Constraint-Programmierung
Vorlesung
Sommersemester 2009

Johannes Waldmann, HTWK Leipzig

22. Juni 2009

Organisatorisches

- jede Woche 1 Vorlesung + 1 Übung
- Übungsaufgaben autotool
- Projekte (langfristig)
- Prüfung $\approx$ Projektverteidigung

Gliederung

- Aussagenlogik
- CNF-SAT-Constraints
- DPLL-Solver
- PBC, QBF
- Prädikatenlogik
- Finite-Domain-Constraints
- naive Lösungsverfahren, Konsistenzen
- lineare Gleichungen, Ungleichungen, Polynomgleichungen
- Termgleichungen, Unifikation
- Kodierungen nach CNF-SAT (FD, Zahlen)
- SMT, DPLL(T)

Projektthemen

- Multiplikation, Faktorisierung (SAT-Encoding)
- (max,plus)-Gleichungssysteme (SMT/Ungl.)
- `http://www.morpionsolitaire.com/`
- (End-)Spiele als QBF
- Constraint-Solver im Suse-Paketmanager
- Puzzles von `http://janko.at/` z. B.:
  - `http://www.janko.at/Raetsel/Nonogramme/`
  - `http://www.janko.at/Raetsel/Zeltlager/`

  (Lösen und Generieren)
- Optimizer-Puzzles `http://www.logic-masters.de/ //OM2009/seiten/anleitung.php`
- „ernsthafte Optimierungs-Aufgaben“, Leistungsvergleich mit evol. Algorithmen.

Aussagenlogik: Syntax

aussagenlogische Formel:

- elementar: Variable $v_1, \ldots,$ (Konstante $0, 1$)
- zusammengesetzt: durch Operatoren
  - einstellig: Negation
  - zweistellig: Konjunktion, Disjunktion, Implikation, Äquivalenz

Aussagenlogik: Semantik

- Wertebereich $\mathbb{B} = \{0, 1\}$, Halbring $(\mathbb{B}, \vee, \wedge, 0, 1)$
- *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 f ein *Modell* von f, Schreibweise: $b \models f$
- *Modellmenge* $\mathrm{Mod}(F) = \{b \mid b \models F\}$
- F *erfüllbar*, wenn $\mathrm{Mod}(F) \neq \emptyset$

Normalformen (DNF, CNF)

Definitionen:

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

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 und äquivalente Formel G' in DNF.

Erfüllbarkeits-Äquivalenz

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

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.) $F = L \vee R$, 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-Transpormation (Übung)

Übungen:

- transformiere $(x_1 \leftrightarrow x_2) \leftrightarrow (x_3 \leftrightarrow x_4)$
- Halb-Adder, Voll-Adder

Überblick

Softwarepaket *satchmo* (SAT encoding monad)
`http://hackage.haskell.org/cgi-bin/hackage-scripts/package/satchmo`

- Grundlagen
- (monadische) Verwaltung von Variablen (State monad), Klauseln (WriterMonad) und Belegungen (ReaderMonad)
- boolesche Unbekannte, Kombinatoren
- Relationen, Graphen
- Zahlen

CNF, Solver

`Satchmo.Data` und `Satchmo.Solve`

- Datentypen für Literal, Klausel, Formel
- Renderer für Dimacs-Format
- Aufruf eines externen Solvers (minisat)
- Parser für Ausgabe des Solvers

Automatische Hilfsvariablen

- Datentyp Boolean, automatisches Weiterzählen
- WriterMonad für CNF
- ReaderMonad für Resultat

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

Realisierung logischer Funktionen

```
module Satchmo.Boolean.Op where ...

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

Dekodierung mit Typklassen

- Typklasse Decode
- generische Instanzen (Paare, Listen, Arrays)

```
class Decode c a | c -> a where decode :: c -> Deco
type Decoder a = Reader ( Map Literal Bool ) a

instance ( Decode c a ) => Decode [c] [a] where
    decode = mapM 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 :: ( Ix a , Ix b, Enum b, Ix c )
  => Relation a b -> Relation b c
  -> SAT ( Relation a c )
transitive :: ( Enum a, Ix a )
           => Relation a a -> SAT Boolean
transitive r = do
    r2 <- product r r
    implies r2 r
```

Anwendung: Graphen

