Constraint-Programmierung
Vorlesung
Sommersemester 2009, 2012

Johannes Waldmann, HTWK Leipzig

6. Juli 2014

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 $\rightarrow \mathbb{N}$
- wenn $z_1 \rightarrow z_2$, dann $W(z_1) > W(z_2)$.

Parameter der Bewertung werden durch Constraint-System beschrieben.

CP-Anwendung: Polynom-Interpretationen

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

  Beispiel-Rechnung: $abab \to baab \to baba \to bbaa$
- Bewertung W durch lineare Funktionen
  $f_a(x) = Px + Q, f_b(x) = Rx + S$ mit $P, Q, R, S \in \mathbb{N}$

  $W(abab) = f_a(f_b(f_a(f_b(0)))), \ldots$
- monoton: $x > y \Rightarrow f_a(x) > f_a(y) \wedge f_b(x) > f_b(y)$
- kompatibel mit Programm: $f_a(f_b(x) > f_b(f_a(x))$
- resultierendes Constraint-System für P, Q, R, S,
- Lösung mittels Z3

Wettbewerbe für Constraint-Solver

- für aussagenlogische Formeln:

  `http://www.satcompetition.org/`

  (SAT $=$ satisfiability)

- für prädikatenlogische Formeln

  `http://smtcomp.sourceforge.net/`

  (SMT $=$ satisfiability modulo theories)

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

- Termination und Komplexität

  `http://www.termination-portal.org/wiki/Termination_Competition`

Gliederung der Vorlesung

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

Organisatorisches

- jede Woche 1 Vorlesung + 1 Übung
- Übungsaufgaben (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:

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

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:

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

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`

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 $|G| = \mathsf{Poly}(|F|)$

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:

  $\chi(G) \leq c, \chi(G) \geq c$

- 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: $R(4,4) \geq 18$, $R(3,3,3) \geq 17$ (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 Variabl
    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 $\rightarrow$ 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] wher
    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),

`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 Ausgabe b und halt.
- (backtrack) falls F eine b-Konfliktklausel enthält, dann:
  - falls Keller leer, dann Ausgabe UNSAT und halt.
  - 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 l enthält: $\mathrm{DPLL}(b \cup \{(l, 1)\})$.
- (decide) sonst wähle $v \notin \mathrm{dom}\, b$, push$(v)$, und $\mathrm{DPLL}(b \cup \{(v, 0)\})$.

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)

DPLL mit CDCL (Plan)

conflict driven clause learning, Lernen von Konfliktklauseln

bei jedem Konflikt:

- (minimale) Ursache (d. h. partielle Belegung) feststellen
- Negation der Konfliktursache als neue Klausel K hinzufügen

Eigenschaften:

- 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

DPLL mit CDCL (Einzelheiten)

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

- Knoten: $\mathrm{dom}(b) \cup \{k\}$
- Kanten: $v \to v'$, falls v' durch eine Propagation belegt wurde mit einer Unit-Klausel, die v enthält

(durch Decide belegten Variablen haben keine Vorgänger.)
Konflikt-Ursache: die Menge der Decide-Knoten,
von denen aus die Konfliktvariable erreichbar ist.
gelernte Klausel: $(\neg b(v_1) \vee \ldots \neg b(v_n)$,
wobei $\{v_1, \ldots, v_n\}$ die Konfliktursache ist
Satz: diese Klausel K folgt aus der Formel F
...d.h. $\mathrm{Mod}(F) = \mathrm{Mod}(F \cup \{K\})$

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:

$$\frac{x \vee y, y \vee \neg z}{x \vee \neg z}$$

- 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`

Binäre Entscheidungsgraphen (ROBDDs)

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`

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)

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 Bit bei) integer multiplication

für Ordnung $y_{2n} > \ldots > y_1$ betrachte Informationsfluß (Anzahl der Kanten) zw. $\{y_{2n}, \ldots, y_{n+1}\}$ und $\{y_n, \ldots, y_1\}$.

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

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

- alle Knoten in Hashtabelle ("hash consing"),

  garantiert Reduktion

- gemeinsames hash consing über mehrere ROBDDs ($\Rightarrow$ "BDD base") erlaubt Nachnutzung von Arbeit

ROBDD-Anwendungen: Modelle zählen

typische Anwendungen von ROBDD ...

