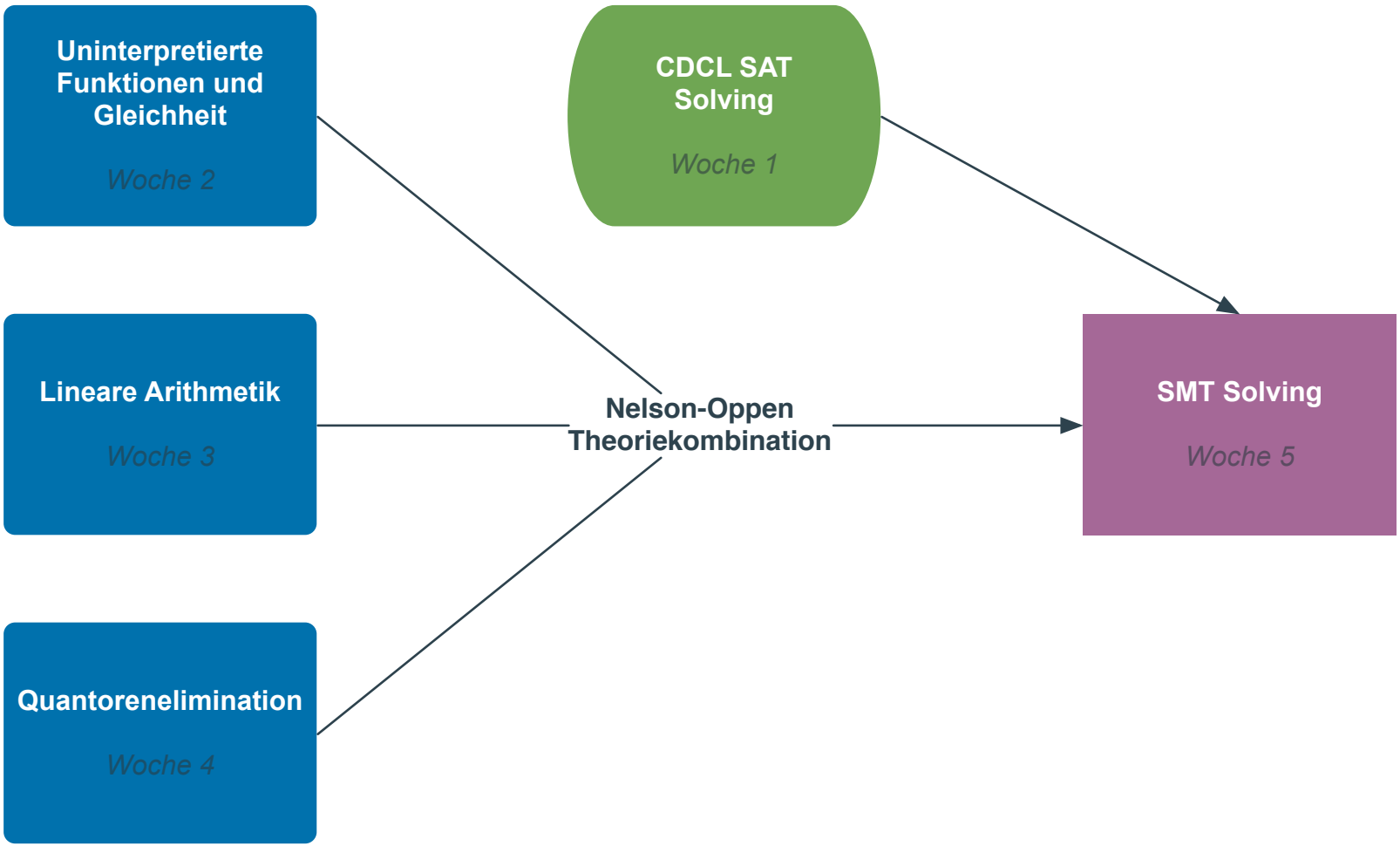
Automatisches Beweisen—Vertiefung
Entscheidbare FOL Fragmente
Uninterpretierte Funktionen und Gleichheit

Christoph Zengler

Arbeitsbereich Symbolisches Rechnen
Prof. Dr. Wolfgang Küchlin
Universität Tübingen

22. November 2011

Wir erinnern uns...



