Constraint-Programmierung Vorlesung Sommersemester 2009, 2012

Johannes Waldmann, HTWK Leipzig

26. Juni 2012

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 $P, Q, R, S \in \mathbb{N}$
  $W(abab) = f_a(f_b(f_a(f_b(0)))), \dots$
- 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 (teilw. autotool)
- Projektarbeit (?), Klausur (?)

mögl. Projektthemen:

- SAT-Kodierungen (Testfälle für eine Bibliothek, die von A. Bau entwickelt wird)
- SAT zum Lösen von Endspielproblemen (`http://senseis.xmp.net/?Havannah`)
- Generierung von SMT-Benchmarks aus Terminationsproblemen
- Test und Erweiterung eines SMT-Solvers
  `https://github.com/jwaldmann/satchmo-smt`

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.

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*: $\mathrm{Mod}(M) = \{b \mid \forall F \in M : b \models F\}$

Der Folgerungsbegriff

eine Formel F *folgt aus* einer Formelmenge M:

- Notation: $M \models F$, Definition: $\mathrm{Mod}(M) \subseteq \mathrm{Mod}(F)$

Beispiele/Übung (beweise!)

- $\{x, \neg y\} \models x \vee y$
- $\forall M : (\mathrm{Mod}(M) = \emptyset) \iff (\forall F : M \models F)$.

  (aus einer widersprüchlichen Formelmenge folgt jede Formel)
- $\forall M, F : (M \models F) \iff (\mathrm{Mod}(M) = \mathrm{Mod}(M \cup \{F\}))$

  (Hinzunahme einer Folgerung ändert die Modellmenge nicht)

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:

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

Modellierung durch SAT

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

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

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

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`

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`

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.

also auch $|G| = \text{Poly}(|F|)$

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):

- transformiere $(x_1 \leftrightarrow x_2) \leftrightarrow (x_3 \leftrightarrow x_4)$

nach Tseitin-Algorithmus.

- Halb-Adder (2 Eingänge x, y, 2 Ausgänge r, c)

  $(r \leftrightarrow (\neg(x \leftrightarrow y))) \wedge (c \leftrightarrow (x \wedge y))$

- Voll-Adder (3 Eingänge, 2 Ausgänge)

Beispiele zur SAT-Modellierung

- Hausaufgabe (leicht): N-Damen-Problem
- Hausaufgabe (schwer): Rösselsprung ($=$ Hamiltonkreis)
- von n Variablen sind genau k wahr (bei Formelgröße $\sim n \cdot k$)
- Puzzles aus *Logisch* (Dt. Rätselverlag)
- Puzzles von `http://www.janko.at/`

Vorgehen bei Modellierung:

- welches sind die Variablen, 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?

SAT: Anwendungen

SAT-Kodierungen

Weil SAT NP-vollständig ist (Ü: Definition) kann man jedes Problem aus NP nach SAT übersetzen,
d.h. in Polynomialzeit eine Formel konstruieren, die genau dann erfüllbar ist, wenn das Problem eine Lösung hat — und man kann aus der erfüllenden Belegung eine Lösung rekonstruieren.
Beispiele:

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

SAT-Kodierung: Independent Set

Def: Graph $G = (V, E)$, Menge $M \subseteq V$ heißt unabhängig, falls $\forall x, y \in M : xy \notin E$.
Entscheidungsproblem:

- Eingabe: (G, k)
- Frage: existiert unabhängige Menge M in G mit $|M| \geq k$?

Optimierungsproblem:

- Eingabe: G
- Ausgabe: möglichst große unabhängige Menge in G

Ist das Optimierungsproblem schwieriger (aufwendiger) als das Entscheidungsproblem? (Hier: nein.)

SAT-Kodierung: Anzahl-Constraints

$\mathrm{count}_{\geq k}(p_1, \ldots, p_n) =$ wenigstens k der n Variablen sind wahr.

- Fallunterscheidung nach der ersten Variable:

$$\begin{aligned} &\mathrm{count}_{\geq k+1}(p_1, p_2, \ldots, p_n) \\ =&p_1 \wedge \mathrm{count}_{\geq k}(p_2, \ldots, p_n) \vee \mathrm{count}_{\geq k+1}(p_2, \ldots, p_n). \end{aligned}$$

- Induktionsanfänge?
- Implementierung mit Hilfsvariablen für jeden Teilausdruck (entspricht Tseitin-Transformation), insgesamt $k \cdot n$ viele.

(Übung: definiere $\mathrm{count}_{=k}, \mathrm{count}_{\leq k}$)
Anwendung: größte unabh. Menge von Springern?

SAT-Kodierung: Vertex Cover

- Def: Graph $G = (V, E)$,

  Menge $M \subseteq V$ heißt Knotenüberdeckung,

  falls $\forall x \in V \setminus M : \exists y \in M : xy \in E$.

- Entscheidungsproblem: (G, k), Frage: … $|M| \leq k$
- „Anwendung": kleinstes Vertex Cover von Damen auf Schachbrett
- Anwendung: Plazierung von Versorgungs- oder Überwachungsknoten in einem Netzwerk

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

SAT-Kodierung: Färbung von Graphen

- Knotenfärbung

  Übung: Definiere *chromatische Zahl* $\chi(G)$

  Übung: welche Aussage kann man (mit polynomiellem Aufwand in $|G|$ und c) SAT-kodieren, welche nicht:

  $\chi(G) \le c, \chi(G) \ge c$

- Kantenfärbung (Ramsey-Probleme)

  Def: $R(s_1, \ldots, s_c) := \min\{n \mid$ jede Kanten-c-Färbung des K_n enthält einen K_{s_1} der Farbe 1 oder ... oder einen K_{s_c} der Farbe c $\}$.