```
knight n s = do
    a <- relation ((1,1),(n,n))
    m <- atmost s $ do
            i <- indices a ; return $ a ! i
    assert [m]
    sequence_ $ do
        p <- indices a
        return $ assert $ do
            q <- indices a
            guard $ p == q || reaches p q
            return $ a!q
    return $ decode a

reaches (px,py) (qx,qy) =
    5 == (px - qx)^2 + (py - qy)^2
```

SAT-Kodierung von Binärzahlen

- Addition
- Vergleich
- Subtraktion
- Multiplikation

Anwendungen: Ramanujans Taxi, Faktorisierung

SAT-Kodierungen

Weil SAT NP-vollständig ist, 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.
Beispiele

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

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

SAT-Kodierung: Anzahl-Constraints

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

- Fallunterscheidung nach erster Variablen:

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

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

(entspr. $\text{count}_{=k}, \text{count}_{\leq k}$)
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: $\ldots |M| \leq k$
Anwendung: kleinstes Vertex Cover von Damen?

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

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`

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 $x_1 \vee x_3$, Belegung $x_1 = 0$, Folgerung: $x_3 = 1$
- bei Konflikt: Backtrack/Backjump

DPLL: Heuristiken

Minisat `http://minisat.se/`

- Wahl der nächsten Entscheidungsvariablen
  (am häufigsten in aktuellen Konflikten)
- Lernen von Konflikt-Klauseln (siehe nächste Folie)
- Vorverarbeitung (Variablen und Klauseln eliminieren)
  (siehe Übung)

DPLL: Konfliktklauseln

bei jedem Konflikt:

- (minimale) Ursache (d. h. Belegung) feststellen
- Negation davon als neue Klausel hinzufügen

unsat, Resolution

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

$$\frac{(x_1 \vee \ldots \vee x_m \vee y), (\neg y \vee z_1 \vee \ldots \vee z_n)}{x_1 \vee \ldots \vee x_m \vee z_1 \vee \ldots \vee z_n}$$

Beweis: 1. Korrektheit ($\Leftarrow$), 2. Vollständigkeit ($\Rightarrow$)

- moderne SAT-Solver können solche Beweise ausgeben
- es gibt nicht erfüllbare F mit (exponentiell) großen Resolutionsbeweisen (sonst wäre NP $=$ co-NP)
- falls Konflikt durch Unit Progagation: (kurzer) Resolutionsbeweis $\Rightarrow$ neue Klausel

Quantifizierte Boolesche Formeln

Syntax: eine QBF hat die Form $\underbrace{Q_1 v_1 \ldots Q_n v_n}_{\text{Präfix}} \underbrace{M}_{\text{Matrix}}$,

wobei $Q_1, \ldots, Q_n \in \{\forall, \exists\}$ und M eine aussagenlogische Formel ist (die die Variablen $v_1, \ldots, v_n$ enthalten kann)

Semantik: $\mathrm{val}(M) = \mathrm{val}(M, \emptyset)$ mit
$\mathrm{val}(\forall x M, b) = \mathrm{val}(M, b \cup \{(x, 0)\}) \wedge \mathrm{val}(M, b \cup \{(x, 1)\})$
$\mathrm{val}(\exists x M, b) = \mathrm{val}(M, b \cup \{(x, 0)\}) \vee \mathrm{val}(M, b \cup \{(x, 1)\})$

QBF-Beispiele mit fester Quantortiefe

- chromatische Zahl
- kleinstes Vertex Cover
- minimales Sudoku
- kleinster Schaltkreis
- ...

Schema: $\exists z : P(z) \wedge \forall z' : z' < z \Rightarrow \neg P(z')$

Zweipersonenspiele

Satz: PSPACE $=$ APTIME (alternierend)
Anwendung (Projekt!): Zweipersonenspiele (mit polynomieller Länge)
