http://www.minizinc.org/`
- MiniZinc: Constraint-Sprache für den Menschen, zur *Modellierung*
  Bereiche für Unbekannte: endliche, Maschinenzahlen (ganz, Gleitkomma)
- Flat-Zinc: Constraint-Sprache für die Maschine, zum *Lösen*
- typische Anwendung (`gecode` als back-end):

```
mzn2fzn golomb.mzn 03.dzn ; fzn-gecode golomb.fzn
```

- Visualisierung des Suchbaums: `fzn-gecode -mode gist golomb.fzn`

[fragile,environment=slide]

[fragile,environment=slide]

18 Bitblasting

[fragile,environment=slide]

Motivation: SMT über Bitvektoren

SMT-Logik `QF_BV` motiviert durch (Rechner-)Schaltkreis- und (Maschinen-)Programmverifikation
`http://smtlib.cs.uiowa.edu/logics-all.shtml#QF_BV`
Constraints für ganze Zahlen fixierter Bitbreite w

- plus, minus, mal (*modulo* 2^w), min, max, ...
- Vergleiche, boolesche Verknüpfungen

```
(declare-fun a () (_ BitVec 12))
(declare-fun b () (_ BitVec 12))
(assert (and (bvult (_ bv1 12) a)
             (bvult (_ bv1 12) b)
             (= (_ bv1001 12) (bvmul a b))))
```

[fragile,environment=slide]

Lazy und Eager Approach für QFBV

- Hintegrund-Theorie $= T =$ CNF-SAT:
  - unbekannter Bitvektor = Folge unbekannter Bits
  - atomares Constraint = aussagenlogische Formel
    (z.B. Formel für Addier- o. Multiplizierschaltkreis)

- weitere spezifische Eigenschaften, z.B. Assoziativität, Kommutativität von Addition, Multiplikation
- DPLL(T): *lazy*

  T-Solver wird „bei Bedarf“ aufgerufen (mehrfach)
- bit-blasting: *eager*

  die Aufgabe wird komplett in die Theorie übersetzt

  und dann komplett durch den SAT-Solver gelöst

  Nachteil: das ignoriert spezifische Eigenschaften

[fragile,environment=slide]

SAT-Kodierung von FD-Constraints

Ansatz:

- FD: Unbekannte u mit Bereich $\{0, 1, \ldots, n-1\}$
- $\Rightarrow$ unbekannter Bitvektor $b = [b_0, \ldots, b_{w-1}]$ mit Constraints

Realisierungen:

- *binär*: b ist Binärdarstellung von u, $\quad (w = \lfloor \log_2 n \rfloor)$

  Constraint: $b < n$
- *unär* $(w = n)$
  - one-hot encoding: $\forall i : b_i \iff (i = u)$
    Constraint: ${}_1(b)$
  - order encoding: $\forall i : b_i \iff (i \leq u)$
    Constraint: (Übung)

[fragile,environment=slide]

one-hot-Kodierung

- SAT-Kodierung von ${}_1$ ist entscheidend

  und deswegen ausführlich untersucht, Übersicht in: Hölldobler, Van Hau Nguyen: `http://www.wv.inf.tu-dresden.de/Publications/2013/report-13-04.pdf`, vgl. `https://github.com/Z3Prover/z3/issues/755`

- Funktion, Relation $\Rightarrow$ Wertetabelle, Beispiel:
  $f(a,b) = c$ wird zu $\bigwedge_{i,j}(a_i \wedge b_j) \Rightarrow c_{f(i,j)}$
- Ü: Relation $<$ (kleiner als)?
- Ü: partielle Funktion? z.B. Addition auf $\{0, \ldots, n-1\}$
- (partielle) Funktion $D^k \to W$ benötigt $|D|^k$ Klauseln
  nur für kleine Bereiche, geringe Stelligkeiten praktikabel

[fragile,environment=slide]

Treppen-Kodierung (order encoding)

- Definition: $\forall i : b_i \iff (i \leq u)$
- Constraints: $\forall i : b_i \Leftarrow b_{i+1}$ (Monotonie)
- Ü: einfache (lineare) Kodierung von $<$ (kleiner als)
- Implementierung der Addition über Sortiernetze (Merge-Netze)
  Een, Sorenson: *Translating Pseudo-Boolean Constraints into SAT*, 2007. `http://minisat.se/downloads/MiniSat+.pdf`