  Ü: beweise mittels SAT-Kodierung: $R(4, 4) \ge 18$, $R(3, 3, 3) \ge 17$ (und Gleichheit durch Nachdenken)

Suche nach symmetrischen Lösungen

falls ein Problem zu groß für den Solver ist:

- weitere Constraints hinzufügen ...
- die effektiv die Anzahl der Variablen verkleinern.
- d. h.: nur nach symmetrischen Lösungen suchen.

Def (allgemein): f ist Symmetrie von M, falls $f(M) \sim M$.
Beispiel: rotationssymmetrische Aufstellungen von Figuren auf Schachbrett (Vertex Cover, Hamiltonkreis).

Neue Schranke für Van der Waerdens Funktion

$W(r, k) =$ die größte Zahl n, für die es eine r-Färbung von $[1, 2, \ldots, n]$ gibt ohne einfarbige arithmetische Folge.
Satz (überhaupt nicht trivial): alle diese Zahlen sind endlich.
Einige Schranken erst vor kurzer Zeit verbessert, durch SAT-Kodierung, siehe Heule, M. J. H: *Improving the Odds: New Lower Bounds for Van der Waerden Numbers.* March 4, 2008. `http://www.st.ewi.tudelft.nl/sat/slides/waerden.pdf`

Software zur SAT-Kodierung

Überblick

Softwarepaket *satchmo* (SAT encoding monad)
`https://github.com/jwaldmann/satchmo`

- kapselt zugrundeliegenden Solver (minisat)
- (monadische) Verwaltung von Variablen (State), Klauseln (Writer) und Belegungen (Reader)
- boolesche Unbekannte, Kombinatoren
- Relationen, Graphen
- Zahlen (unär, binär), Polynome

vergleichbar: `http://rise4fun.com/Z3Py`
(Übung: Unterschiede?)

Technische Grundlagen

- Datentypen für Literal, Klausel, Formel
- Solver als externer Prozeß:
  - Renderer für Dimacs-Format
  - Aufruf eines externen Solvers (minisat)
  - Parser für Ausgabe des Solvers
- *oder* Solver über API:
  - Klauseln in RAM schreiben
  - Solver aufrufen
  - Belegung aus RAM auslesen

Automatische Hilfsvariablen

im Quelltext: Haskell-Namen (`x`), für Solver: Nummern (`1`)

```
import Satchmo.Boolean
import Satchmo.Solve
test :: IO ( Maybe Bool )
test = solve $ do
    x <- boolean -- Allokation einer Variablen
    assert [ not x ]
    return $ decode x
```

Realisierung:

- `boolean :: SAT Boolean` (ist Aktion)
- automatisches Weiterzählen (StateMonad für Nummer)
- WriterMonad für CNF (Klauseln)
- ReaderMonad für Resultat (Belegung)

Realisierung logischer Funktionen

```
and :: [ Boolean ] -> SAT Boolean
and xs = do
  y <- boolean -- Hilfsvariable lt. Tseitin
  forM xs $ \ x -> do
      assert [ not y, x ]
  assert $ y : map not xs
  return y
```

Übung:

- Typ von `forM` (benutze `:i forM` in `ghci` oder `http://haskell.org/hoogle/`)
- `or`
- `xor` (Anzahl der `True` ist ungerade)

Dekodierung mit Typklassen

- Typklasse Decode
- feste Instanz (Satchmo.Boolean → Prelude.Bool)
- generische Instanzen (Paare, Listen, Arrays, Maps)

(vereinfachte Darstellung)

```
type Decoder a = Reader (Map Literal Bool) a
class Decode c a where
    decode :: c -> Decoder a
instance Decode Boolean Bool where
    decode x = ... -- benutzt Minisat-API
instance (Decode c a) => Decode [c] [a] where
    decode xs = forM xs decode
```

Relationen

```
relation :: ( Ix a, Ix b )
  => ((a,b),(a,b)) -> SAT ( Relation a b )
instance (Ix a, Ix b) =>
  Decode ( Relation a b ) ( Array (a,b) Bool )
product :: ..=> Relation a b -> Relation b c
  -> SAT ( Relation a c )
implies :: ..=> Relation a b -> Relation a b
  -> SAT Boolean
```

Anwendungen:

```
transitive r = do
    r2 <- product r r ; implies r2 r
```

Übung: eine schwach zusammenhängende Relation; eine s.z. Halbordnung, die keine totale Ordnung ist.

SAT-Solver in Paketmanagern

- installierte Pakete: $A.2$,...
- zu installierende Pakete: B, ...
- verfügbare Pakete (auf Server): $B.3, B.4$,...
- Abhängigkeiten: $B.3$ erfordert $C(\geq 1.5)$,...
- Konflikte: $C.1.5\#A(< 2.1)$,...

gesucht: konfliktfreier Install-Plan

- Michael Schröder: *Using SAT for Package Dependencies*, FOSDEM 2008
  `http://en.opensuse.org/openSUSE:Libzypp_satsolver`
- Mancoosi (managing software complexity)
  `http://www.mancoosi.org/` (2008-2011)

Pentomino

Es gibt 12 *Pentominos*: zusammenhängende Figuren aus je 5 Einheitsquadraten.
(Henry Ernest Dudeny, 1907; Solomon W. Golomb, 197*)

- Kann man damit ein Schachbrett ohne das 2×2-Mittelquadrat überdecken?