$\exists s_1 \forall w_2 \ldots \forall w_n : (s_1, w_2, \ldots, w_n)$ ist gültige Zugfolge und Resultat ist verloren für Weiß.

Komplexität

Dieses Problem QBF-SAT ist PSPACE-vollständig:

- Eingabe: eine QBF F
- Frage: ist $\mathrm{val}(F) = 1$

Bemerkung: QBF $\in$ PSPACE (alle Belegungen durchprobieren).
Aufgabe: es ist ein (Polynomialzeit-)Verfahren anzugeben, das zu jeder polynomiell platzbeschränkten Maschine M und Eingabewort w eine QBF F konstruiert, so daß $w \rightarrow_M^* \mathtt{OK} \iff \models F$.
Problem: F darf nur polynomiell groß sein, die Rechnungen von M können aber exponentiell lang sein!

Lösung

Beweis: kodiere $F_k(p,q) \iff p \to^{\leq 2^k} q$, es gibt einen Pfad der Länge $\leq 2^k$ von p nach q, durch QBF der Größe $O(k)$.
$F_0(p,q) := (p = q) \vee (p \to^1 q)$
$F_{k+1}(p,q) :=$
$\exists m \forall x \forall y (((x = p \wedge y = m) \vee (x = m \wedge y = q)) \Rightarrow F_k(x,y))$

Anwendung (Projekte!): (lange) Ableitungen (Schleifen) in längenerhaltenden Ersetzungssystemen, in Verschiebepuzzles (Lunar Lockout, Sokoban,...)

Satz von Savitch

Satz (Savitch): NPSPACE(f) $\subseteq$ DPSPACE(f^2).
(N: nichtdeterministisch, D: deterministisch)
Beweis: für $M \in$ NPSPACE konstruiere Formel F wie vorhin
(Größe?)
dann teste $F \in$ QBF-SAT mit einer DPSPACE-Maschine.

Pseudo-Boolean Constraints

Idee:

- Klausel: $v_1 \vee \overline{v}_2 \vee v_3$
- Ungleichung: $v_1 + \overline{v_2} + v_3 \geq 1$

pseudo-Boolesches Constraint (PBC) ist Ungleichung

$$c_1 v_1 + \ldots + c_n v_n \geq d,$$

mit Koeffizienten $c_1, \ldots, c_n, d \in \mathbb{Z}$ für Unbekannte $v_1, \ldots \in \mathbb{B} = \{0, 1\}$.
PBC-Optimierungsaufgabe: Menge von PBC und Zielfunktion $z_1 v_1 + \ldots + z_n v_n$
Lösung davon ist Belegung, die alle Constraints erfüllt und Zielfunktion maximiert.

Wdhlg. Prädikatenlogik (Syntax)

Signatur Σ	▶ Relationssymbole $R_1, \ldots,$ ▶ Funktionssymbole $f_1, \ldots,$
Term t	▶ Variable x ▶ $f(t_1, \ldots, t_k)$
Formel F	▶ $R(t_1, \ldots, t_k)$ ▶ $F_1 \wedge F_2; F_1 \vee F_2; F_1 \rightarrow F_2; F_1 \leftrightarrow F_2; \neg F_1$ ▶ $\forall x : F_1; \exists x : F_1$

Menge der freien, gebundenen Variablen einer Formel

geschlossene (closed) Formel: keine freien Variablen

Prädikatenlogik (Semantik)

zu Signatur $\Sigma = (\Sigma_F, \Sigma_R)$ passende *Struktur*:

- ein Grundbereich M
- zu jedem $f \in \Sigma_{F,k}$ ein $[f] : M^k \to M$
- zu jedem $r \in \Sigma_{R,k}$ ein $[r] : M^k \to \mathbb{B}$

Belegung: $b : \text{Var} \to M$

Definiere

- Wert eines Terms t in Struktur M unter Belegung b
- Wert einer Formel F in Struktur M unter Belegung b
- Wert einer geschlossenen Formel F in Struktur M

Theorien

Theorie einer Struktur $=$ alle Formeln, die in dieser Struktur wahr sind.

Beispiel: Signatur $\Sigma_F = \{0_0, S_1, +_2\}, \Sigma_R = \{=_2\}$,
Struktur: $\mathbb{N}$ mit Null, Nachfolger, Plus, Gleichheit.
Theorie T der natürlichen Zahlen mit Addition.
Formel $(\forall x : \forall y : \forall z : x + (y + z) = (x + y) + z) \in T$.

typische (algorithmische) Fragen:

- Erfüllbarkeit (gehört $\exists$-Abschluß von Formel F zu T?)
- Wahrheit (gehört geschlossene Formel F zu T?)

Lineare Gleichungen

eingeschränkte Syntax ($\approx$ Signatur)
Terme:

- Summand $\rightarrow$ ganze Zahl $\cdot$ Variable
- Summe $\rightarrow$ Summand ($+$ Summand)*

Formeln:

