SAT Solving und Anwendungen

SAT Modulo Theories (SMT Solving)

Prof. Dr. Wolfgang Küchlin
Rouven Walter, M.Sc. Informatik

Universität Tübingen

1. Februar 2018

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Motivation

- Viele wissenschaftlich und industriell interessante Probleme können in SAT codiert und mit SAT Solvern gelöst werden
- Codierung nach SAT oft problemlos möglich (Hardware, Software, Konfigurationprobleme, Schedulingprobleme, ...)
- Problem: Codierungen nach SAT werden oft zu groß

Beispiel: Lineare Arithmetik

- Um einen arithmetischen Ausdruck zu codieren muss man zuerst ein Addierwerk, und darauf aufbauend einen Multiplizierer in Aussagenlogik modellieren
- Codierung eines n-Bit (Integer-)Multiplizierers ist hart:

n	Anzahl Variablen	Anzahl Klauseln
8	313	1001
16	1265	4177
24	2857	9529
32	5089	17057
64	20417	68929

Motivation

- Für viele Theorien sind bereits Entscheidungsverfahren bekannt, die mehr Domänenwissen einbeziehen als eine Codierung nach SAT
  - **Gleichheitslogik:** Kongruenzhülle, Graph-basierte Algorithmen
  - **Uninterpretierte Funktionen:** Ackermann oder Bryant Reduktion und dann Gleichungslogik
  - **Lineare Arithmetik:** Simplex, Fourier-Motzkin, Omega
  - **Bit Vektoren:** Bit-Flattening
  - **Arrays:** Übersetzung zu uninterpretierten Funktionen
  - **Zeigerlogik:** Modellierung des Speichers als Array
  - **Quantifizierte Formeln:** Quantorenelimination
- Warum also nicht einfach die jeweilige Logik benutzen und Solver für diese Tools verwenden?
- Problem: Kombination von Theorien

Beispiel

$$\underbrace{g(a) = x \wedge (f(g(a)) \neq f(c) \vee g(a) = d)}_{\text{Uninterpretierte Funktionen und Gleichungslogik}} \wedge \underbrace{a < d \wedge f(a) \leq c}_{\text{Lineare Arithmetik}}$$

Grundidee für SMT — 1

- Betrachte das aussagenlogische Skelett einer Formel und benutze SAT Solver um dies zu lösen
- Wenn Skelett bereits UNSAT, dann ist die originale Formel UNSAT
- Ansonsten: Verwende erfüllende Belegung und schicke die Konjunktion der jeweiligen atomaren Formeln an die jeweiligen Theorie Solver
- Versuche, aus fehlgeschlagenen Versuchen zu lernen

Beispiel

$$g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d$$

- Ersetze atomare Formeln durch neue aussagenlogische Formeln:

$$P_1 \land (\neg P_2 \lor P_3) \land \neg P_4$$

- SAT Solver liefert Modell $P_1, \neg P_2, \neg P_4$ zurück
- Konjunktion atomarer Formeln wird an Theory Solver geschickt:

$$g(a) = c \land f(g(a)) \neq f(c) \land c \neq d$$

Grundidee für SMT — 2

Beispiel (ctd.)

- Formel: $g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$
- Skelett: $P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$
- 1. Modell: $P_1, \neg P_2, \neg P_4$
- Theory Solver bekommt: $g(a) = c \wedge f(g(a)) \neq f(c) \wedge c \neq d$

- Theory Solver liefert UNSAT
  - wenn $g(a) = c$ dann gilt $f(g(a)) = f(c)$ und damit $f(c) \neq f(c) \Rightarrow$ Widerspruch
- Dieses Modell wird in Zukunft geblockt (*blocking clause*)
- Schicke an SAT Solver: $\{(P_1), (\neg P_2, P_3), (\neg P_4), (\neg P_1, P_2, P_4)\}$
- SAT Solver liefert Modell $P_1, P_2, P_3, \neg P_4$
- Theory Solver liefert UNSAT
- Neue Formel: $\{(P_1), (\neg P_2, P_3), (\neg P_4), (\neg P_1, P_2, P_4), (\neg P_1, \neg P_2, \neg P_3, P_4)\}$
- SAT Solver liefert UNSAT $\Rightarrow$ Formel ist UNSAT

Fahrplan für heute

1. **Formales**
2. **Interessante Theorien**
   - Gleichheitslogik und uninterpretierte Funktionen
   - Arithmetik
   - Arrays
   - Bit Vektoren
3. **Ansätze zum Entscheiden von SMT Problemen**
   - Eager Approach
   - Lazy Approach
   - Das $\mathcal{T}$-DPLL Framework
4. **Kombination von Theorien**
   - Nelson-Oppen Methode für konvexe Theorien
   - Nelson-Oppen Methode für nicht-konvexe Theorien

Syntax

Definition (Signatur)

Signatur Σ: Menge an *Prädikats-* und *Funktionssymbolen*

- Jedes Symbol hat eine Stelligkeit (Arität)
- Σ^P bzw. Σ^F Menge der Prädikats- bzw. Funktionssymbole
- 0-stellige Funktionssymbole: *Konstanten*
- 0-stellige Prädikatssymbole: *Aussagenlogische Konstanten*

Notation:

- a, b, c: Konstanten
- A, B: Aussagenlogische Symbole
- f, g: nicht-konstante Symbole aus Σ^F
- p, q: nicht-konstante Symbole aus Σ^P

Variablenfreie Terme & Formeln

Quantoren- und variablenfreie Terme und Formeln

$$
\begin{array}{lcll}
t & ::= & c & \text{mit } c \in \Sigma^F \text{ und Arität } 0 \\
 & \mid & f(t_1, \ldots, t_n) & \text{mit } f \in \Sigma^F \text{ und } n > 0 \\
 & \mid & ite(\varphi, t_1, t_2) & \\
 & & & \\
\varphi & ::= & A & \text{mit } A \in \Sigma^P \text{ und Arität } 0 \\
 & \mid & p(t_1, \ldots, t_n) & \text{mit } p \in \Sigma^P \text{ und } n > 0 \\
 & \mid & t_1 = t_2 \mid \bot \mid \top \mid \neg\varphi_1 & \\
 & \mid & \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \leftrightarrow \varphi_2 & \\
 & \mid & \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 &
\end{array}
$$

- *Atomare Formel / Atom*: $A, p(t_1, \ldots, t_n), t_1 = t_2, \bot, \top$
- *Literal:* Atomare Formel oder deren Negation (Notation: ℓ)

Was ist mit Variablen?

Aus technischen Gründen werden Variablen wie Konstanten behandelt. Der Σ-Term $x < y + 1$ ist variablenfrei und x und y werden als Konstanten zu Σ hinzugefügt.

Semantik — 1

Evaluation von Formeln zu einem Wahrheitswert aus $\{\textbf{true}, \textbf{false}\}$ bzgl. eines Modells $\mathcal{A}$

Modell $\mathcal{A}$ für eine Signatur Σ

Paar $(A, (\_)^{\mathcal{A}})$

- A: Universum
- $(\_)^{\mathcal{A}}$ Abbildung der Symbole in Σ
    - $a^{\mathcal{A}} \in A$ (Konstanten)
    - $f^{\mathcal{A}} : A^n \to A$ für $f \in \Sigma^F$ mit Arität n (Funktionen)
    - $B^{\mathcal{A}} \in \{\textbf{true}, \textbf{false}\}$ (Aussagenlogische Konstanten)
    - $p^{\mathcal{A}} : A^n \to \{\textbf{true}, \textbf{false}\}$ für $p \in \Sigma^P$ mit Arität n (Prädikate)

$\Rightarrow$ Interpretation, die jeden Σ-Term t auf ein Element $t^{\mathcal{A}} \in A$ und jede Σ-Formel φ auf ein Element $\varphi^{\mathcal{A}} \in \{\textbf{true}, \textbf{false}\}$ abbildet

- $f(t_1, \ldots, t_n)^{\mathcal{A}} = f^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}})$
- $ite(\varphi, t_1, t_2)^{\mathcal{A}} = \texttt{if } \varphi^{\mathcal{A}} \texttt{ then } t_1^{\mathcal{A}} \texttt{ else } t_2^{\mathcal{A}}$
- $p(t_1, \ldots, t_n)^{\mathcal{A}} = p^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}})$
- $\bot^{\mathcal{A}} = \textbf{false}$
- $\top^{\mathcal{A}} = \textbf{true}$
- $(t_1 = t_2)^{\mathcal{A}} = \textbf{true}$ gdw. $t_1^{\mathcal{A}} = t_2^{\mathcal{A}}$