(und viele ähnliche Fragen dieser Art.)
Lösung durch SAT-Modellierung.
Variablen? Constraints?

Schwierige Fälle für die SAT-Kodierung

- wenn das Problem nicht in NP ist, dann hat es keine polynomiell große Kodierung.
- wahrscheinliche Beispiele: Suchprobleme mit (exponentiell) tiefen Bäumen, z. B. Verschiebepuzzles (Rushhour, Atomix, Lunar Lockout (?))
- Markus Holzer, Stefan Schwoon: *Assembling Molecules in Atomix in Hard*, TCS 2003,
  `http://dblp.uni-trier.de/rec/bibtex/journals/tcs/HolzerS04`
- PSPACE-hardness ist offen für Lunar Lockout

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)
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)
  - Backtrack (zu letztem Decide)
  - Backjump (zu früherem Decide)

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)

Beweise für Nichterfüllbarkeit

- bisher: Interesse an erfüllender Belegung $m \in \operatorname{Mod}(F)$ (= Lösung einer Anwendungsaufgabe)
- jetzt: Interesse an $\operatorname{Mod}(F) = \emptyset$.

  Anwendungen: Schaltkreis C erfüllt Spezifikation $S \iff \operatorname{Mod}(C(x) \neq S(x)) = \emptyset$.

Solver rechnet lange, evtl. Hardwarefehler usw.

- $m \in \operatorname{Mod}(F)$ kann man leicht prüfen

  (unabhängig von der Herleitung)
- wie prüft man $\operatorname{Mod}(F) = \emptyset$?

  (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: Klausel C entsteht, indem Klauseln C_1, C_2 nach nach der gemeinsamen Variablen y *resolviert* werden,
- Schreibweise: $C = C_1 \oplus_y C_2$
- Satz: $\{C_1, C_2\} \models C_1 \oplus_y C_2$.

Resolution und Unerfüllbarkeit

Satz: F in CNF nicht erfüllbar $\iff$ es gibt eine *Resolutions-Ableitung* der leeren Klausel.
Beweis:

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

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.

Resolution, Bemerkungen

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

DPLL: Lernen von Konfliktklauseln

bei jedem Konflikt:

- (minimale) Ursache (d. h. partielle Belegung) feststellen
- Implementierung: (partieller) Implikationsgraph
  - Knoten: Literal mit decision-level
  - Kanten: Unit-Progagation-Schritte
- Negation der Konfliktursache als neue Klausel hinzufügen

conflict-driven backtracking:

- Backjump zum zweit-tiefsten decision level der gelernten Konfliktklausel

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

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`

Binäre Entscheidungsgraphen (ROBDDs)

Motivation

gesucht ist eine Datenstruktur, die eine aussagenlogische Formel *kanonisch* repräsentiert und für die aussagenlog. Operationen *effizient* ausführbar sind.
Ansätze:

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

Lösung: bessere Darstellung für $\mathrm{Mod}(F)$
Literatur: Kroening, Strichman: Decision Procedures, 2.4.

Darstellung von Modellmengen

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

- Blätter: 0, 1

  repräsentiert leere (bzw. volle) Modellmenge $\emptyset$ bzw. $\{\emptyset\}$
- innere Knoten t
  - Variable v
  - Kinder (Bäume) l, r
    alle Variablen in l und r sind größer als v

  repräsentiert Modellmenge
  $[t] = \{\{(v,0)\} \cup b \mid b \in [l]\} \cup \{\{(v,1)\} \cup b \mid b \in [r]\}$

Für jede Formel F existiert Baum B mit $[B] = \mathrm{Mod}(F)$.

Entscheidungsbäume mit Sharing

ausgehend von eben definiertem Baum: konstruiere DAG (gerichteten kreisfreien Graphen):

- Blätter zusammenfassen: nur zwei Blätter 0,1
- alle Knoten mit gleichem (v, l, r) zusammenfassen
- alle Knoten mit (v, l, r) und $l = r$ durch l ersetzen

reduziertes geordnetes binäres Entscheidungsdiagramm (ROBDD)
Randal E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Transactions on Computers, C-35(8):677-691, 1986

Eigenschaften von ROBDDs

- kanonisch
- Erfüllbarkeit, Allgemeingültigkeit trivial
- Operationen? (folgende Folien)
- Größe ist abhängig von Variablenreihenfolge (Beispiel)
- es gibt Formeln, für die jede Variablenreihenfolge ein exponentiell großes ROBDD liefert (Beispiel)

Operationen mit ROBDDs

binäre boolesche Operation f:

- auf Blättern 0,1 Wert ausrechnen
- auf Knoten mit gleichen Variablen: gemeins. Fallunterscheidung
- auf Knoten mit versch. Variablen: Fallunterscheidung in kleinerer Variablen

der Trick ist: 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

OBDD-Anwendung: Modelle zählen

typische Anwendung von OBDD …

- *nicht* Erfüllbarkeit (, denn …)
- sondern das Zählen von Modellen:

$$|\mathrm{Mod}(0)| = 0, |\mathrm{Mod}(1)| = 1,$$
$$|\mathrm{Mod}(v,l,r)| = |\mathrm{Mod}(l)| + |\mathrm{Mod}(r)|$$

diese Zahlen bottom-up dranschreiben: Linearzeit
dabei beachten, daß bei der Konstruktion evtl. Variablen verschwunden sind

ROBBD-Implementierungen

```
import qualified Prelude ; import OBDD
import qualified Data.Set as S

let t = or [ unit "x" Prelude.True
           , unit "y" Prelude.False ]
number_of_models (S.fromList ["x", "y"]) t
```