[fragile,environment=slide]

Binäre Addition

$[x_0, \ldots] + [y_0, \ldots] = [z_0, \ldots]$
Hilfsvariablen: $[c_0, \ldots]$ = Überträge

- Anfang: $\mathrm{HALFADD}(x_0, y_0; z_0, c_0)$
- Schritt: $\forall i : \mathrm{FULLADD}(x_i, y_i, c_i; z_i, c_{i+1})$
- Ende: $c_w = 0$ (kein Überlauf)

Realisierung:

- $\mathrm{HALFADD}(x, y; r, c) \iff (r \leftrightarrow (x \oplus y)) \wedge (c \leftrightarrow (x \wedge y))$
- $\mathrm{FULLADD}(x, y, z; r, c) \iff \ldots$ (zweimal HALFADD)

Aufgaben: bestimme und vergleiche

- Tseitin-Transformation für FULLADD
- CNF *ohne* Hilfsvariablen für FULLADD

[fragile,environment=slide]

Multiplikation

$[x_0, \ldots]_2 \cdot [y_0, \ldots]_2 = [z_0, \ldots]_2$,

- Schulmethode: $z = \sum 2^i \cdot x_i \cdot y$ (sequentielle Summation)
- Verbesserungen: C.S. Wallace (1964), L. Dadda (1965),
  benutze *full-adder* für *verschränkte* Summation,
  (ähnlich zu carry-save-adder)
  verringert Anzahl der Gatter und Tiefe des Schaltkreises
  vgl. Townsend et al.:*A Comparison of...*, 2003 `http://www.cerc.utexas.edu/~whitney/pdfs/spie03.pdf`,
- scheint für CNF-Kodierung wenig zu helfen
  Testfall: Faktorisierung.

[fragile,environment=slide]

Bewertung von CNF-Kodierungen

- praktisches Ziel: geringer Solver-Laufzeit in *Anwendungen* (die viele Instanzen von kodierten elementaren Funktionen enthalten)
- das kann man aber nicht vorher messen

bei der Kodierung einer *einzelnen* Funktion

- Anzahl der (Hilfs-)Variablen, Klauseln; Größe der Klauseln

Zusammenhang zur Zielfunktion ist nicht klar. Besser ist

- Messung des Verhaltens der kodierten Formel bzgl. (Konflikterkennung und) Unit-Propagationen,

denn darauf kommt es bei DPLL an.

[fragile,environment=slide]

Kodierungen und Propagationen

- Def. CNF C heißt implikationstreu (arc-consistent, *forcing*), falls für alle partiellen Beleg.n b, Literale l gilt:

  wenn $b \cup C \models l$, dann $b \cup C \vdash_{\text{UP}} l$.

- Def. $M \vdash_{\text{UP}} l$ bedeutet: das Literal l ist aus Klauselmenge M durch Unit-Propagationen ableitbar

- Bsp: $(r, c) = (x \oplus y, x \wedge y)$ (Halb-Adder) Tseitin-kodiert $\{\overline{x} \vee \overline{y} \vee \overline{r}, \ldots\}$ ist nicht implikationstreu. Beweis: $b = \{\overline{r}, \overline{c}\}$

- um Implikationstreue zu erreichen, kann man passende *redundante* Klauseln hinzufügen.

  Ü: welche — für Halb-Adder, Full-Adder?

  Ü: hilft das für Multiplikation (Test: Faktorisierung)?

19 Anwendg.: Bounded Model Checking

[fragile,environment=slide]

Begriff, Motivation

- *model checking*: feststellen, ob
  - ein *Modell* eines realen Hard- oder Softwaresystems
    (z.B. Zustandsübergangssystem f. nebenläufiges Programm)
  - eine *Spezifikation* erfüllt
    (z.B. gegenseitiger Ausschluß, Liveness, Fairness)

- *symbolic* model checking:

  symbolische Repräsentation von Zustandsfolgen

  im Unterschied zu tatsächlicher Ausführung (Simulation)

- *bounded*: für Folgen beschränkter Länge

[fragile,environment=slide]