- Gleichung $\rightarrow$ Summe $=$ Summe
- Gleichungssystem $\rightarrow$ Gleichung ($\wedge$ Gleichung)*

Struktur: rationale Zahlen, übliche Operationen.
Erfüllbarkeit ist entscheidbar (durch Gauß-Elimination).
Jede Lösungsmenge ist (leer oder) affin linear.

Lineare Ungleichungen

vgl. lineare Optimierung (engl. linear programming)
Terme wie bei Gleichungen, aber Formeln:

- Ungleichung $\rightarrow$ Summe $\geq$ Summe
- Ungleichungssystem $\rightarrow$ Unleichung ($\wedge$ Ungleichung)*

Eigenschaften:

- Erfüllbarkeit ist entscheidbar (z. B. Simplex-Algorithmus)
- Lösungsmenge ist Durchschnitt von Halbräumen
- Lösungsmenge ist (leer oder) konvex

Verfahren nach Fourier und Motzkin (I)

transformiert ein lin. Unglsys. in ein erfüllbarkeitsäquivalentes mit einer Variablen weniger.
Beispiel: aus $(x + y \leq 1 \wedge 2x + 3y \geq 0)$ wird $(y \geq -2)$
Das ist (wie schon Gauß für Gleichungssyst.) ein Verfahren zur *Quantor-Elimination*
$(\exists x, y : x + y \leq 1 \wedge 2x + 3y \geq 0) \iff (\exists y : y \geq -2)$

Verfahren nach Fourier und Motzkin (II)

Elimination einer Variablen:

- wähle eine Variable x,
- stelle alle Ungl, in denen x vorkommt, nach x um
- wenn alle die Form "$x \leq \ldots$" haben, dann ...
- wenn $x \leq G_1 \wedge \ldots$ und $x \geq K_1 \wedge \ldots$, dann ...

Fragen:

- Korrektheit
- Komplexität

Weitere Verfahren für Lin. Ungl.

- Fourier-Motzkin
  einfach, ineffizient
- Simplex-Methode
  recht einfach, meist effizient, *aber keine Polynomialzeit*
- Ellipsoid-Methode
  kompliziert, Polynomialzeit

Implementierungen

GNU Linear Programming Kit
`http://www.gnu.org/software/glpk/` `http://www.cs.unb.ca/~bremner/docs/glpk/glpk_faq.txt`

- linear programming (lineare Optimierung)
- (mixed) integer programming (ganzzahlige lin. Opt.)

Schnittstellen:

- Text (GNU MathProg $\subseteq$ AMPL)
- API (C, Java)

Hausaufgabe: installieren, ausprobieren (Text-Schnittstelle)

(Un)Gl. zwischen Polynomen

Erweiterung der Syntax:

- Summand $\rightarrow$ Zahl $(\cdot$ Variable$)^*$

das aus der Schule bekannte „Lösen quadratischer Gleichungen“ ist Quantor-Elimination:
$(\exists x : x^2 - x + a \leq 0) \iff (a \leq 1/4)$.
in mehreren Variablen? — Für Struktur $\mathbb{R}$ entscheidbar

- Gleichungen: Gröbnerbasen
- Ungleichungen: Alfred Tarksi, QEPCAD,

aber für $\mathbb{Z}$ nicht! (Juri Matiasevitch, Julia Robinson)

Fourier-Motzkin für SAT?

(Wiederholung F-M)
$\exists x : K_1 \leq x \wedge \ldots \wedge K_p \leq x \wedge x \leq G_1 \wedge \ldots \wedge x \leq G_q$
$\iff \bigwedge_{1 \leq i \leq p, 1 \leq j \leq q} K_i \leq G_j$

vergleiche mit *Resolution*:

$$\begin{aligned} (A \vee x) \wedge (\neg x \vee B) &\Rightarrow (A \vee B) \\ (\neg A \rightarrow x) \wedge (x \rightarrow B) &\Rightarrow (\neg A \rightarrow B) \\ (\neg A \leq x) \wedge (x \leq B) &\Rightarrow (\neg A \leq B) \end{aligned}$$

Elimination von x durch *vollständige* Resolution
$(A_1 \vee x, \ldots, A_p \vee x)$ mit $(\neg x \vee B_1, \ldots, \neg x \vee B_q)$

Übung: unit propagation als Spezialfall
Projekt: SAT-Solver bzw. Präprozessor bauen

Erweiterungen

- andere Zahlen (nicht nur reelle)
- andere Strukturen (nicht nur Zahlen, (Un)gleichungen)
- mehr Formeln (nicht nur $\wedge$, sondern auch $\vee$)

Mixed Integer Programming

= lineare Ungleichungssysteme für