`http://hackage.haskell.org/package/obdd-0.2`

- Namen der Prelude-Funktionen für `Bool` werden verwendet für `OBDD v`
- viele denkbare Verbesserungen, z. B. Cache für alle Teilformeln und Teil-DAGs

Prädikatenlogik

Plan

(für den Rest der Vorlesung)

- Prädikatenlogik (Syntax, Semantik)
- konjunktive Constraints in verschiedenen Bereichen

  z. B. Terme, Gleichungen und Ungleichungen

  auf Zahlen (ganze, rationale, reelle).
- beliebige Boolesche Verknüpfungen

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

Syntax der Prädikatenlogik

- Signatur
  - Relations-
  - Prädikatsymbole
- Term (Funktionssymbolen mit Argumenten)
- Formel
  - Relationssymbol mit Argumenten,
  - Boolesche Verknüpfungen davon
  - Quantoren

  gebundenes und freies Vorkommen von Variablen

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 $(S, b) \models F$
Erfüllbarkeit, Allgemeingültigkeit (Def, Bsp)

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

- Einschränkung des Universums

  (äquivalent: Erweiterung jeder Formel um Axiome, die das gewünschte Universum festlegen)
- Einschränkung der Formelsyntax
  - nur bestimmte Quantoren, nur an bestimmten Stellen (im einfachsten Fall: gar keine)
  - nur bestimmte Verknüpfungen (Negation nur für atomare Formeln, Verknüpfung nur durch $\wedge$)

Termgleichungen

Einführung

- Signatur:
  - Funktionssymbole beliebig
  - Relationssymbol nur $=$
- Universum: Terme (d.h. Leibniz-Axiom für $=$)
- Formel: Konjunktion von Gleichungen (zwischen Termen mit Variablen)

Anwendungen: Verarbeitung symbolischer Ausdrücke

- logische Programmierung (Prolog)
- Typprüfung, Typinferenz (für polymorphe Funktionen)

Unifikation—Begriffe

- Signatur $\Sigma = \Sigma_0 \cup \ldots \Sigma_k$,
- $\operatorname{Term}(\Sigma, V)$ ist kleinste Menge T mit $V \subseteq T$ und $\forall 0 \le i \le k, f \in \Sigma_i, t_1 \in T, \ldots, t_i \in T : f(t_1, \ldots, t_i) \in T$.
- Substitution: partielle Abbildung $\sigma : V \to \operatorname{Term}(\Sigma, V)$, so daß kein $v \in \operatorname{dom} \sigma$ in $\operatorname{img} \sigma$ vorkommt,
- Substitution σ auf Term t anwenden: $t\sigma$
- Produkt von Substitutionen: so definiert, daß $t(\sigma_1 \circ \sigma_2) = (t\sigma_1)\sigma_2$

Unifikation—Definition

Unifikationsproblem

- Eingabe: Terme $t_1, t_2 \in \operatorname{Term}(\Sigma, V)$
- Ausgabe: eine allgemeinster Unifikator (mgu): Substitution σ mit $t_1\sigma = t_2\sigma$.

allgemeinst $=$ minimal bzgl. der Prä-Ordnung
$\sigma_1 \le \sigma_2 \iff \exists \tau : \sigma_1 \circ \tau = \sigma_2$
Satz: jedes Unifikationsproblem ist entweder gar nicht oder bis auf Umbenennung eindeutig lösbar

Unifikation—Algorithmus

$\mathrm{mgu}(s,t)$ nach Fallunterscheidung

- s ist Variable: ...
- t ist Variable: symmetrisch
- $s = f(s_1, s_2)$ und $t = g(t_1, t_2)$: ...

Bemerkungen:

- korrekt, übersichtlich, aber nicht effizient,
- es gibt Unif.-Probl. mit exponentiell großer Lösung,
- eine komprimierte Darstellung davon kann man aber in Polynomialzeit ausrechnen.

Übung zur Prädikaten-Logik

- Die folgende Formel E ist erfüllbar.

$$(\forall x : P(x, f(x))) \wedge (\forall y : \neg P(y,y))$$
$$\wedge (\forall u : \forall v : \forall w : ((P(u,v) \wedge P(v,w)) \rightarrow P(u,w))$$

  Bestimmen Sie ein Modell mit möglichst kleinem Universum.
- Geben Sie eine (kleine) erfüllbare Formel F an, so daß jedes Modell von F wenigstens 5 Elemente enthält.
- Beweisen Sie: Erfüllbarkeit ist bereits für Formeln ohne Funktionssymbole unentscheidbar.
  Geben Sie dazu ein Verfahren an, das aus einer beliebigen Formel F eine Formel G *ohne*
  Funktionssymbole erzeugt, so daß gilt: F besitzt Modell $\iff$ G besitzt Modell. Hinweis: zusätzliche Relationssymbole.
  Wenden Sie das Verfahren auf die o.g. Formel E (und das von Ihnen gefundene Modell) an.
- Beweisen Sie, daß Erfüllbarkeit und Allgemeingültigkeit entscheidbar sind für Formeln,
  - die nur nullstellige Rel.-Symbole und keine Fkt.-Symbole enthalten,
  - die nur einstellige Rel.-Symbole und keine Fkt.-Symbole enthalten.

  Hinweis: es genügt, endliche Modelle bis zu einer gewissen Größe zu betrachten.