Semantik — 2

- Ein Σ-Modell $\mathcal{A}$ *erfüllt* (bzw. *falsifiziert*) eine Σ-Formel φ gdw. $\varphi^{\mathcal{A}} = \textbf{true}$ (bzw. $= \textbf{false}$)
- In SMT: Kein beliebiges Modell, sondern Modell, das zu einer bestimmten Theorie $\mathcal{T}$ gehört

Σ-Theorie

Ein oder mehrere (möglicherweise unendlich viele) Σ-Modelle

$\Rightarrow$ Variablenfreie Σ-Formel φ ist in einer Σ-Theorie $\mathcal{T}$ erfüllbar (*$\mathcal{T}$-erfüllbar*), gdw. es ein Modell T in $\mathcal{T}$ gibt, sodass $\varphi^{T} = \textbf{true}$

$\Rightarrow$ Eine Menge Γ von variablenfreien Σ-Formeln *$\mathcal{T}$-folgert* eine Formel φ, $\Gamma \models_{\mathcal{T}} \varphi$, gdw. jedes Modell von $\mathcal{T}$, das alle Formeln in Γ erfüllt auch φ erfüllt.

- Γ ist *$\mathcal{T}$-konsistent* gdw. $\Gamma \not\models_{\mathcal{T}} \bot$
- φ ist *$\mathcal{T}$-valide* gdw. $\emptyset \models_{\mathcal{T}} \varphi$
- Eine Klausel c ist ein *Theorie Lemma*, wenn c $\mathcal{T}$-valide ist, d.h. $\emptyset \models_{\mathcal{T}} c$

Uninterpretierte Symbole

- Üblicherweise will man in den Formeln auch uninterpretierte Symbole
  - uninterpretierte Konstanten: Ersetzen Variablen
  - uninterpretierte Aussagenlogische Konstanten: Ersetzen Teilformeln

Formal:

- Wir betrachten nicht $\mathcal{T}$, sondern eine Erweiterung $\mathcal{T}'$:
  - Sei Σ' eine beliebige Signatur, die Σ enthält
  - Eine *Erweiterung* $\mathcal{A}'$ zu Σ' eines Σ-Modells $\mathcal{A}$ ist ein Σ'-Modell mit
    - dem selben Universum wie $\mathcal{A}$ und
    - alle Symbole aus Σ werden in Σ' gleich interpretiert
  - $\mathcal{T}'$ ist die Menge aller möglichen Erweiterungen der Modelle aus $\mathcal{T}$ zu Σ'
- Wir sprechen jedoch weiterhin von $\mathcal{T}$-erfüllbar, $\mathcal{T}$-folgern, usw., haben jedoch im Kopf, dass wir eigentlich von der Erweiterung $\mathcal{T}'$ sprechen

Problemstellung – ground $\mathcal{T}$-satisfiability problem

Gegeben eine Σ-Theorie $\mathcal{T}$. Ist eine variablenfreie Formel über einer beliebigen Erweiterung von Σ mit uninterpretierten Konstanten $\mathcal{T}$-erfüllbar?

Bemerkung: φ ist $\mathcal{T}$-unerfüllbar, gdw. $\neg\varphi$ $\mathcal{T}$-valide ist.

Kombinierte Theorien

Wenn zwei (oder mehr) Theorien $\mathcal{T}_1$ und $\mathcal{T}_2$ in einer Formel kombiniert werden:

- Theorien über Axiome definiert? $\Rightarrow$ neue Theorie Vereinigung der beiden Axiommengen
- Besitzen die beiden Signaturen von $\mathcal{T}_1$ und $\mathcal{T}_2$ gemeinsame Symbole?
  - Haben diese Symbole nicht die selbe Bedeutung (selbe Relation bzgl. des Universums), muss eines der beiden umbenannt werden

Bei uns jedoch: Theorie ist eine Menge von Modellen

- Ein Σ-Modell $\mathcal{A}$ ist das Σ-Reduktum eines Σ'-Modells $\mathcal{B}$ mit $\Sigma' \supseteq \Sigma$, wenn
  - $\mathcal{A}$ das selbe Universum hat wie $\mathcal{B}$ und
  - $\mathcal{A}$ die Symbole exakt wie $\mathcal{B}$ interpretiert
- Die *Kombination* $\mathcal{T}_1 \oplus \mathcal{T}_2$ von $\mathcal{T}_1$ und $\mathcal{T}_2$ ist die Menge aller $(\Sigma_1 \cup \Sigma_2)$-Modelle $\mathcal{B}$, deren Σ_1-Reduktum isomorph ist zu einem Modell von $\mathcal{T}_1$ und deren Σ_2-Reduktum isomorph ist zu einem Modell von $\mathcal{T}_2$.
- Zusammenhang zu axiomatischer Sichtweise: Wenn jedes $\mathcal{T}_i$ die Menge aller Σ_i-Modelle ist, die eine Menge Γ_i an Axiomen erfüllen, dann ist $\mathcal{T}_1 \oplus \mathcal{T}_2$ genau die Menge aller $(\Sigma_1 \cup \Sigma_2)$-Modelle, die $\Gamma_1 \cup \Gamma_2$ erfüllen.

Abstraktion

- assoziiere mit jeder Signatur Σ eine Signatur Ω, die enthält:
  - aussagenlogische Konstanten von Σ
  - Menge an neuen aussagenlogischen Symbolen mit der selben Kardinalität wie die Menge der variablenfreien Σ-Atome
- Bijektion $\mathcal{T}2\mathcal{B}$ (*aussagenlogische Abstraktion 'Theory-to-Bool'*) zwischen den variablenfreien Σ-Formeln ohne *ite* Ausdrücke und den aussagenlogischen Formeln über Ω
  - Jede aussagenlogische Konstante aus Σ wird auf sich selbst abgebildet
  - Alle nicht-propositionalen Σ-Atome (atomare Formeln) werden auf neue aussagenlogische Symbole in Ω abgebildet (also: atomare Formeln werden durch Symbole repräsentiert).
  - *ite* Ausdrücke werden durch rein aussagenlogische Ausdrücke ersetzt
- Inverses von $\mathcal{T}2\mathcal{B}$ ist $\mathcal{B}2\mathcal{T}$ (*Refinement*)

Notation:

- φ^p anstelle von $\mathcal{T}2\mathcal{B}(\varphi)$
- Für Menge Γ an Σ-Formeln: $\Gamma^p = \{\varphi^p \mid \varphi \in \Gamma\}$
- Eine Σ-Formel φ ist aussagenlogisch unerfüllbar, wenn $\varphi^p \models \bot$
- $\Gamma \models_p \varphi$ bedeutet $\Gamma^p \models \varphi^p$
- $\Gamma \not\models_p \varphi$ impliziert $\Gamma \not\models_{\mathcal{T}} \varphi$, jedoch nicht umgekehrt

Wo sind wir?