Literatur, Software

- Armin Biere et al.: *Symbolic Model Checking without BDDs*, TACAS 1999, `http://fmv.jku.at/bmc/`

  Software damals: Übersetzung nach SAT, später: SMT (`QB_BV`), Solver: `http://fmv.jku.at/boolector/`

- Daniel Kroening und Ofer Strichman: *Decision Procedures, an algorithmic point of view*, Springer, 2008. `http://www.decision-procedures.org/`

  Software: `http://www.cprover.org/cbmc/`

- Nikolaj Bjørner et al.: *Program Verification as Satisfiability Modulo Theories*, SMT-Workshop 2012, `http://smt2012.loria.fr/`

  Softw.: `https://github.com/Z3Prover/z3/wiki`

[fragile,environment=slide]

BMC für Mutual Exclusion-Protokolle

System mit zwei (gleichartigen) Prozessen A, B:

```
A0: maybe goto A1
A1: if l goto A1 else goto A2
A2: l := 1; goto A3
A3: [critical;] goto A4
A4: l := 0; goto A0

B0: maybe goto B1
B1: if l goto B1 else goto B2
B2: l := 1; goto B3
B3: [critical;] goto B4
B4: l := 0; goto B0
```

Schließen sich `A3` und `B3` gegenseitig aus? (Nein.)

(nach: Donald E. Knuth: TAOCP, Vol. 4 Fasz. 6, S. 20ff)

[fragile,environment=slide]

Modell: Zustandsübergangssystem

Zustände:

- jeder Zustand besteht aus:
  - Inhalte der Speicherstellen (hier: $l \in \{0, 1\}$)
  - Programmzähler (PC) jedes Prozesses
    (hier: $A \in \{0 \ldots 4\}, B \in \{0 \ldots 4\}$)
- Initialzustand: $I = \{l = 0, A = 0, B = 0\}$
- Menge der Fehlerzustände: $F = \{A = 3, B = 3\}$

Übergangsrelation (nichtdeterministisch): für $P \in \{A, B\}$:

- P führt eine Aktion aus (schreibt Speicher, ändert PC)

Aussagenlog. Formel für $I \rightarrow^{\leq k} F$ angeben,
deren Erfüllbarkeit durch SAT- oder SMT-Solver bestimmen
[fragile,environment=slide]

Übung BMC

- Software: `https://gitlab.imn.htwk-leipzig.de/waldmann/boumchak`
- überprüfe 1. gegenseitigen Ausschluß, 2. deadlock, 3. livelock (starvation) für weitere Systeme, z.B.

  E. W. Dijkstra, 1965: `https://www.cs.utexas.edu/~EWD/transcriptions/EWD01xx/EWD123.html#2.1.`

  G. L. Peterson, *Myths About the Mutual Exclusion Problem*, Information Processing Letters 12(3) 1981, 115–116

[fragile,environment=slide]

[fragile,environment=slide]

20 Zusammenfassung

[fragile,environment=slide]

Kernaussagen

- Constraint-Programmierung =
  - anwendungsspezifische logische Formel,
  - generische bereichsspezifische Such/Lösungsverfahren
- CP ist eine Form der deklarativen Programmierung
- typische Anwendungsfälle für CP mit Formeln ohne Quantoren, mit freien Variablen, Solver sagt:
  - JA: beweist Existenz-Aussage, rekonstruiere Lösung der Anwendungsaufgabe aus Modell
  - NEIN: das beweist All-Aussage (z. B. Implementierung erfüllt Spezifikation für jede Eingabe)

[fragile,environment=slide]

Kernthemen

Aussagenlogik (SAT)

- Grundlagen: Formeln, BDDs, Resolution, DPLL, CDCL
- Anwendungen: SAT-Kodierungen für kombinatorische Aufgaben, Bit-Blasting für andere Constraint-Bereiche

Prädikatenlogik: Bereiche und -spezifische Verfahren:

- finite domains, lokale Konsistenz
- Zahlen: Differenzlogik, lineare Ungleichungen (Fourier-Motzkin), Polynome, Presburger-Arithmetik
- uninterpretierte Funktionen, Arrays

Prädikatenlogik: allgemeine Verfahren:

- Kombination von Theorie und SAT (DPLL(T))

Beispiel-Klausur (anderes Jahr, mglw. andere Gewichtung der Themen) `http://www.imn.htwk-leipzig.de/~waldmann/edu/ss14/cp/klaus/`

[fragile,environment=slide]

Typische Anwendungen

- Ressourcen-Zuordungs-, -Optimierungs-Aufgaben mit
  - nicht linearen Funktionen
  - nicht nur konjuktiver Verknüpfung von Teilbedingungen
- Hardware-Verifikation (digitale Schaltnetze, Schaltwerke)
- Software-Verifikation (bounded model checking)

  Gastvortrag Dr. Carsten Fuhs: *Constraint-basierte Techniken in der automatisierten Programmanalyse*

  am 6. Februar, 13:45, Z 417. `https://portal.imn.htwk-leipzig.de/termine/constraint-basierte-techniken-in-der-automatisierten-program`

21 Einordnung, Ausblick

[fragile,environment=slide]

QBF-SAT (quantifizierte Boolesche Formeln)

- Bsp: $\exists x : \forall y : \exists z : (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$
- QDIMACS-Format:
```
p cnf 3 2
e 1 0  a 2 0  e 3 0
1 2 3 0  -1 -2 -3 0
```
- Ausdruckskraft steigt durch Quantorwechsel:
  - optimale Lösung: $\exists x : (P(x) \wedge \forall y : P(y) \Rightarrow |x| \leq |y|)$
  - (BMC): Pfad der Länge 2^k durch Formel der Größe k
  - QBF-SAT ist PSPACE-vollständig
- Testfälle, Competition: `http://www.qbflib.org/`
- Solver: z.B. `http://fmv.jku.at/depqbf/`

[fragile,environment=slide]

Symmetrien

- eine Symmetrie s eines Constraint-Systems P ist eine Permutation der Unbekannten U

  Bsp: Rechts-Links-Spiegelung einer Matrix

- Symmetrie-*Brechung* (korrekt und vollständig):

  wenn Menge von Symmetrien S mit P *verträglich* ist (Definition: $\forall s \in S : \forall x : P(x) \iff P(s(x))$),

  dann füge Constraints C hinzu mit

  $\forall x : P(x) \Rightarrow \exists y : x \sim_S y \wedge P(y) \wedge C(y)$

- Symmetrie-*Vermutung* (korrekt, evtl. unvollständig):

  füge Constraints $\forall s \in S : x = s(x)$ hinzu

  bzw. reduziere die Anzahl der Variablen

[fragile,environment=slide]

Symmetrien - Beispiel

- dominierende Menge von Springern auf Schachbrett

  Def: $D \subseteq V$ ist dominierende Menge in $G = (V, E)$,

  falls $\forall x \in V \setminus D : \exists y \in D : \{x, y\} \in E$

- `https://github.com/jwaldmann/ersatz/blob/master/examples/DominatingSet.hs`

```
b :: [[ Bit ]] <- allocate w
break_symmetries  [transpose, reverse, map reverse] b
assert_symmetries [ reverse, map reverse ] b
```

- Quellen: `http://oeis.org/A006075`, Frank Rubin 2005: `http://www.contestcen.com/knight.htm`
- für $w > 20$: verbessern — oder Optimalität beweisen!

[fragile,environment=slide]

Geschichte der Constraint-Progr. (I)

- Lineare Optimierung (Dantzig 1947 et al.)

  ⇒ Mixed Integer LP (Gomory 195* et al.) `https://www-03.ibm.com/ibm/history/exhibits/builders/builders_gomory.html`

- PROLOG (Kowalski 1974)

  - löst Unifikations-Constraints über Bäumen
  - mit fixierter Suchstrategie (SLD-Resolution)

  ⇒ Constraint Logic Programming (spezielle Syntax und Strategien für Arithmetik, endliche Bereiche)

  Global constraints in CHIP: Beldiceanu, Contejean 1994 `http://dx.doi.org/10.1016/0895-7177(94)90127-9`

[fragile,environment=slide]

Geschichte der Constraint-Progr. (II)