(vgl. Uwe Schöning: Logik für Informatiker, Spektrum)

Übung zu Termgleichungen

- sind die Substitutionen $\sigma_1 = \{X \mapsto Y\}$ und $\sigma_2 = \{Y \mapsto X\}$ vergleichbar bzgl. $\leq$?
- $\leq$ auf Substitutionen ist keine Halbordnung.
  Die durch $\leq$ definierte Äquivalenzrelation ist ...
- vervollständigen Sie den Unifikationsalgorithmus, vergleichen Sie mit `http://dfa.imn.htwk-leipzig.de/cgi-bin/gitweb.cgi?p=ws11-cb.git;a=blob;f=kw56/Type/Constraint.hs;hb=master`
- wenden Sie den Algorithmus an zur Lösung von

$$f(X_1, f(X_2, f(X_3, a))) = f(f(X_2, X_2), f(f(X_3, X_3), f(a, a)))$$

Lineare Gleichungen und Ungleichungen

Syntax, Semantik

- lin. (Un-)Gleichungssystem $\rightarrow$ (Constraint $\wedge)^*$ Constraint
- Constraint $\rightarrow$ Ausdruck Relsym Ausdruck
- Relsym $\rightarrow$ $=$ | $\leq$ | $\geq$
- 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: $\bigwedge_i \sum_j a_{i,j} x_j \geq b_i$

$$\begin{array}{rcl} x - 4y & \geq & 0 \\ & \dots & \end{array}$$

- 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)

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:
$U_x^{\downarrow} := \{A \mid (x \geq A) \in U\}$, $U_x^{\uparrow} := \{B \mid (x \leq B) \in U\}$,
$U_x^{-} = \{C \mid C \in U, C$ enthält x 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 $\mathbb{Z}$

LP ist in P (Ellipsoid-Verfahren), aber IP ist NP-vollständig:

- SAT $\leq_P$ IP,
- jedes IP besitzt obere Schranke für Größe einer Lösung;

deswegen gibt es keinen effizienten Algorithmus (falls P $\neq$ NP)

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

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: $\sum_i x_{i,j} = 1, \sum_j x_{i,j} = 1$?

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: $\forall 1 \leq i \neq j \leq n : u_i - u_j + n x_{i,j} \leq n - 1$

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?

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 …

Übungen zu lin. Gl./Ungl.

- lin. Gl. und Gauß-Verfahren in GF(2)

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

- lin. Ungl. mit CLP lösen

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

- (mixed) integer programming mit CBC
  `http://www.coin-or.org/projects/Cbc.xml`

  z. B. TSP, min/max-Probleme

(Integer/Real) Difference Logic

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

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),

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.

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 $D(x) = -\infty$

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 $i \xrightarrow{w_{i,j}} j$ mit $d(i) + w_{i,j} < d(j)$
    $d(j) := d(i) + w_{i,j}$
jederzeit gilt die *Invariante*:

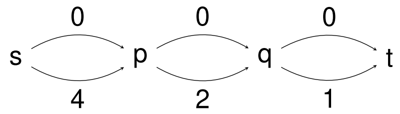
- $\forall x \in V$: es gibt einen Weg von s nach x mit Gewicht $d(x)$
- $\forall x \in V : D(x) \leq d(x)$.

verbleibende Fragen:

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

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)

Polynomgleichungen

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`

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/`)

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 $x \in \mathbb{Z} \wedge x \geq 0$

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)

Lösbarkeit in ganzen Zahlen: ist nicht entscheidbar

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

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 $[p_0(x), p_1(x), \dots]$
- Satz: $|\{x \mid P(x) = 0 \wedge a < x \leq b\}| = \sigma(P,a) - \sigma(P,b)$

Quantoren-Elimination über $\mathbb{R}$

- Sturmsche Ketten verallgemeinern für Polynome in mehreren Variablen
- Variablen der Reihe nach entfernen
- Realisierung durch
  - Tarski (1930): Laufzeit $\exp(\exp(\dots(|V|)\dots))$
  - 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`

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 $M =$
$\{(x_1,\dots,x_i) \mid \exists x_{i+1} \in \mathbb{Z}, \dots, x_k \in \mathbb{Z} : P(x_1,\dots,x_k) = 0\}$
Beispiele:

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

Abschluß-Eigenschaften? (Durchschnitt, Vereinigung, …)

Diophantische Mengen

Satz (Davis, Matiyasevich, Putnam, Robinson; $\approx$ 1970)
M diophantisch $\iff$ M 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`

Presburger-Arithmetik

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!)

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?

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 $\mathrm{Mod}(F) := \{m \in \mathbb{N}^k \mid \{x_1 \mapsto m_1, \dots, x_k \mapsto m_k\} \models F\}$
für F mit freien Var. $x_1, \dots, x_k$,

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.

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$\}$,
- $\{c(x, y) \mid x + x = y\}$, $\{c(x, y, z) \mid x + y = z\}$,
- $\{c(x, y) \mid x \le y\}$, $\{c(x, y) \mid x < y\}$

Formeln und Modellmengen

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

- $\mathrm{Mod}(\mathrm{False}) = \emptyset$, $\mathrm{Mod}(\neg F) = \dots$
- $\mathrm{Mod}(F_1 \land F_2) = \mathrm{Mod}(F_1) \cap \mathrm{Mod}(F_2)$
- $\mathrm{Mod}(\exists x_i.F) = \mathrm{proj}_i(\mathrm{Mod}(F))$

  Projektion entlang i-ter Komponente: $\mathrm{proj}_i : \mathbb{N}^k \to \mathbb{N}^{k-1}$:

  $(x_1, \dots, x_k) \mapsto (x_1, \dots, x_{i-1}, x_{x+1}, \dots, x_k)$

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?