1. **Formales ✓**
2. **Interessante Theorien**
   - Gleichheitslogik und uninterpretierte Funktionen
   - Arithmetik
   - Arrays
   - Bit Vektoren
3. **Ansätze zum Entscheiden von SMT Problemen**
   - Eager Approach
   - Lazy Approach
   - Das $\mathcal{T}$-DPLL Framework
4. **Kombination von Theorien**
   - Nelson-Oppen Methode für konvexe Theorien
   - Nelson-Oppen Methode für nicht-konvexe Theorien

Gleichheitslogik und uninterpretierte Funktionen (EUF)

- **Üblicherweise:** Aus einer Theorie ergeben sich Restriktionen, wie Symbole der Theorie interpretiert werden (Arithmetik, Pointerlogik,...)
- **Spezialfall:** Keinerlei Restriktionen (außer Bedeutung von $=$), d.h. die Theorie $\mathcal{T}_\Sigma$ besteht aus allen möglichen Modellen für eine Signatur Σ
- Man bezeichnet diese Theorie mit $\mathcal{T}_\mathcal{E}$ und spricht von der

  Theorie mit Gleichheit und uninterpretierten Funktionen (EUF)
- Entscheidbar in Polynomialzeit (*congruence closure*, siehe nächste Folie)

Verwendung: Abstraktion

- Abstrahieren von Theorien, die „schwerer" zu entscheiden sind
- Abstraktion UNSAT $\Rightarrow$ Originalformel UNSAT (Gilt nicht für den Fall SAT, da in der Originalformel die constraints der Theorie hinzukommen)

Beispiel

- Originalformelmenge: $\{a \cdot (f(b) + f(c)) = d, b \cdot (f(a) + f(c)) \neq d, a = b\}$
- Abstraktion: $\{h(a, g(f(b), f(c))) = d, h(b, g(f(a), f(c))) \neq d, a = b\}$
- Bereits Abstraktion ist UNSAT, daher auch die Originalformelmenge UNSAT

Entscheidungsverfahren für EUF: Kongruenzhülle

Kongruenzhüllen Algorithmus (Shostak, 1978)

- **Eingabe:** Eine Konjunktion φ von Gleichungen und Ungleichungen über Variablen und uninterpretierten Funktionen
- **Ausgabe:** SAT, wenn φ erfüllbar ist, ansonsten UNSAT

1. Generiere unter Kongruenz abgeschlossene Äquivalenzklassen
   1. *Initialisierung:* Zwei Terme t_1 und t_2 (Variablen oder UFs) sind in der selben Äquivalenzklasse, wenn es eine Gleichung $t_1 = t_2$ in φ gibt. Alle anderen Terme sind in einer eigenen Äquivalenzklasse.
   2. Teilen zwei Klassen einen Term, werden sie vereinigt. (Bis zum Fixpunkt)
   3. Sind t_1 und t_2 in der selben Klasse und sind $f(t_1)$ und $f(t_2)$ zwei Terme in φ, dann vereinige die Klassen von $f(t_1)$ und $f(t_2)$ (Bis zum Fixpunkt)
2. Ausgabe:
   - UNSAT, wenn es eine Ungleichung $t_1 \neq t_2$ in φ gibt, aber t_1 und t_2 in der selben Äquivalenzklasse sind
   - SAT sonst

Bemerkung: Algorithmus leicht erweiterbar auf UFs mit Stelligkeit > 1

Kongruenzhülle — Beispiel

Beispiel

$$\varphi = x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge f(x_1) \neq f(x_3)$$

1. Kongruenzhülle:
   1. *Initialisierung:*
   $$\{x_1, x_2\}, \{x_2, x_3\}, \{x_4, x_5\}, \{f(x_1)\}, \{f(x_3)\}$$
   2. *Vereinigen:*
   $$\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{f(x_1)\}, \{f(x_3)\}$$
   3. *Kongruenzhülle:*
   $$\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{f(x_1), f(x_3)\}$$

- $f(x_1)$ und $f(x_3)$ sind in der selben Äquivalenzklasse
- $f(x_1) \neq f(x_3)$ ist Term in φ

$\Rightarrow$ φ ist UNSAT

Arithmetik

- Signatur: $\Sigma_{\mathcal{Z}} = (0, 1, +, -, \leq)$
- $\mathcal{T}_{\mathcal{Z}}$ besteht aus den Modellen, die $\Sigma_{\mathcal{Z}}$ über den Ganzen Zahlen interpretieren (auch bekannt als *Presburger Arithmetik*)
- $\mathcal{T}_{\mathcal{R}}$ besteht aus den Modellen, die $\Sigma_{\mathcal{Z}}$ über den Reellen Zahlen interpretieren

Entscheidbarkeit

Sowohl $\mathcal{T}_{\mathcal{R}}$ als auch $\mathcal{T}_{\mathcal{Z}}$ und deren Erweiterungen sind entscheidbar

- $\mathcal{T}_{\mathcal{R}}$ ist in Polynomialzeit entscheidbar (aber Simplex, der im worstcase exponentiell ist, ist in der Praxis oft der beste Algorithmus)
- $\mathcal{T}_{\mathcal{Z}}$ ist NP-vollständig (siehe Pseudo Boolesche Optimierung...)
- Sobald man zu $\Sigma_{\mathcal{Z}}$ die Multiplikation hinzunimmt, wird $\mathcal{T}_{\mathcal{Z}}$ sogar für Konjunktionen von variablenfreien Formeln unentscheidbar (Matiyasevich 1971)
- $\mathcal{T}_{\mathcal{R}}$ mit Multiplikation ist entscheidbar (Tarski), aber doppelt exponentiell (Davenport-Heinz 1980)

- Multiplikation mit Konstanten kann in $\mathcal{T}_{\mathcal{Z}}$ verwendet werden: $4 \cdot x$ ist äquivalent zu $x + x + x + x$

Restriktionen der Arithmetik

Differenzlogik

- Jedes Atom ist von der Form $a - b \bowtie t$ mit
  - a und b uninterpretierte Konstanten
  - $\bowtie \in \{=, \leq\}$
  - t Ganzzahl
- Effiziente Entscheidungsverfahren vorhanden

UTVPI

- *Unit Two Variables per Inquality*
- Erlauben neben obigen Formeln auch noch $a + b \bowtie t$

Anwendungen:

- Modellierung von endlichen Mengen
- Programm Arithmetik
- Zeigermanipulation
- Speichermodellierung
- Physikalische Eigenschaften
- ...

Arrays

- Signatur $\Sigma_{\mathcal{A}} = (\mathit{read}, \mathit{write})$
- Array a
  - $\mathit{read}(a, i)$: Wert von a am Index i
  - $\mathit{write}(a, i, v)$: Array, das identisch zu a ist, nur mit Wert v an Index i

Formale Axiome

1. $\forall a\ \forall i\ \forall v\ (\mathit{read}(\mathit{write}(a, i, v), i) = v)$
2. $\forall a\ \forall i\ \forall j\ \forall v\ (i \neq j \rightarrow \mathit{read}(\mathit{write}(a, i, v), j) = \mathit{read}(a, j))$

- Theorie $\mathcal{T}_{\mathcal{A}}$ umfasst alle Modelle dieser Axiome
- Häufig auch noch $\forall a\ \forall b\ (\forall i\ (\mathit{read}(a, i) = \mathit{read}(b, i))) \rightarrow a = b$ (Extensionalitätsaxiom); Theorie heißt dann $\mathcal{T}_{\mathcal{A}ex}$
- Sowohl $\mathcal{T}_{\mathcal{A}}$ als auch $\mathcal{T}_{\mathcal{A}ex}$ sind NP-vollständig, aber es gibt „gute“ Algorithmen für praktische Probleme
- Kann auf EUF reduziert werden
- **Anwendung:** Modellierung von Datenstrukturen und Speicher

Bit Vektoren

- Bereits bekannt aus Bounded Model Checking
- Konstanten: Bit Vektoren, jede Konstante hat eine feste Anzahl an Bits assoziiert
- Funktionen und Prädikate: Extraktion, Konkatenation, Bitweise Boolesche Operatoren, Arithmetische Operatoren
- Entscheidbarkeit der Theorien ist NP-vollständig (kann in Polynomialzeit auf SAT reduziert werden $\rightarrow$ Bit Blasting)
- Bit Vektor Probleme können oft effizienter gelöst werden als die entsprechende Modellierung auf Bit Ebene
- **Anwendungen:** Modellierung von Schaltkreisen, Maschinenoperationen

Wo sind wir?