- fixierte, bekannte Suchstrategie: PROLOG
- Strategie innerhalb des CP bestimmt: gecode
- (praktisch unbekannte) Strategie im externen Solver:
  - SAT
    - DPLL: Martin Davis, Hilary Putnam (1960), George Logemann, Donald W. Loveland (1962),
    - SAT ist NP-vollst. (Steven Cook, Leonid Levin, 1971)
    - CDCL: J.P Marques-Silva, Karem A. Sakallah (1996)
  - SMT
    - DPLL(T): the *lazy* approach to SMT
    - Yices 2006,
    - Z3 (de Moura, Bjorner, 2008)

[fragile,environment=slide]

Constraint-Programmier-Sprachen

- Ziel: Funktion (Formel) $c : P \times U \to \{0, 1\}$,

  Eingabe: $p \in P$, Solver bestimmt $u \in U$ mit $c(p, u) = 1$.

- praktische Verwendbarkeit der Sprache hängt ab von

  - Ausdrucksstärke auf U
    z.B. Vergleich, Addition, Multiplikation
  - Ausdrucksstärke auf P
    z.B. Iteration über Array fixierter Größe, Test $(x_1 - y_1)^2 + (x_2 - y_2)^2 = 5$ für bekannte x_i, y_i

- Ansätze (mit Beispielen)

  - DSL: CPLEX für LP/MIP, minizinc für FD
  - EDSL: gecode für FD in C++, ersatz für SAT in Haskell

[fragile,environment=slide]

Randbemerkung: ASP/SAT Confusion

- Answer Set Programming: Marek und Truszczyński, 1999 `http://xxx.lanl.gov/pdf/cs/9809032`
- ASP definiert eine Semantik für erweiterte logische Programme (mit Negation, Disjunktion)
- diese Semantik kann (falls das Universum endlich ist)
  durch Übersetzen nach SAT realisiert werden
  (SAT-Solver wird dabei ggf. mehrfach aufgerufen)
- ASP wird häufig beworben als Front-End (DSL) für SAT,
  das geht, hat aber nichts mit ASP zu tun

`http://www.imn.htwk-leipzig.de/~waldmann/etc/untutorial/asp/`
[fragile,environment=slide]

Constraints für (baum)strukturierte Daten

- Sprache CO4, Dissertation von A. Bau `http://abau.org/co4.html`
- Constraint `c :: P -> U -> Bool` *in Haskell*
- ... das geht in `ersatz` auch?
  Ja — aber nur für $U =$ `Bool`, `[Bool]`, usw., und auch dafür nicht vollständig:
  `if a == b then c else d && e`
  $\Rightarrow$ `ite (a === b) c (d && e)`
- CO4 gestattet für P *und* U:
  - algebraische Datentypen (`data`)
  - pattern matching (`case`)

[fragile,environment=slide]

SAT-Kodierung von Bäumen

- kodiere (endl. Teilmengen von) algebraischen Datentypen
  `data U = C1 T11 .. T1i | C2 T21 .. T2j | ..`
- durch Baum mit Grad $\max(i, j, \ldots)$,
  in jedem Knoten die FD-Kodierung des Konstruktor-Index
- ```
  e = case (x :: U) of
        C1 .. -> e1 ; C2 .. -> e2
  ```
  übersetzt in $\bigwedge_k (\text{index}(x) = k) \Rightarrow (e = e_k)$

Spezifikation, Implementierung, Korrektheit, Anwendungsfälle (Terminations-Analyse, RNA-Design), Messungen, Verbesserungen, Erweiterungen.
siehe Publikationen auf `http://abau.org/co4.html`
[fragile,environment=slide]
  ```

Themen für Master-Projekt und -Arbeit

- Verbesserung von CNFs

  durch Analyse von Klausel(teil)mengen mit BDDs

  (ersetzen durch erf.-äquivalente kleinere Menge)

- SAT-Kodierung für Anzahl-Constraints für `ersatz`

  Implementieren, Messen (Parameter einstellen), vergleichen mit Minizinc-Benchmarks in Gecode

- Formulieren von Matrix-Constraints zur Terminationsanalyse mit Minizinc, Lösen mit Gecode, vergleichen mit SAT- und SMT (QFBV)

- autotool-Erweiterungen (Aufgaben zur Vorlesung)

- industrielle Anwendungen