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:

  $\mathrm{Lang}(A) = \{w \mid A$ akzeptiert $w\} = \{w \mid I \circ \delta'(w) \circ F^{\mathsf{T}} \neq \emptyset\}$

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

Komplexität: $|Q| = |Q_1| \cdot |Q_2|$ (hier: Zeit = Platz)

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)
- $I = \{I_1\}, F = 2^{Q_1 \setminus F_1}$
- $\delta(c)(M) = \dots$

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)

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

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

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)

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?

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)

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 $\le f(|x|)$ 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.

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 $x = 2^{2^n} \land xy = z$
  (die naive Implementierung ist aber zu groß)
- und Trick (“the device preceding Theorem 8”):
  $H(x_1) \land H(x_2)$ ist äquivalent zu
  $\forall x \forall y : (x = x_1 \lor \ldots) \to \ldots$

Uninterpretierte Funktionen (UF)

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

$$(\forall x : g(g(x)) = x \land g(h(x)) = x) \to (\forall x : h(g(x)) = x)$$

gehört zu *jeder* Theorie (mit passender Signatur),
weil sie zur Theorie der *freien Termalgebra* gehört.

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`

Terme

- Signatur Σ:
  Menge von Funktionssymbolen mit Stelligkeiten
- $t \in \mathrm{Term}(\Sigma)$: 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 $t \in \mathrm{Term}(\Sigma)$ ein Wert $[t] \in D$ zugeordnet
- $\mathrm{Term}(\Sigma)$ ist selbst eine Σ-Algebra

Gleichheit von Termen

In jeder Algebra gelten diese Formeln:

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

(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?

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:

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

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 $F_i = g(F_{i1}, \dots, F_{ik}), F_j = g(F_{j1}, \dots, F_{jk})$ 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.

Gleichheits-Constraints

(Dis)Equality Constraints (Bsp, Def)

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

(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%2Feq_diamond%2Feq_diamond10.smt2

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

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

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

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.

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.)

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.

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.

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`

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.

Testklausur KW23

SAT, CNF, Kodierung

- Geben Sie eine konjunktive Normalform für $a \leftrightarrow (b \to c)$ an und begründen Sie die Äquivalenz.
- Spezifizieren Sie die Tseitin-Transformation:

  was ist Eingabe E, was ist Ausgabe A,
  welche Beziehungen bestehen zwischen E und A.
- Beschreiben Sie eine effiziente SAT-Kodierung für „von den Variablen $x_1, \ldots, x_n$ ist eine ungerade Anzahl wahr“.

  Verwenden Sie Hilfsvariablen,
  spezifizieren Sie deren Bedeutung.

ROBDD

Definieren Sie (jeweils Wort und Bedeutung): B, R, O.

Für die boolesche Funktion G:

$$\mathrm{bin}(x_1, x_0) \geq \mathrm{bin}(y_1, y_0)$$

(Größenvergleiche von Binärzahlen,
x_0 und y_0 sind jeweils LSB)

- Bestimmen Sie das ROBDD von G
  für die Variablenreihenfolge x_0, x_1, y_0, y_1.
- Geben Sie besser Variablenreihenfolge an

  (zu der ein kleineres ROBDD von G gehört, das Sie auch angeben sollen)

Lineare Ungleichungen

Für das Ungleichungssystem:
$x_1 + x_2 \leq x_3 \wedge x_2 \leq x_1 + 7 \wedge x_1 + 10 \leq x_3 + x_2$,

- Entscheiden Sie die Lösbarkeit des Ungleichungssystems, indem Sie (in dieser Reihenfolge) die Variablen x_1, x_2, x_3 durch das Fourier-Motzkin-Verfahren eliminieren.
- Geben Sie eine bessere Eliminationsreihenfolge an.

Differenz-Logik

Die Erfüllbarkeit einer Konjunktion von Differenz-Constraints kann reduziert werden auf eine Eigenschaft eines Graphen.

- Definieren Sie alle Parameter des Graphen.
- Definieren Sie die zu testende Eigenschaft.

(jeweils allgemein und am Beispiel
$x_1 \leq x_2 + 3 \wedge x_2 \leq x_3 + 4 \wedge x_3 \leq x_1 - 7 \wedge x_4 + 5 \leq x_2$)

In welchen Anwendungsproblemen treten solche Constraints auf? (Was ist dabei die Bedeutung der Variablen, was die Bedeutung der Zahlen?)

EC/UF

EC: Equality-Constraints

- Definieren Sie die Syntax von EC (durch Angabe einer Signatur)
- Geben Sie je ein Beispiel für ein erfüllbares und ein nicht erfüllbares EC mit zwei Variablen an.
- Beweisen Sie: (*) für alle $C \in$ EC mit n Variablen gilt:

  C besitzt Modell $\iff$ C besitzt Modell mit Universum der Größe $\leq n$.

UF: EC mit uninterpretierten Funktionen

- Geben Sie ein UF-Constraint an, für das (*) nicht gilt.
- (Zusatz): gilt UF(*) mit einer anderen Schranke (als $\leq n$)?

Presburger-Arithmetik (PA)

Wir definieren eine Folge $F_0, F_1, \ldots$ von Formeln durch
$F_0(x, z) = (x + 1 = z), F_{k+1}(x, z) = \exists y : F_k(x, y) \wedge F_k(y, z)$.

- Für welche z gilt $F_3(0, z)$?
- Beschreiben Sie die Bedeutung von $F_k(x, z)$ durch eine nicht rekursive Formel, bei der Sie zusätzlich die Funktionssymbole für Multiplikation und Potenzierung verwenden dürfen.
- Reduzieren Sie die Formelgröße von F_k (innerhalb PA), indem Sie eine äquivalente Definition von F_{k+1} angeben, in der F_k nur einmal vorkommt.