reelle, *ganze*, *binäre* (d.h. $\{0, 1\}$) Zahlen.

- konkrete Syntax: AMPL (Robert Fourer and David M. Gay and Brian W. Kernighan, A Modeling Language for Mathematical Programming. Management Science 36 (1990) 519-554. `http://www.ampl.com/REFS/abstracts.html#amplmod`)
- Solver: `http://www.ampl.com/`, `http://www.gnu.org/software/glpk/`, `http://scip.zib.de/`
- ist NP-schwer (binäre Zahlen $\rightarrow$ CNF-SAT).

Anwendungsaufgabe

Motivation: (Un)gleichungssysteme über

- $(\{-\infty\} \cup \mathbb{N}, \max, +)$ (arktischer Halbring)
- $(\mathbb{N} \cup \{+\infty\}, \min, \max)$ („fuzzy“ Halbring)

- Term $\rightarrow$ Variable | Zahl | $\max(t_1, t_2)$ | $\min(t_1, t_2)$ | $t_1 + t_2$
- Formel $\rightarrow$ Gleichung. (Ungl. nicht nötig, warum?)

(auch für reelle Zahlen) *nicht* auf lineares Ungl.sys. reduzierbar (sonst P=NP).
Anwendung: Terminationsbeweise mit arktischen/fuzzy Matrizen

Ordnungsconstraints für Matrizen

Wortersetzungssystem R über Σ,
Bsp: $R = \{aa \to aba\}$ über $\Sigma = \{a, b\}$

Matrix-Interpretation (Halbring D, Dimension n)
$[\cdot] : \Sigma \to D^{n \times n}$ für Buchstaben
für Wörter durch Multiplikation: $[x_1, \ldots, x_k] = [x_1] \circ \ldots \circ [x_k]$

zulässig: $\forall x \in \Sigma : [x]_{1,1} \neq 0_D$
kompatibel: $\forall (l, r) \in R : \forall i, j : [l]_{i,j} >_0 [r]_{i,j}$
wobei $p >_0 q \iff (p = 0_D = q) \vee (p > q)$

Beispiel: $D = (\mathbb{N} \cup \{+\infty\}, \min, \max)$,

$$[a] = \begin{pmatrix} 2 & +\infty & 0 \\ 1 & +\infty & 0 \\ +\infty & +\infty & +\infty \end{pmatrix}, [b] = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Lösungsansätze

(= Projektthemen)

- simulieren als MIP, dann lösen mit glpk oder scip
- SMT/QFLRA, QFLIA, dann lösen mit (z. B.) Z3.2
- („bitblasting“, SMT/QFBV oder direkt nach SAT)

Literatur:

- Definition, Competition (SMT/LIB, /COMP, /EXEC)
  `http://www.smtexec.org/exec/?jobs=311`
- „SMT als Buch“: Daniel Kroening und Oliver Strichman:
  Decision Procedures—an Algorithmic Point of View,
  Springer 2008. `http://www.decision-procedures.org/`

Max/Min als MIP

Idee: $\max(x, y) = z$ ist äquivalent zu

$$\begin{aligned} \exists b \in \{0, 1\} : \quad & x \leq z \\ \wedge \quad & y \leq z \\ \wedge \quad & z \leq x + M \cdot b \\ \wedge \quad & z \leq y + M \cdot (1 - b) \end{aligned}$$

dabei M eine *genügend große* Konstante
(Übung: was heißt das?)

aber: je kleiner M, desto besser für die Solver.

DPLL(T)

Ansatz: für jede elementare Formel $F = P(t_1, \ldots, t_k)$
eine neue boolesche Unbekannte $p_F \leftrightarrow F$,
dann SAT-Problem für die p_* lösen.
Ideen:

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

(optimized lazy approach)

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

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)

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

Realisierung:

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

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

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

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

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

Multiplikation

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

- Schulmethode: $x_0 \cdot y + \text{tail}(x) \cdot y$ 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: jedes Resultatbits hängt von *allen* Eingabebits 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)