1. **Formales ✓**
2. **Interessante Theorien ✓**
   - Gleichheitslogik und uninterpretierte Funktionen
   - Arithmetik
   - Arrays
   - Bit Vektoren
3. **Ansätze zum Entscheiden von SMT Problemen**
   - Eager Approach
   - Lazy Approach
   - Das $\mathcal{T}$-DPLL Framework
4. **Kombination von Theorien**
   - Nelson-Oppen Methode für konvexe Theorien
   - Nelson-Oppen Methode für nicht-konvexe Theorien

Eager vs. Lazy Encodings

Eager Approach

- **Methodologie:** Übersetze gesamtes Problem in eine erfüllbarkeitsäquivalente Formel in Aussagenlogik und benutze SAT Solver
- **Warum „eager“:** Sämtliche Theorieinformation wird von Anfang an verwendet (in ihrer Übersetzung)
- **Vor/Nachteile:**
  - \+ Fortschritte in SAT direkt nutzbar (verwende besten SAT Solver)
  - – Schwierige Kodierung einiger Theorien in AL

Lazy Approach

- **Methodologie:** Rufe speziellen Theory Solver auf, wenn er gebraucht wird
- **Warum „lazy“:** Theorieinformation wir erst benutzt, wenn der jeweilige Theory Solver aufgerufen wird
- **Vor/Nachteile:**
  - \+ Modular und flexibel (Theory Solver können gepluggt werden)
  - – Die (SAT-)Suche wird nicht durch die Theorieinformation geleitet (kein feed-back von der Theorie zum SAT-Solver)

Eager Approach — Beispiel

Beispiel

Formel: $f(a) = f(b) \wedge f(b) \neq f(c)$

1. Entferne Funktionen und Prädikate durch Konstanten (z.B. Ackermann Reduktion)
   - Ersetze $f(a)$ durch A, $f(b)$ durch B und $f(c)$ durch C
   - Füge Klauseln hinzu: $a = b \rightarrow A = B$, $a = c \rightarrow A = C$ und $b = c \rightarrow B = C$
   - Jetzt sind alle Atome Gleichungen zwischen Konstanten

$$A = B \wedge \neg(B = C) \wedge a = b \rightarrow A = B \wedge a = c \rightarrow A = C \wedge b = c \rightarrow B = C$$

2. Übersetze Formeln in Aussagenlogik
   - Small Domain Encoding
     - Wenn es n verschiedene Konstanten gibt, gibt es im Fall dieser einfachen Gleichungslogik ein Modell mit Größe $\leq n$, falls es überhaupt ein Modell gibt (andere Logiken wie z.B. Differenzlogik oder UTVPI haben höhere Schranken)
     - Benutze $\log n$ Bits um den Wert jeder Konstante zu kodieren
     - $a = b$ wird mithilfe der Bits für a und b (bitweise) übersetzt
   - Direct Encoding (Per-Constraint Encoding)
     - Jedes Atom $a = b$ wird mit Variable $P_{a,b}$ ersetzt
     - Relevantes Wissen über die ersetzte Relation wird als Constraints zugefügt
     - Hier z.B. Transitivitätsconstraints: $P_{a,b} \wedge P_{b,c} \rightarrow P_{a,c}$

Lazy Approach: Was sollte ein Theorie ($\mathcal{T}$-)Solver können?

- **Model Generation:** Wird der $\mathcal{T}$-Solver auf eine $\mathcal{T}$-konsistente Menge Γ angewandt, kann er ein $\mathcal{T}$-Modell $\mathcal{I}$ mit $\mathcal{I} \models_{\mathcal{T}} \Gamma$ zurückgeben
- **Conflict Set Generation:** Wird der $\mathcal{T}$-Solver auf eine $\mathcal{T}$-inkonsistente Menge Γ angewandt, kann er eine (möglichst minimale) Teilmenge η von Γ zurückgeben, die die Inkonsistenz verursacht hat
- **Incrementality:** Der $\mathcal{T}$-Solver ist inkrementell. Nach Lösung von Γ kann er in diesem Zustand weitermachen, um $\Gamma \cup \Delta$ zu lösen
- **Backtrackability:** Der $\mathcal{T}$-Solver kann Berechnungsschritte effizient rückgängig machen und zu einem früheren Zustand zurückkehren
- **Deduction of Unassigned Literals:** Auf eine $\mathcal{T}$-konsistente Menge Γ angewandt, kann ein $\mathcal{T}$-Solver Deduktionen der Form $\Gamma' \models_{\mathcal{T}} \ell^p$ ausführen mit $\Gamma' \subseteq \Gamma$ und ℓ^p ein bisher unbelegtes Literal
- **Deduction of Interface Equalities:** Wenn der $\mathcal{T}$-Solver SAT zurückgibt, kann er Deduktionen der Form $\Gamma \models_{\mathcal{T}} e$ vollziehen
    - e ist eine Gleichung zwischen Variablen oder Termen, die in Atomen von Γ vorkommen

Offline Integration

- Einfachste Integrationsform
- Eingabeformel φ mit aussagenlogischer Abstraktion φ^p
- Entscheide φ^p (wird in CNF übersetzt) mit DPLL Solver
- Ergebnis UNSAT: Auch φ ist UNSAT
- Ergebnis SAT mit erfüllender Belegung (Modell) Γ^p
  - Menge an entsprechenden $\mathcal{T}$-Literalen Γ wird separat mit $\mathcal{T}$-Solver entschieden
  - Γ ist $\mathcal{T}$-konsistent: Auch φ ist $\mathcal{T}$-konsistent
  - Γ ist $\mathcal{T}$-inkonsistent: $\neg\Gamma^p$ als Klausel zu φ^p hinzugefügt und SAT Solver wird komplett neu gestartet um weiteres Modell von φ^p zu finden
- DPLL Solver wird als Black Box verwendet
  - keine Veränderungen nötig
  - jeder Solver kann benutzt werden

Nachteile

- SAT Solver wird jedesmal komplett neu gestartet
- $\mathcal{T}$-Solver bekommt immer nur vollständige Modelle von φ^p zur Entscheidung

Online Integration

- Intelligentere Integrationsform
- SAT Solver wird dahingehend modifiziert, dass er die erfüllenden Belegungen Γ^p von φ^p enumeriert und jeweils die $\mathcal{T}$-Literale Γ mit dem $\mathcal{T}$-Solver auf Konsistenz prüft, bis ein $\mathcal{T}$-konsistentes Γ gefunden ist (oder die Modelle von φ^p erschöpft sind).
- Die folgende Folie zeigt eine online $\mathcal{T}$-DPLL Prozedur
  - Eingabe φ ist eine $\mathcal{T}$-Formel
  - Eingabe Γ ist eine (Referenz auf eine) anfangs leere Menge von $\mathcal{T}$-Literalen.
  - Der eingebettete DPLL-Solver arbeitet auf φ^p und erneuert Γ^p
  - $\mathcal{T}$-DPLL kennt die Literale von φ und die bijektive Abbildung $\mathcal{B}2\mathcal{T}/\mathcal{T}2\mathcal{B}$.
- DPLL Solver wird verändert und mit $\mathcal{T}$-Solver integriert
- Backtracking und Lernen des DPLL-Solvers kann vom $\mathcal{T}$-Solver profitieren

Nachteile

- $\mathcal{T}$-DPLL profitiert nicht automatisch von Verbesserungen im SAT Solving
- Es können nicht automatisch verschiedene SAT Solver ausgetauscht werden

Online Integration: $\mathcal{T}$-DPLL

Algorithm 1: Online Schema für $\mathcal{T}$-DPLL

Input: $\mathcal{T}$-Formel φ, $\mathcal{T}$-Belegung Γ
Output: SAT oder UNSAT

if $\mathcal{T}-\texttt{preprocess}(\varphi, \Gamma) ==$ *Conflict* **then**
  return *UNSAT*
while *true* **do**
  $\mathcal{T}-\texttt{decide\_next\_branch}(\varphi^p, \Gamma^p)$
  while *true* **do**
    status $= \mathcal{T}-\texttt{deduce}(\varphi^p, \Gamma^p)$
    if *status* $==$ $\mathcal{T}$-*SAT* **then**
      $\Gamma = \mathcal{B}2\mathcal{T}(\Gamma^p)$
      return *SAT*
    else if *status* $==$ $\mathcal{T}$-*Conflict* **then**
      blevel $= \mathcal{T}-\texttt{analyze\_conflict}(\varphi^p, \Gamma^p)$
      if *blevel* $==$ *-1* **then**
        return *UNSAT*
      $\mathcal{T}-\texttt{backtrack}(blevel, \varphi^p, \Gamma^p)$
    else
      break