- (*nicht* Erfüllbarkeit, denn ...)
- Äquivalenz
- Zählen von Modellen:

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

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

```
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

vergleiche: Implementierung in C:

http://sourceforge.net/projects/buddy/

Übung BDD

- ROBDD für die Zählfunktion

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

- obere Schranke für Anzahl der BDD für n Variablen mit k Knoten ($2^{\text{Anzahl der Bits}}$)

  vergleiche mit Anzahl Boolescher Funktionen von n Variablen, Schlußfolgerung?

- Bsp. ROBDD-Operation

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

  Modellierung (Formel), Implementierung mit `obdd`

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

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: $\bigwedge_i \sum_j a_{i,j} x_j \geq b_i$

$$x - 4y \geq 0$$

$$\ldots$$

- 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)

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

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?

min und max als MIP

- kann man den Max-Operator durch lin. UngIn 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 $i \overset{w_{i,j}}{\to} j$ mit $d(i) + w_{i,j} < d(j)$

$d(j) := d(i) + w_{i,j}$

jederzeit gilt die *Invariante*:

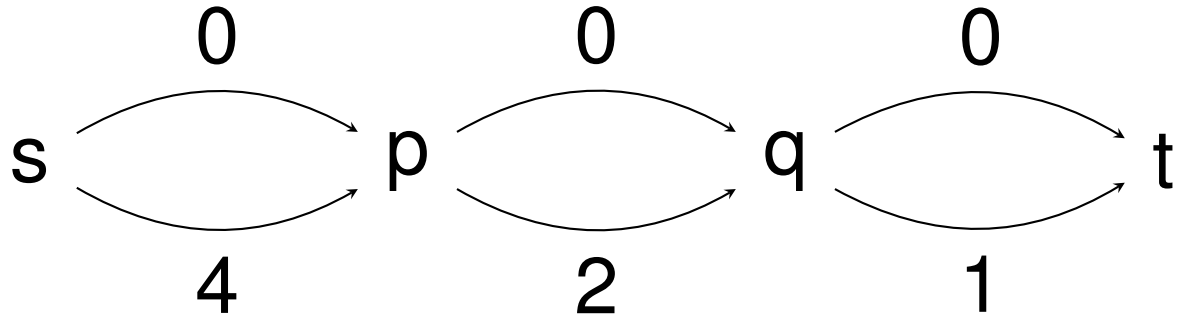
- $\forall x \in V$: es gibt einen Weg von s nach x mit Gewicht $d(x)$
- $\forall x \in V : D(x) \leq d(x)$.

verbleibende Fragen:

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

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)

Lösbarbeit 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 $[p_0(x), p_1(x), \ldots]$

- 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 $\exp(\exp(\ldots(|V|)\ldots))$
  - 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`

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, . . .)

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 $\mathrm{Mod}(F) := \{m \in \mathbb{N}^k \mid \{x_1 \mapsto m_1, \ldots, x_k \mapsto m_k\} \models F\}$

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.

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

Formeln und Modellmengen

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

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

  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?

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 $w = w_1 \ldots w_n \in \Sigma^*$:

  $\delta'(w) = \delta(w_1) \circ \ldots \circ \delta(w_n)$

- A akzeptiert w gdw. $\exists p \in I : \exists q \in F : \delta'(w)(p, q)$

- Menge (Sprache) der akzeptierten Wörter:

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

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)
- $I = I_1 \times I_2, F = F_1 \times F_2$
- $\delta(c)((p_1, p_2), (q_1, q_2)) = \ldots$

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)

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)

Automaten (Operationen: Projektion)

Eingabe: Automat $A_1 = (\Sigma^k, Q_1, I_1, \delta_1, F_1)$, Zahl i

Ausgabe: A mit $\mathrm{Lang}(A) = \mathrm{Lang}(\mathsf{proj}_i(A_1))$

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

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 $\leq f(|x|)$ 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.

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

  $\forall x \forall y : (x = x_1 \vee \ldots) \rightarrow \ldots$

SMT, DPLL(T)

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

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.)

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)
```

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

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)

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

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.

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?

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

Finite Domain Constraints

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

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*

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.).

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

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

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

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?

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.

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

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)

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 $x_1 = 2$)

Übung Konsistenz

- weitere Konsistenzbegriffe (Hyperkante, Pfad)
- Entwurf autotool-Aufgabe zu FD-Solver (im Vergleich zu DPLL)
- Konsistenzbegriffe anwenden auf Kodierung von SAT als FD-Problem
- (Wiederholung Fourier-Motzkin, ergänze Quelltext, so daß erfüllende Belegung berechnet wird)

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.

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 $x_0 \geq x_1 \geq \ldots$.

- 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 $\Rightarrow$ 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 $F \rightarrow P_i$
(minimal bezüglich Inklusion von Literalen)

dann bestimme eine kleinste Teilmenge $M \subseteq P$ mit $\bigwedge M \rightarrow F$.
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`