Programme mit Arrays

Definition

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

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`

Array-Logik (Beispiel mit Quantoren)

(aus Kroening/Strichman, Kap. 7)
Gültigkeit der Invariante $\forall x : 0 \le x < i \Rightarrow a[x] = 0$ für die Schleife

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

folgt aus Gültigkeit der Formel

$$
\begin{aligned}
& (\forall x : (0 \le x \land x < i) \Rightarrow \mathrm{select}(a, x) = 0) \\
\land \quad & (a' = \mathrm{store}(a, i, 0)) \\
\Rightarrow \quad & \dots
\end{aligned}
$$

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)
```

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)
```

Definition von Array-Formeln

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

- I (index guard): boolesche Kombination von linearen Gln. und Ungln. über $i_1, \dots, i_k$,
- V (value constraint): boolesche Komb. von Gleichheitsconstraints für Array-Elemente.

  Index-Ausdrücke in select und store $\in \{i_1, \dots, i_k\}$.

NNF: $\land/\lor$ beliebig, $\neg$ nur ganz unten, keine anderen Operatoren.

Vereinfachung von Array-Formeln:

- store entfernen:
  ersetze $\mathrm{store}(a, i, e)$ durch a' und füge Constraints hinzu:
  $\mathrm{select}(a', i) = e \land (\forall j : j \neq i \to \mathrm{select}(a', j) = \mathrm{select}(a', j))$.
- 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

Übung Array-Formeln/NNF

- $P \to Q$ in NNF?
- $\neg\forall i : P(i) \Rightarrow Q(i)$ auf *Pränexform* bringen (Quantor(en) außen)
- Regel „ersetze $\neg\forall i : \dots$ “ 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 $\neq x$

Kombination von Theorien

Motivation

(Kroening/Strichman: Kapitel 10)

- Lineare Arithmetik + uninterpretierte Funktionen:

  $(x_2 \ge x_1) \land (x_1 - x_3 \ge x_2) \land (x_3 \ge 0) \land f(f(x_1) - f(x_2)) \neq f(x_3)$

- Bitfolgen + uninterpretierte Funktionen:

  $f(a[32], b[1]) = f(b[32], a[1]) \land a[32] = b[32]$

- Arrays + lineare Arithmetik

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

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 $T_1 \oplus T_2$

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

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)

Nelson-Oppen (I)

(Wdhlg) Formel/Ausdruck
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.

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 $T_i \models (\phi_i \to x_i = y_i)$, dann Gleichung $x_i = y_i$ zu allen ϕ_j hinzufügen,…
- bis sich nichts mehr ändert, dann ϕ erfüllbar

(Beispiele)
(Beweis)

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.

SMT, DPLL(T)

SMT

SMT = Satisfiability modulo Theory

- Aufgabenstellung:

  Erfüllbarkeitsproblem für beliebige boolesche Kombinationen von atomaren Formeln aus einer Theorie

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

  Erweiterung des DPLL-Algorithmus: DPLL(T)

Literatur: Kroening/Strichman, Kap. 12

DPLL(T)

Ansatz: für jedes Atom $F = P(t_1, \ldots, t_k)$
eine neue boolesche Unbekannte $p_F \leftrightarrow F$, dann:

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

Ideen zur Realisierung/Verbesserung:

- naiv (SAT): belegen, testen, backtrack.
- DPLL (SAT): partiell belegen, t., b., propagieren (, lernen)
- DPLL(T) (SMT): Test in Theorie T, Theorie-Propagation.

DPLL(T), Beispiel

T = LRA (linear real arithmetic)
durch Tseitin-Transformation als CNF darstellen
(zusätzliche boolesche Variablen)
DPLL(T) benutzt

- T-Solver für Konjunktion von vollst. belegten Klauseln

  (z. B. Simplexverfahren)
- T-Propagation

  (z. B. $x \leq y \wedge y \leq z \Rightarrow x \leq z$)
- T-Lerner

  bei Nichterfüllbarkeit liefert T-Solver eine „Begründung“ (kleine nicht erfüllbare Teilmenge)

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.php`

…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.

DPLL(T) Übung

- ein DPLL(T)-Beispiel für T= Gleichheitslogik oder Differenzlogik durchrechnen.
- dabei ggf. T-Entscheidungsverfahren wiederholen.
- welche Möglichkeiten bestehen dabei für T-Propagation? T-Explanation?

DPLL/CDCL

CDCL: Spezifikation

conflict driven clause learning

- wenn (nach Unit-Propagation) ein Konflikt auftritt,
- dann wird eine neue Klausel C' berechnet („gelernt“) und zur Formel F hinzugefügt
- sowie ein früherer Entscheidungspunkt p als Ziel für Backjump bestimmt.
- so daß $F \models C'$, d. h. die Bedeutung wird nicht geändert,
- und C' in p sofort eine Unit-Propagation bewirkt

CDCL: Implementierung, Übung

(Kroening/Strichman Abschn. 2.2.4)

```
cl := aktuelle Konfliktklausel;
while (cl enthält >= 2 Literale >= aktuell)
  lit := zuletzt zugewiesenes Literal aus cl;
  var := die Variable aus lit;
  ante := die Klausel, die in der Propagation
      benutzt wurde, welche  lit  belegt hat;
  cl := Resolve (ante, cl, var);
```

Beispiel: $-1 \vee 2, -1 \vee 3 \vee 5, -2 \vee 4, -3 \vee -4, 1 \vee 5 \vee -2, 2 \vee 3, 2 \vee -3, 6 \vee -5, \ldots$. (weitere Klauseln mit anderen Var.)
Entscheidungen: $\ldots, -6, \ldots, 1$. Welche Klausel wird gelernt? Zu welchem Punkt wird zurückgekehrt?
(vlg. auch Beispiel in Folien v. Nieuvenhuis.)