$\mathcal{T}$-DPLL Algorithmus — Erklärungen

- $\mathcal{T}$−`preprocess`: Simplifiziert φ und updated Γ, so dass $\mathcal{T}$-Erfüllbarkeit von $\varphi \wedge \Gamma$ erhalten bleibt (AL Simplifikation + $\mathcal{T}$-Rewriting)
- $\mathcal{T}$−`decide_next_branch`: Wählt nächste Variable aus
- $\mathcal{T}$−`deduce`: *Siehe nächste Folie*
- $\mathcal{T}$−`analyze_conflict`: Erweiterung der klassischen DPLL Konfliktanalyse
  - Boolescher Konflikt: Boolesche Konfliktmenge η^p (entspricht einer gelernten Klausel $\neg\eta^p$) und entsprechendes blevel
  - $\mathcal{T}$-Konflikt: Benutze die AL-Abstraktion η^p der Konfliktmenge η des $\mathcal{T}$-Solvers
- $\mathcal{T}$−`backtrack`: Wie Backtracking im DPLL
  - $\neg\eta^p$ wird zu φ^p hinzugefügt
  - Backtracking zu Level blevel

Erweiterung von DPLL

1. **Deduktion** nicht nur Boole´sch ($\Gamma^p \wedge \varphi^p \models_p \ell^p$) sondern auch in der Theorie ($\Gamma \models_{\mathcal{T}} \ell$)
2. Nicht nur Boole´sche **Konflikte** ($\varphi^p \wedge \Gamma^p \models_p \perp$) sondern auch Theorie Konflikte ($\Gamma \models_{\mathcal{T}} \perp$)

Deduktion im $\mathcal{T}$-DPLL Framework

$\mathcal{T}-\mathtt{deduce}(\varphi^p, \Gamma^p)$

Folgert iterativ Boole´sche Literale ℓ^p, die durch die aktuelle Belegung impliziert werden (d.h. $\varphi^p \wedge \Gamma^p \models_p \ell^p$), bis eine der folgenden Bedingungen wahr wird:

1. Γ^p verletzt φ^p aussagenlogisch, d.h. $\Gamma^p \wedge \varphi^p \models_p \bot$
   - Verhalten wie DPLL
   - Rückgabe: CONFLICT
2. Γ^p erfüllt φ^p aussagenlogisch, d.h. $\Gamma^p \models_p \varphi^p$
   - $\mathcal{T}$-Solver wird auf Γ angewandt
   - Wenn $\mathcal{T}$-konsistent, Rückgabe: SAT
   - Andernfalls Rückgabe: CONFLICT
3. Keine weiteren Literale können mehr gefolgert werden
   - Rückgabewert: UNKNOWN
   - Oder: $\mathcal{T}$-Solver wird auf (noch unvollständigem) Γ aufgerufen, wenn Γ schon jetzt $\mathcal{T}$-inkonsistent, dann Rückgabe CONFLICT (Early Pruning)

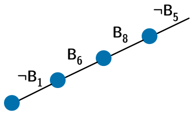
Bemerkung: Große Verbesserung kann erzielt werden, wenn der $\mathcal{T}$-Solver „Deduction of Unassigned Literals“ beherrscht

$\mathcal{T}$-DPLL — Beispiel

$\varphi =$		$\varphi^p =$
c_1:	$\{\neg(2x_2 - x_3 > 2) \lor A_1\}$	$\{\neg B_1 \lor A_1\}$
c_2:	$\{\neg A_2 \lor (x_1 - x_5 \leq 1)\}$	$\{\neg A_2 \lor B_2\}$
c_3:	$\{(3x_1 - 2x_2 \leq 3) \lor A_2\}$	$\{B_3 \lor A_2\}$
c_4:	$\{\neg(2x_3 + x_4 \geq 5) \lor \neg(3x_1 - x_3 \leq 6) \lor \neg A_1\}$	$\{\neg B_4 \lor \neg B_5 \lor \neg A_1\}$
c_5:	$\{A_1 \lor (3x_1 - 2x_2 \leq 3)\}$	$\{A_1 \lor B_3\}$
c_6:	$\{(x_2 - x_4 \leq 6) \lor (x_5 = 5 - 3x_4) \lor \neg A_1\}$	$\{B_6 \lor B_7 \lor \neg A_1\}$
c_7:	$\{A_1 \lor (x_3 = 3x_5 + 4) \lor A_2\}$	$\{A_1 \lor B_8 \lor A_2\}$

$\mathcal{T}$-DPLL — Beispiel

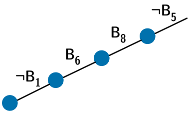
$\varphi =$		$\varphi^p =$
c_1:	$\{\neg(2x_2 - x_3 > 2) \vee A_1\}$	$\{\neg B_1 \vee A_1\}$
c_2:	$\{\neg A_2 \vee (x_1 - x_5 \leq 1)\}$	$\{\neg A_2 \vee B_2\}$
c_3:	$\{(3x_1 - 2x_2 \leq 3) \vee A_2\}$	$\{B_3 \vee A_2\}$
c_4:	$\{\neg(2x_3 + x_4 \geq 5) \vee \neg(3x_1 - x_3 \leq 6) \vee \neg A_1\}$	$\{\neg B_4 \vee \neg B_5 \vee \neg A_1\}$
c_5:	$\{A_1 \vee (3x_1 - 2x_2 \leq 3)\}$	$\{A_1 \vee B_3\}$
c_6:	$\{(x_2 - x_4 \leq 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\}$	$\{B_6 \vee B_7 \vee \neg A_1\}$
c_7:	$\{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\}$	$\{A_1 \vee B_8 \vee A_2\}$



- Initiale Belegung: $\Gamma^p = \{\neg B_5, B_8, B_6, \neg B_1\}$
- Damit erfüllt: c_1, c_4, c_6, c_7
- Keine Propagation möglich
- Erweiterter Fall 3) von $\mathcal{T}-\texttt{deduce}$

$\mathcal{T}$-DPLL — Beispiel

	$\varphi =$	$\varphi^p =$
c_1:	$\{\neg(2x_2 - x_3 > 2) \vee A_1\}$	$\{\neg B_1 \vee A_1\}$
c_2:	$\{\neg A_2 \vee (x_1 - x_5 \leq 1)\}$	$\{\neg A_2 \vee B_2\}$
c_3:	$\{(3x_1 - 2x_2 \leq 3) \vee A_2\}$	$\{B_3 \vee A_2\}$
c_4:	$\{\neg(2x_3 + x_4 \geq 5) \vee \neg(3x_1 - x_3 \leq 6) \vee \neg A_1\}$	$\{\neg B_4 \vee \neg B_5 \vee \neg A_1\}$
c_5:	$\{A_1 \vee (3x_1 - 2x_2 \leq 3)\}$	$\{A_1 \vee B_3\}$
c_6:	$\{(x_2 - x_4 \leq 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\}$	$\{B_6 \vee B_7 \vee \neg A_1\}$
c_7:	$\{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\}$	$\{A_1 \vee B_8 \vee A_2\}$



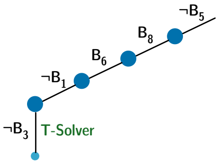
- $\Gamma^p = \{\neg B_5, B_8, B_6, \neg B_1\}$
- $\mathcal{T}$-Solver wird auf

$$\Gamma = \{\neg(3x_1 - x_3 \leq 6), (x_3 = 3x_5 + 4), (x_2 - x_4 \leq 6), \neg(2x_2 - x_3 > 2)\}$$

angewendet

$\mathcal{T}$-DPLL — Beispiel

	$\varphi =$	$\varphi^p =$
c_1:	$\{\neg(2x_2 - x_3 > 2) \vee A_1\}$	$\{\neg B_1 \vee A_1\}$
c_2:	$\{\neg A_2 \vee (x_1 - x_5 \leq 1)\}$	$\{\neg A_2 \vee B_2\}$
c_3:	$\{(3x_1 - 2x_2 \leq 3) \vee A_2\}$	$\{B_3 \vee A_2\}$
c_4:	$\{\neg(2x_3 + x_4 \geq 5) \vee \neg(3x_1 - x_3 \leq 6) \vee \neg A_1\}$	$\{\neg B_4 \vee \neg B_5 \vee \neg A_1\}$
c_5:	$\{A_1 \vee (3x_1 - 2x_2 \leq 3)\}$	$\{A_1 \vee B_3\}$
c_6:	$\{(x_2 - x_4 \leq 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\}$	$\{B_6 \vee B_7 \vee \neg A_1\}$
c_7:	$\{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\}$	$\{A_1 \vee B_8 \vee A_2\}$