CNF-Kompression (Motivation)

vereinfachte Annahme: SAT-Solver ist schneller, wenn

- weniger (Zusatz-)Variablen
- weniger Klauseln

führt zu der Aufgabe:

- Eingabe: eine boolesche Formel F
- Ausgabe: eine klein(st)e zu F (erfüllbarkeits)äquivalente CNF.

Anwendungen:

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

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, kann durch entsprechende Constraint-Solver gelöst werden (glpsol, scip)

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

- (wie bisher) Boolesche Kombinationen werden durch DPLL(T) behandelt,
- 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*:

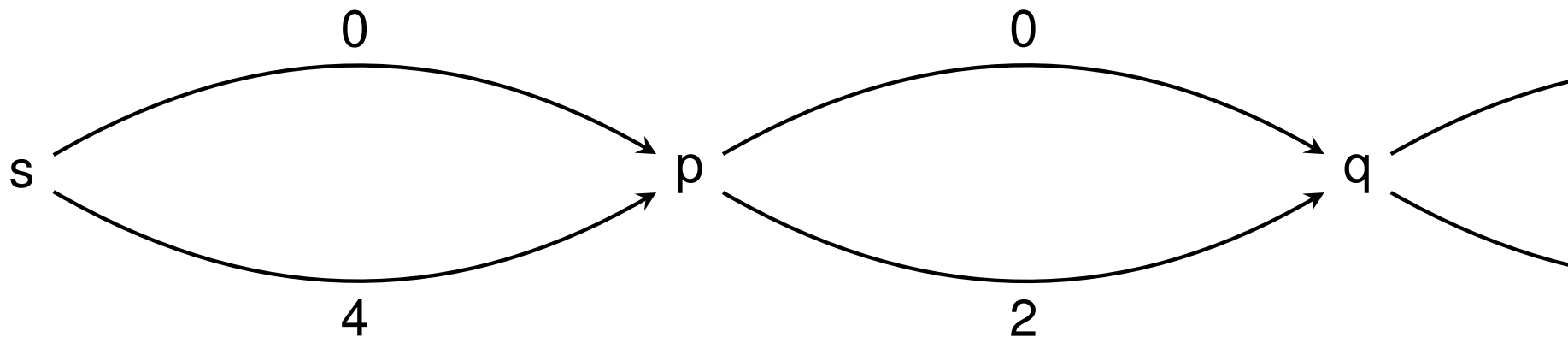
- $\forall x \in V$: es gibt einen Weg von s nach x mit Gewicht $d(x)$
- $\forall x \in V : D(x) \leq d(x)$.

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
  dann testen, ob alle Kanten entspannt sind.
  Wenn nein, dann existiert negativer Kreis. (Beweis?)
- Dijkstra (schneller, aber nur bei Gewichten ≥ 0 korrekt)

Anwendung: Min/Max-Matrixungleichungen

die Halbringoperationen in Min/Max lassen sich (trivial) in RDL/IDL darstellen

Motivation

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

- Formel F selbst?
- Modellmenge Mod(F) als Menge von Belegungen?

Lösung: bessere Darstellung für Mod(F)
Literatur: Kroening, Struchman: 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
- innere Knoten t
  - Variable v
  - Kinder l, r

  repräsentiert Formel $[t] = (\neg v \wedge [l]) \vee (v \wedge [r])$

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 $l = v$ 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).
resultierende Laufzeit für $f(s,t)$ ist $O(|s| \cdot |t|)$.

OBDD-Anwendung: Modelle zählen

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

diese Zahlen bottom-up dranschreiben: Linearzeit

Finite-Domain-Constraints (FD)

Definition

- prädikatenlogische Formel über mehrsortiger Signatur,
- für jede Sorte eine endliche Menge

Beispiele

- magisches Quadrat, Sudoku
- Rösselsprung, n-Damen-Problem
- Graphenfärbungen
- SAT

FD-Lösungsverfahren

Prinzip (vgl. DPLL für SAT)

- entscheiden:
  eine Variable auswählen, für diese alle möglichen Belegungen ausprobieren
- propagieren:
  Menge der möglichen Belegungen anderer Variablen verkleinern

welche Variable wird Entscheidungsvariable?

- Aktivität
- Wertebereich

Konsistenzen (Überblick)

man betrachtet jeweils

- Constraints (log. Formeln) $C(\ldots, x, \ldots)$
- Bereiche $x \in \{e_1, \ldots, e_n\} = E$

Constraint-System heißt ...-konsistent, wenn ausgewählte Constraints und Bereiche zusammenpassen.
Wenn es nicht paßt: Bereiche durch gewisse Regeln verkleinern (bis konsistent oder offensichtlich unlösbar)

- Knotenkonsistenz
- Kantenkonsistenz
- Pfadkonsistenz

Knotenkonsistenz

ein Constraintsystem heißt *knotenkonsistent*, wenn für jede Variable x gilt:

- jeder Wert aus dem Bereich von x ...
- erfüllt alle Constraints, in denen nur x vorkommt.

Kantenkonsistenz

ein Constraintsystem heißt *kantenkonsistent*, wenn für alle Variablenpaare x, y gilt:

- zu jedem Wert a aus dem Bereich von x ...
- gibt es einen Wert b aus dem Bereich von y,
- so daß (a, b) alle Constraints erfüllt, in denen nur x und y vorkommen.