Bitblasting

Idee

Constraints für ganze Zahlen

- plus, minus, mal, min, max, …
- boolesche Verknüpfungen

die Zahlen werden „gesprengt“ (blast = Explosion)
und man rechnet mit ihren Bits (CNF-SAT)
(nach Festlegung der Bitbreite der Zahlen)

$x = [x_0, x_1, \ldots, x_{w-1}]$
Notation hier: LSB ist links

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:

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

dafür CNF ohne Hilfsvariablen!

Subtraktion

- Subtraktion als Umkehrung der Addition
  $x - y = z \iff x = z + y$
- negative Zahlen als Zweierkomplement (geeignete Behandlung von Überläufen)

Vergleiche

Größer: $x = [x_0, \ldots] > [y_0, \ldots] = y$, falls

- $(x_0 > y_0) \wedge (\mathrm{tail}(x) = \mathrm{tail}(y))$
- oder $\mathrm{tail}(x) > \mathrm{tail}(y)$

Gleich: $x = [x_0, \ldots] = [y_0, \ldots] = y$, falls

- $|x| = |y| = 0$
- oder $(x_0 = y_0) \wedge (\mathrm{tail}(x) = \mathrm{tail}(y))$

Damit $x > y$ mit linear vielen Variablen und Klauseln möglich. Welches sind die besten Faktoren?

Multiplikation

$[x_0,\ldots]\cdot[y_0,\ldots]$,

- Schulmethode: $x_0 \cdot y + 2 \cdot [x_1,\ldots]\cdot y$ rekursiv
- Überläufe im Resultat früh erkennen, nur mit nötigen Stellen rechnen
- Karatsuba-Multiplikation:

  $(p+qB)(r+sB) = pr + \underbrace{(ps+qr)}_{=t}B + qsB^2$

  berechne $t = (p+q)(r+s) - pr - qs$ mit nur 3 Multiplikationen.

SAT-Formeln für binäre Multiplikation sind sehr schwer: Resultatbit k hängt von *allen* Eingabebits $0,\ldots,k$ ab.

Bitshift

$x \ll y = z$ ... um einen *unbekannten* Betrag y! (Hausaufgabe.)

Unärkodierung

- als *monotone* Bitfolge

  $3 = [1,1,1,0,0,0,0,0]$

  $x = [x_0,\ldots]$ mit $x_0 \ge x_1 \ge \ldots$.
- Übung: Min, Max, Größer.
- unäre Addition durch Sortiernetze (!)

  (= Idee der minisat-Autoren Een/Sörenssen)
- aktuelle Anwendung: Codish et al.: *Exotic Semiring Constraints*, `http://smt2012.loria.fr/`

CNF-Kompression (Motivation)

vereinfachte Annahme: SAT-Solver ist schneller, wenn

- weniger (Zusatz-)Variablen
- weniger Klauseln

führt zu der Aufgabe:

- Eingabe: boolesche Formel F
- Ausgabe: kleine zu F (erfüllbarkeits)äquivalente CNF.

Anwendungen:

- F ist Halbadder, Volladder
- F ist Addition, Multiplikation usw. für geringe Bitbreite

Aber: mehr Klauseln ⇒ mehr Propagationen

CNF-Kompression (Implementierung)

zu gegebener Formel F bestimme Menge der *Prim-Implikanten* $P = \{P_1,\ldots\}$,
das sind *minimale* Klauseln P_i mit $F \to P_i$
(minimal bezüglich Inklusion von Literalen)

dann bestimme eine kleinste Teilmenge $M \subseteq P$ mit $\bigwedge M \to F$.
das ist eine ganzzahlige lineare Optimierungsaufgabe (Übung: welche?), kann durch entsprechende Constraint-Solver gelöst werden (glpsol, scip, clp)

Diskussion: `http://groups.google.com/group/minisat/msg/012bb4712c7e90a7`

Anwendung: `https://github.com/jwaldmann/satchmo/blob/master/Satchmo/Binary/Op/Common.hs#L118`

Zusammenfassung, Ausblick

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)

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:

- Zahlen: lineare Gleichungen, lineare Ungleichungen, Polynome, Presburger-Arithmetik
- uninterpretierte Funktionen, Arrays

Prädikatenlogik: allgemeine Verfahren:

- Kombinationen von Theorien (Nelson-Oppen)
- Kombination von Theorie und SAT (DPLL(T))

Themen zum Weiterarbeiten

- optimierte SAT-Kodierung für binäre Addition, Multiplikation
- Optimierung von Parametern für Constraint-Solver (innerhalb eines Terminationsbeweisers)
- „didaktisch wertvolle“ Implementierungen (d. h. in Haskell) wichtiger Algorithmen (DPLL, CDCL, Fourier-Motzkin, DPLL(T))
- ... und daraus abgeleitete autotool-Aufgaben
- Verbesserung von satchmo-smt (bitblasting SMT solver)
- SAT-Kodierungen in der Bioinformatik (RNA folding)
- SAT-Kodierungen für Zweipersonen(end)spiele
- SAT-Solver auf (hoch)paralleler Hardware