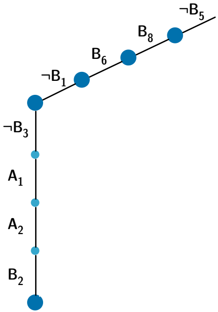


- $\mathcal{T}$-Solver folgert (z.B.) $\neg(3x_1 - 2x_2 \leq 3)^p = \neg B_3$ als Konsequenz von $\neg B_5$ und $\neg B_1$ (B_3 ist in c_3 und c_5 enthalten)
- Also $\neg B_5 \wedge \neg B_1 \models \neg B_3$ (Deduction of unassigned literal $\neg B_3$)
- $\Gamma^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3\}$

$\mathcal{T}$-DPLL — Beispiel

	$\varphi =$	$\varphi^p =$
c_1:	$\{\neg(2x_2 - x_3 > 2) \vee A_1\}$	$\{\neg B_1 \vee A_1\}$
c_2:	$\{\neg A_2 \vee (x_1 - x_5 \leq 1)\}$	$\{\neg A_2 \vee B_2\}$
c_3:	$\{(3x_1 - 2x_2 \leq 3) \vee A_2\}$	$\{B_3 \vee A_2\}$
c_4:	$\{\neg(2x_3 + x_4 \geq 5) \vee \neg(3x_1 - x_3 \leq 6) \vee \neg A_1\}$	$\{\neg B_4 \vee \neg B_5 \vee \neg A_1\}$
c_5:	$\{A_1 \vee (3x_1 - 2x_2 \leq 3)\}$	$\{A_1 \vee B_3\}$
c_6:	$\{(x_2 - x_4 \leq 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\}$	$\{B_6 \vee B_7 \vee \neg A_1\}$
c_7:	$\{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\}$	$\{A_1 \vee B_8 \vee A_2\}$

- Unit Propagations:
  - A_1 wegen c_5
  - A_2 wegen c_3
  - B_2 wegen c_2
- Dadurch $\Gamma'^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$
- Schicke entsprechende Teilmenge γ' von Γ' an $\mathcal{T}$-Solver: $\gamma'^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, B_2\}$
- Rückgabe: UNSAT (wegen $\mathcal{T}$-Literalen 1, 2 und 6)
- Rückgabe von $\mathcal{T}$−deduce: CONFLICT



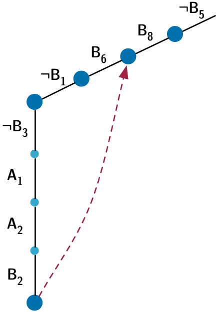
$\mathcal{T}$-DPLL — Beispiel

$\varphi =$		$\varphi^p =$
c_1:	$\{\neg(2x_2 - x_3 > 2) \vee A_1\}$	$\{\neg B_1 \vee A_1\}$
c_2:	$\{\neg A_2 \vee (x_1 - x_5 \leq 1)\}$	$\{\neg A_2 \vee B_2\}$
c_3:	$\{(3x_1 - 2x_2 \leq 3) \vee A_2\}$	$\{B_3 \vee A_2\}$
c_4:	$\{\neg(2x_3 + x_4 \geq 5) \vee \neg(3x_1 - x_3 \leq 6) \vee \neg A_1\}$	$\{\neg B_4 \vee \neg B_5 \vee \neg A_1\}$
c_5:	$\{A_1 \vee (3x_1 - 2x_2 \leq 3)\}$	$\{A_1 \vee B_3\}$
c_6:	$\{(x_2 - x_4 \leq 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\}$	$\{B_6 \vee B_7 \vee \neg A_1\}$
c_7:	$\{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\}$	$\{A_1 \vee B_8 \vee A_2\}$
c_8:	...	$\{B_5 \vee \neg B_8 \vee \neg B_2\}$

- $\mathcal{T}$−`analyze_conflict` und $\mathcal{T}$−`backtrack` folgern und lernen die Klausel

$$c_8 = B_5 \vee \neg B_8 \vee \neg B_2$$

- Rücksprung zu entsprechendem blevel
- c_8 ist danach unit



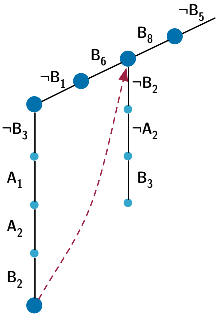
$\mathcal{T}$-DPLL — Beispiel

$\varphi =$		$\varphi^p =$
c_1:	$\{\neg(2x_2 - x_3 > 2) \vee A_1\}$	$\{\neg B_1 \vee A_1\}$
c_2:	$\{\neg A_2 \vee (x_1 - x_5 \leq 1)\}$	$\{\neg A_2 \vee B_2\}$
c_3:	$\{(3x_1 - 2x_2 \leq 3) \vee A_2\}$	$\{B_3 \vee A_2\}$
c_4:	$\{\neg(2x_3 + x_4 \geq 5) \vee \neg(3x_1 - x_3 \leq 6) \vee \neg A_1\}$	$\{\neg B_4 \vee \neg B_5 \vee \neg A_1\}$
c_5:	$\{A_1 \vee (3x_1 - 2x_2 \leq 3)\}$	$\{A_1 \vee B_3\}$
c_6:	$\{(x_2 - x_4 \leq 6) \vee (x_5 = 5 - 3x_4) \vee \neg A_1\}$	$\{B_6 \vee B_7 \vee \neg A_1\}$
c_7:	$\{A_1 \vee (x_3 = 3x_5 + 4) \vee A_2\}$	$\{A_1 \vee B_8 \vee A_2\}$
c_8:	...	$\{B_5 \vee \neg B_8 \vee \neg B_2\}$

- Unit Propagations
  - $\neg B_2$ wegen c_8
  - $\neg A_2$ wegen c_2
  - B_3 wegen c_3
- Alle Klauseln sind erfüllt

$\Rightarrow$ $\mathcal{T}$−deduce gibt SAT zurück

$\Rightarrow$ $\mathcal{T}$-DPLL gibt SAT zurück



Wo sind wir?

1. **Formales ✓**
2. **Interessante Theorien ✓**
   - Gleichheitslogik und uninterpretierte Funktionen
   - Arithmetik
   - Arrays
   - Bit Vektoren
3. **Ansätze zum Entscheiden von SMT Problemen ✓**
   - Eager Approach
   - Lazy Approach
   - Das $\mathcal{T}$-DPLL Framework
4. **Kombination von Theorien**
   - Nelson-Oppen Methode für konvexe Theorien
   - Nelson-Oppen Methode für nicht-konvexe Theorien

Kombination von Theorien

Beispiel

Häufig müssen verschiedene Theorien kombiniert werden:

- Lineare Arithmetik (LA) und Uninterpretierte Funktionen (EUF):

$$(x_2 \geq x_1) \wedge (x_1 - x_3 \leq x_2) \wedge (x_3 \geq 0) \wedge f(f(x_1) - f(x_2)) \neq f(x_3)$$

- Bit Vektoren (BV) und Uninterpretierte Funktionen (EUF):

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

- Arrays (AR) und lineare Arithmetik (LA):

$$x = a\{i \leftarrow e\}[j] \wedge y = a[j] \wedge x > e \wedge x > y$$

Idee von Nelson-Oppen Methode

- Eigener Solver für jede Theorie
- Solver können „Interface"-Informationen untereinander austauschen

Nelson-Oppen Methode — Voraussetzungen

Um die Nelson-Oppen Methode anwenden zu können, müssen die Theorien $T_1, \ldots, T_n$ im einfachsten Fall folgende Eigenschaften erfüllen:

1. $T_1, \ldots, T_n$ sind quantorenfreie first-order Theorien mit Gleichheit
2. Es gibt jeweils eine Entscheidungsprozedur für $T_1, \ldots, T_n$
3. Die Signaturen sind disjunkt, d.h. für alle $1 \leq i < j \leq n, \Sigma_i \cap \Sigma_j = \emptyset$
4. $T_1, \ldots, T_n$ werden über unendlichen Domänen interpretiert (z.B. lineare Arithmetik über $\mathcal{R}$, aber nicht Theorie der endlich breiten Bit Vektoren)

Es gibt Erweiterungen von Nelson-Oppen für jede dieser Restriktionen

Im Allgemeinen wird die Methode wesentlich effizienter, wenn zusätzlich gilt:

5. $T_1, \ldots, T_n$ sind konvexe Theorien

Die Methode prüft die Erfüllbarkeit einer Konjunktion φ von atomaren Formeln (ggf. zuvor DNF herstellen). Die atomaren Formeln werden zunächst mit Hilfe von Hilfsvariablen in neue „reine“ (*pure*) Atome zerlegt, die jeweils zu genau einer Theorie gehören (*purification*).