beachte: auch für y, x

Pfadkonsistenz

ein Constraintsystem heißt *pfadkonsistent*, wenn für alle Variablen x, y, z gilt:

- $C_{x,y} \circ C_{y,z} \supseteq C_{x,z}$

wobei $C_{x,y} =$ Menge aller erfüllenden Belegungen (= geordnete Paare) von Constraints, in denen nur x, y vorkommen.

k-Konsistenz

Eine partielle Belegung b von Variablen eines CS heißt *konsistent*, wenn b alle Constraints erfüllt, in denen nur Variablen aus dom b vorkommen.
Ein CS ist k-konsistent, wenn jede konsistente $(k-1)$-Belegung zu einer konsistenten k-Belegung fortgesetzt werden kann.
Genauer: ...
Beziehungen zu Knoten-, Kanten-, Pfadkonsistenz.

Konsistenz unter Kodierungen

Beispiel: FD-CS für Zahlen und SAT-CS für binäre Kodierung.
es ist möglich, daß das Original inkonsistent ist, aber die Kodierung konsistent.
Beispiel: $x \in [0, 1, \ldots, 5]$ unär und binär

(Dis)Equality Constraints (Bsp, Def)

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

(Quelle: Strichman, Rozanov, SMT Workshop 2007,

`http://www.lsi.upc.edu/~oliveras/espai/smtSlides/ofer.ppt`,

`http://www.smtexec.org/exec/benchmarkDetail.php?benchmark=661781`)

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

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

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

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.

Motivation, Definition

Wahrheit/Erfüllbarkeit hängt von Theorie (d.h. Interpretation der Funktionssymbole) ab, Bsp:

$$\forall a, b : a \circ b = b \circ a$$

gilt für Zahlen, aber nicht für Matrizen. Aber

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

ist allgemeingültig in *jeder* Theorie,
weil es in der Theorie der *freien Termalgebra* allgemeingültig ist.

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

Die Relation $=$ wird beschrieben durch das Axiom

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

(in SMT-Sprechweise: *functional consistency*)

- Definition: eine Σ-Algebra heißt *frei*, wenn keine weiteren Gleichheiten gelten.
- Beispiel: jede Termalgebra ist frei.
- Gegen-Beispiel: $\Sigma = \{+, 1\}, D = \mathbb{N}$ ist nicht frei: $(1 + 1) + 1 = 1 + (1 + 1)$.

Logik und Unentscheidbarkeit (I)

Def: das Gültigkeitsproblem der Prädikatenlogik:

- Eingabe: eine prädikatenlogische Formel F über Σ,
- Frage: ist F in Term(Σ) allgemeingültig

Beispiel: Σ: zwei einstellige Funktionsymbole f, g,
ein nullstelliges Funktionssymbol a,
ein zweistelliges Prädikatsymbol R.

$$\forall x, y : R(x, y) \rightarrow \left(\begin{array}{rr} & R(f(x), g(f(f(y)))) \\ \wedge & R(g(x), f(y)) \\ \wedge & R(g(f(f(x))), g(y)) \end{array} \right)$$
$$\wedge \quad R(a, a) \wedge \exists z : R(g(z), g(z))$$

Logik und Unentscheidbarkeit (II)

Satz: Das Gültigkeitsproblem der Prädikatenlogik ist unentscheidbar.
(Kurt Gödel, Alan Turing.
`http://thocp.net/biographies/papers/turing_oncomputablenumbers_1936.pdf`)

Beweis:

- Halteproblem für TM ist unentscheidbar
- Halteproblem $\leq$ Postsches Korrespondenzproblem (PCP)
- PCP $\leq$ Gültigkeitsproblem

verwendet *Reduktion* $A \leq B :\Longleftrightarrow$
$\exists$TM-berechenbares$f : \forall x : x \in A \iff f(x) \in B$.

Entscheidbare Fälle

. . . durch Einschränkung der Position der Quantoren.
in SMT-LIB: nur außen existentiell.

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