Konvexe Theorien

Definition (Konvexe Theorie)

Eine Σ-Theorie T ist konvex, wenn für jede konjunktive Σ-Formel φ gilt:

$$(\varphi \Rightarrow \bigvee_{i=1}^{n} x_i = y_i) \text{ ist } T\text{-valide für ein endliches } n > 1 \implies$$
$$(\varphi \Rightarrow x_i = y_i) \text{ ist } T\text{-valide für ein } i \in \{1, \ldots, n\}$$

mit x_i, y_i Variablen.

D.h. Wenn eine Formel eine Disjunktion von Gleichungen impliziert, impliziert sie mindestens eine dieser Gleichungen separat.

Beispiel

- Lineare Arithmetik über $\mathbb{R}$ ist konvex
- Lineare Arithmetik über $\mathbb{Z}$ ist nicht konvex
  - $x_1 = 1 \wedge x_2 = 2 \wedge 1 \leq x_3 \wedge x_3 \leq 2 \Rightarrow (x_3 = x_1 \vee x_3 = x_2)$ gilt
  - $x_1 = 1 \wedge x_2 = 2 \wedge 1 \leq x_3 \wedge x_3 \leq 2 \Rightarrow x_3 = x_1$ gilt nicht
  - $x_1 = 1 \wedge x_2 = 2 \wedge 1 \leq x_3 \wedge x_3 \leq 2 \Rightarrow x_3 = x_2$ gilt nicht

Purification — 1

- Erfüllbarkeitsäquivalente Transformation einer Konjunktion φ zu φ'
- In Konjunktion φ' ist jede atomare Formel aus nur einer Theorie (*ist pur*)
- Es werden Hilfsvariablen $a_{ij}, b_{ij}, \ldots$ aus einer Menge C eingeführt, die jeweils 2 Theorien T_i und T_j verbinden indem sie beiden angehören.

Definitionen: Sei $\Sigma = \Sigma_1 \cup \Sigma_2 \cup C$. Ein Σ-Term t ist ein *i-Term* wenn sein oberstes Funktionssymbol in $\Sigma_i \cup C$ ist. Ein Σ-Literal α ist ein *i-Literal* wenn sein oberstes Prädikatssymbol in $\Sigma_i \cup C$ ist oder wenn es die Form $(\neg)(s = t)$ hat und s und t beides i-Terme sind. Falls s und t unterschiedlichen Theorien angehören, wird $s = t$ einer der beiden Theorien zugeschlagen. Ein Teilterm eines i-Atoms α ist ein fremder (*alien*) Teilterm, wenn das oberste Symbol nicht in $\Sigma_i \cup C$ ist und alle Superterme i-Terme sind. Ein i-Term oder i-Literal ist rein (*pure*) wenn nur Symbole aus $\Sigma_i \cup C$ enthalten sind.

Purification:

1. $\varphi' := \varphi$
2. Iteriere solange wie möglich: Für jeden fremden Teilterm t eines Literals in φ'
   - Ersetze t mit neuer Hilfsvariable (Konstante) a_t
   - Füge Constraint $a_t = t$ zu φ' hinzu

Purification — 2

Beispiel

Lineare Arithmetik + Uninterpretierte Funktionen:

$$\varphi = x_1 \leq f(g())$$

Nach Purifikation:

$$\varphi' = x_1 \leq a \wedge a = f(g())$$

Beispiel

Lineare Arithmetik + Uninterpretierte Funktionen:

$$\varphi = (f(x_1, 0) \geq x_3)$$

Purifikation in 2 Schritten mit $C = \{a, b, c\}$:

$$\varphi' = a \geq x_3 \wedge a = f(x_1, 0)$$

$$\varphi'' = a \geq x_3 \wedge a = f(b, c) \wedge b = x_1 \wedge c = 0$$

Die Nelson-Oppen Methode für konvexe Theorien

- **Eingabe:** Konjunktion φ über verschiedenen konvexen Theorien $T_1, \ldots, T_n$
- **Ausgabe:** SAT, wenn φ erfüllbar ist, UNSAT sonst

1. **Purification:** Purifizieren von φ zu $\varphi' = \{F_1, \ldots, F_n\}$ mit $F_i \in T_i$.
2. **T_i-Decision:** Wende Entscheidungsverfahren für T_i auf F_i an
   - Wenn ein i existiert, so dass F_i in T_i nicht erfüllbar ist, Rückgabe: UNSAT
3. **Equality Propagation:** Wenn i und j existieren, so dass
   - F_i in T_i eine „Interface"-Gleichung $a = b$ mit zwischen T_i und T_j geteilten Hilfsvariablen impliziert und
   - diese Gleichung aber noch nicht von F_j in T_j impliziert wird,

   dann füge diese Gleichung zu F_j hinzu und gehe wieder zu Schritt 2.
4. Rückgabe SAT

Eingabeformel muss eine Konjunktion sein. Im Allgemeinen macht die Hinzunahme von Disjunktionen eine Theorie nicht-konvex.

Equality Propagation — 1

Nach der Purifikation:

1. Für alle i: F_i gehört zu T_i und ist eine Konjunktion von T_i-Literalen
2. Geteilte (*shared*) Variablen sind erlaubt
3. φ ist in der kombinierten Theorie erfüllbar, gdw. $\bigwedge_{i=1}^{n} F_i$ in der kombinierten Theorie erfüllbar ist

Beispiel

$(f(x_1, 0) \geq x_3) \wedge (f(x_2, 0) \leq x_3) \wedge (x_1 \geq x_2) \wedge (x_2 \geq x_1) \wedge (x_3 - f(x_1, 0) \geq 1)$
mischt lineare Arithmetik und Uninterpretierte Formeln.

Purifikation: $(a_1 \geq x_3) \wedge (a_2 \leq x_3) \wedge (x_1 \geq x_2) \wedge (x_2 \geq x_1) \wedge (x_3 - a_1 \geq 1) \wedge$
$(a_1 = f(b_1, b_0)) \wedge (b_1 = x_1) \wedge (b_0 = 0) \wedge$
$(a_2 = f(b_2, b_0)) \wedge (b_2 = x_2)$

Vorgenommene Optimierungen:

- Beide Instanzen von $f(x_1, 0)$ werden auf die selbe Hilfsvariable a_1 abgebildet
- Beide Instanzen von 0 werden auf die selbe Hilfsvariable b_0 abgebildet

Equality Propagation — 2

Beispiel

$(a_1 \geq x_3) \wedge (a_2 \leq x_3) \wedge (x_1 \geq x_2) \wedge (x_2 \geq x_1) \wedge (x_3 - a_1 \geq 1) \wedge$
$(a_1 = f(b_1, b_0)) \wedge (a_2 = f(b_2, b_0)) \wedge (b_0 = 0) \wedge (b_1 = x_1) \wedge (b_2 = x_2)$

F_1 (Arithmetik über $\mathbb{R}$)	F_2 (EUF)
$a_1 \geq x_3$	$a_1 = f(b_1, b_0)$
$a_2 \leq x_3$	$a_2 = f(b_2, b_0)$
$x_1 \geq x_2$	
$x_2 \geq x_1$	
$x_3 - a_1 \geq 1$	
$b_0 = 0$	
$b_1 = x_1$	
$b_2 = x_2$	

Equality Propagation — 2

Beispiel

$(a_1 \geq x_3) \wedge (a_2 \leq x_3) \wedge (x_1 \geq x_2) \wedge (x_2 \geq x_1) \wedge (x_3 - a_1 \geq 1) \wedge$
$(a_1 = f(b_1, b_0)) \wedge (a_2 = f(b_2, b_0)) \wedge (b_0 = 0) \wedge (b_1 = x_1) \wedge (b_2 = x_2)$

F_1 (Arithmetik über $\mathbb{R}$)	F_2 (EUF)
$a_1 \geq x_3$	$a_1 = f(b_1, b_0)$
$a_2 \leq x_3$	$a_2 = f(b_2, b_0)$
$x_1 \geq x_2$	
$x_2 \geq x_1$	
$x_3 - a_1 \geq 1$	
$b_0 = 0$	
$b_1 = x_1$	
$b_2 = x_2$	
$x_1 = x_2$	
$b_1 = b_2$	$b_1 = b_2$

- Aus $(x_1 \geq x_2) \wedge (x_2 \geq x_1)$ folgere $(x_1 = x_2)$
- Aus $(x_1 = x_2)$ folgere $(b_1 = b_2)$ in F_1 und ebenfalls $(b_1 = b_2)$ in F_2

Equality Propagation — 2

Beispiel

$(a_1 \geq x_3) \wedge (a_2 \leq x_3) \wedge (x_1 \geq x_2) \wedge (x_2 \geq x_1) \wedge (x_3 - a_1 \geq 1) \wedge$
$(a_1 = f(b_1, b_0)) \wedge (a_2 = f(b_2, b_0)) \wedge (b_0 = 0) \wedge (b_1 = x_1) \wedge (b_2 = x_2)$

F_1 (Arithmetik über $\mathbb{R}$)	F_2 (EUF)
$a_1 \geq x_3$	$a_1 = f(b_1, b_0)$
$a_2 \leq x_3$	$a_2 = f(b_2, b_0)$
$x_1 \geq x_2$	
$x_2 \geq x_1$	
$x_3 - a_1 \geq 1$	
$b_0 = 0$	
$b_1 = x_1$	
$b_2 = x_2$	
$x_1 = x_2$	
$b_1 = b_2$	$b_1 = b_2$
$a_1 = a_2$	$a_1 = a_2$

- Wegen $(b_1 = b_2)$ folgere $(a_1 = a_2)$ in F_2 und ebenfalls $(a_1 = a_2)$ in F_1.

Equality Propagation — 2

Beispiel

$(a_1 \geq x_3) \wedge (a_2 \leq x_3) \wedge (x_1 \geq x_2) \wedge (x_2 \geq x_1) \wedge (x_3 - a_1 \geq 1) \wedge$
$(a_1 = f(b_1, b_0)) \wedge (a_2 = f(b_2, b_0)) \wedge (b_0 = 0) \wedge (b_1 = x_1) \wedge (b_2 = x_2)$

F_1 (Arithmetik über $\mathbb{R}$)	F_2 (EUF)
$a_1 \geq x_3$	$a_1 = f(b_1, b_0)$
$a_2 \leq x_3$	$a_2 = f(b_2, b_0)$
$x_1 \geq x_2$	
$x_2 \geq x_1$	
$x_3 - a_1 \geq 1$	
$b_0 = 0$	
$b_1 = x_1$	
$b_2 = x_2$	
$x_1 = x_2$	
$b_1 = b_2$	$b_1 = b_2$
$a_1 = a_2$	$a_1 = a_2$

Aus $(a_1 = a_2)$ folgere $(a_1 = x_3)$ und mit $(x_3 - a_1 \geq 1)$ einen Widerspruch in F_1, also Gesamtergebnis UNSAT.

Nicht-konvexe Theorien — 1

Beispiel (Problem bei nicht-konvexen Theorien)

- $\varphi = (1 \leq x) \wedge (x \leq 2) \wedge p(x) \wedge \neg p(1) \wedge \neg p(2)$ mit $x \in \mathbb{Z}$
  Theorien: LA über $\mathbb{Z}$ (nicht konvex) plus EUP (uninterpretierte Prädikate)
- $\varphi' = (1 \leq x) \wedge (x \leq 2) \wedge p(a_0) \wedge \neg p(a_1) \wedge \neg p(a_2) \wedge a_0 = x \wedge a_1 = 1 \wedge a_2 = 2$

F_1 (Arithmetik über $\mathbb{Z}$)	F_2 (EUP)
$1 \leq x$	$p(a_0)$
$x \leq 2$	$\neg p(a_1)$
$a_0 = x$	$\neg p(a_2)$
$a_1 = 1$	
$a_2 = 2$	

- Sowohl F_1 als auch F_2 unabhängig voneinander erfüllbar
- Keine neuen Gleichungen werden impliziert

$\Rightarrow$ Rückgabewert: SAT

- Originalformel ist jedoch UNSAT in der kombinierten Theorie
- Lösung: F_1 impliziert $x = 1 \vee x = 2$.

Nicht-konvexe Theorien — 2

Beispiel (Case Split bei nicht-konvexen Theorien)

- $\varphi' = (1 \leq x) \wedge (x \leq 2) \wedge p(a_0) \wedge \neg p(a_1) \wedge \neg p(a_2) \wedge a_0 = x \wedge a_1 = 1 \wedge a_2 = 2$
- F_1 impliziert $x = 1 \vee x = 2 \Rightarrow$ **Case Split**

F_1 (LA über $\mathbb{Z}$)	F_2 (EUP)
$1 \leq x$	$p(a_0)$
$x \leq 2$	$\neg p(a_1)$
$a_0 = x$	$\neg p(a_2)$
$a_1 = 1$	
$a_2 = 2$	
$x = 1$	
$x = a_1$	
$a_0 = a_1$	$a_0 = a_1$
	false

F_1 (LA über $\mathbb{Z}$)	F_2 (EUP)
$1 \leq x$	$p(a_0)$
$x \leq 2$	$\neg p(a_1)$
$a_0 = x$	$\neg p(a_2)$
$a_1 = 1$	
$a_2 = 2$	
$x = 2$	
$x = a_2$	
$a_0 = a_2$	$a_0 = a_2$
	false

- $a_0 = a_1$ bzw. $a_0 = a_2$ sind die einzigen abgeleiteten Gleichungen in der Theorie LA über $\mathbb{Z}$, die auch in EUP sind und deshalb propagiert werden.
- In beiden Fällen ist die Rückgabe false, also ist Gesamtergebnis UNSAT

Die Nelson-Oppen Methode für nicht-konvexe Theorien

- **Eingabe:** Konjunktion φ über verschiedenen Theorien $T_1, \dots, T_n$
- **Ausgabe:** SAT, wenn φ erfüllbar ist, UNSAT sonst

1. **Purification:** Purifizieren von φ zu einer Literalmenge $\varphi' = \{F_1, \dots, F_n\}$
2. T_i**-Decision:** Wende Entscheidungsverfahren für T_i auf F_i an
   - Wenn ein i existiert, so dass F_i in T_i nicht erfüllbar ist, Rückgabe: UNSAT
3. **Equality Propagation:** Wenn i und j existieren, so dass
   - F_i in T_i eine „Interface"-Gleichung $a = b$ mit zwischen T_i und T_j geteilten Hilfsvariablen impliziert und
   - diese Gleichung aber noch nicht von F_j in T_j impliziert wird,

   dann füge diese Gleichung zu F_j hinzu
4. **Splitting:** Wenn ein i existiert mit
   - $F_i \Rightarrow (x_1 = y_1 \vee \cdots \vee x_k = y_k)$ und
   - $\forall j \in \{1, \dots, k\} : F_i \not\Rightarrow x_j = y_j$

   Rufe Nelson-Oppen rekursiv auf den Teilproblemen

   $$\varphi' \wedge x_1 = y_1, \dots, \varphi' \wedge x_k = y_k$$

   auf. Ist eines der Subprobleme SAT, so gebe SAT zurück, ansonsten wenn alle UNSAT sind, dann UNSAT.
5. Rückgabe SAT

Wir sind fertig

1. **Formales ✓**
2. **Interessante Theorien ✓**
   - Gleichheitslogik und uninterpretierte Funktionen
   - Arithmetik
   - Arrays
   - Bit Vektoren
3. **Ansätze zum Entscheiden von SMT Problemen ✓**
   - Eager Approach
   - Lazy Approach
   - Das $\mathcal{T}$-DPLL Framework
4. **Kombination von Theorien ✓**
   - Nelson-Oppen Methode für konvexe Theorien
   - Nelson-Oppen Methode für nicht-konvexe Theorien