```
(benchmark check  :logic QF_UF
:extrafuns ((f U U)(g U U)(x U)(y U))
:formula (not (implies (= x y)
                       (= (f(f(g x))) (f(f(g y)))) ))  )
```

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 $F_i = g(F_{i1}, \ldots, F_{ik}), F_j = g(F_{j1}, \ldots, F_{jk})$ mit $i \neq j$:
  $(f_{i1} = f_{j1} \wedge \ldots \wedge f_{ik} = f_{jk}) \rightarrow f_i = f_j$

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

Motivation

(Kroening/Strichman: Kapitel 10)

- Lineare Arithmetik + uninterpretierte Funktionen:
  $(x_2 \geq x_1) \wedge (x_1 - x_3 \geq x_2) \wedge (x_3 \geq 0) \wedge f(f(x_1) - f(x_2)) \neq f(x_3)$
- Bitfolgen + uninterpretierte Funktionen:
  $f(a[32], b[1]) = f(b[32], a[1]) \wedge a[32] = b[32]$
- Arrays + lineare Arithmetik
  $x = v\{i := e\}[j] \wedge y = v[j] \wedge x > ex > y$

Definition

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

- 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: Entscheidungsverfahren (Constraint-Solver) für T_1, T_2 (einzeln)
- gesucht: . . . für $T_1 \oplus T_2$

Konvexe Theorien

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

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

konvex oder nicht?

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

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 $T_1, \ldots, T_n$:

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

(Beispiele)
(Beweis)

Nelson-Oppen für nicht konvexe Theorien

Falls $T_i \models (\phi_i \rightarrow (x_1 = y_1 \vee \ldots x_k = y_k))$,
dann *split*: für $1 \leq j \leq k$ betrachte $\phi \wedge (x_j = y_j)$
(Beispiel)
(Beweis)

Definition

(SMTLIB: QF_A)

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

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

Store entfernen

gegeben eine ϕ Formel mit Teilausdruck store(a, i, e),
ersetze diesen durch a' und füge Constraints hinzu:
select$(a', i) = e \wedge \forall j \neq i$: select$(a', j) =$ select(a', i).
da es sich um quantorenfreie Formeln handelt: j läuft über alle Index-Ausdrücke, die in der Formel vorkommen.

Spezialfälle

durch beschriebene Transformation kann man Array-Logik reduzieren auf Kombination von

- Index-Logik (z. B. lineare Arithmetik)
- Element-Logik
- uninterpretierte Funktionen.

Lösen mit Nelson-Oppen möglich.
Ähnliche Rechnungen finden (seit vielen Jahrzehnten!) im Schleifen-Optimiere von `gcc` statt (Beispiel).
Mit Array-Logik kann man beweisen, daß der Optimierer recht hat.