Gleichheitslogik

Syntax

- Aussagenlogik, bei der Atomare Formeln durch Gleichungen zwischen Variablen oder zwischen Variablen und Konstanten ersetzt werden
- Unendliche Domäne $\mathcal{D}$

EBNF-Grammatik für Gleichungslogik

Term	$=$	*Termvariable*	über einer Domäne $\mathcal{D}$
	$\mid$	*Konstante*	aus einer Domäne $\mathcal{D}$

Formel	$=$	$\top \mid \bot$	Konstanten
	$\mid$	Term $=$ Term	Gleichung
	$\mid$	$\neg$ Formel	Negation
	$\mid$	Formel $\wedge$ Formel	Konjunktion
	$\mid$	Formel $\vee$ Formel	Disjunktion
	$\mid$	Formel $\rightarrow$ Formel	Implikation
	$\mid$	Formel $\leftrightarrow$ Formel	Äquivalenz
	$\mid$	(Formel)	Klammerung

Gleichheitslogik

Semantik

- Universum ist $\mathcal{D}$
- Konstanten sind 0-stellige Funktionssymbole, die mit dem jeweiligen Wert aus $\mathcal{D}$ interpretiert werden, wobei die Werte für paarweise verschiedene Funktionssymbole ebenfalls paarweise verschieden sein müssen.
- Einziges Prädikat $=$ wird mit der üblichen Gleihheit über $\mathcal{D}$ interpretiert

Damit ist eine Interpretation $\mathcal{M}_{\mathrm{GL}}$ fixiert.

- Belegung β bildet Variablen nach $\mathcal{D}$ ab
- Im Gegensatz zur allgemeinen FOL reicht für die Erfüllbarkeit die **Existenz** einer Belegung β aus, so dass die Formel zu `true` evaluiert.

Definition (Erfüllbarkeit in der Gleichheitslogik)

Eine Formel φ in GL ist erfüllbar gdw. es eine Belegung β gibt, so dass $\texttt{holds}(\mathcal{M}_{\mathrm{GL}}, \beta, \varphi) = \texttt{true}$ gilt.

Gleichheitslogik

Beispiele & Komplexität

Beispiel (Gleichungslogik)

- $a = b \wedge \neg(b = c) \rightarrow \neg(a = c)$
- $a = 24 \vee a = 28 \rightarrow \neg(a = -2)$

Wir können statt $\neg(\cdots = \ldots)$ auch $(\cdots \neq \ldots)$ schreiben:

- $a = b \wedge b \neq c \rightarrow a \neq c$
- $a = 24 \vee a = 28 \rightarrow a \neq -2$

- Erfüllbarkeitsproblem für Gleichungslogik NP-vollständig
- Kann die selben Probleme modellieren wie SAT

? Warum dann überhaupt Gleichungslogik?

1. Einfachere Problemcodierung (kein Übersetzten nach AL)
2. Strukturinformation in Entscheidungsverfahren nutzbar

Entfernen von Konstanten

Algorithmus: `removeConstants`(φ)

Eingabe: Eine Formel φ in Gleichheitslogik
Ausgabe: Eine Formel φ' in Gleichheitslogik mit φ und φ' erfüllbarkeitsäquivalent und φ' konstantenfrei

1. $\varphi' = \varphi$
2. Ersetze jede Konstante c_i in φ' mit einer neuen Variable C_i
3. Füge für jedes Paar von Konstanten c_i, c_j neue Constraints $C_i \neq C_j$ zu φ'

Beispiel (Entfernen von Konstanten)

1. $\varphi = (a = -2 \wedge b \neq 28 \rightarrow a \neq 24) \leftrightarrow (a = 24 \vee a = 28 \rightarrow a \neq -2)$
2. Ersetzen der Konstanten in φ':
   $\varphi' = (a = C_1 \wedge b \neq C_2 \rightarrow a \neq C_3) \leftrightarrow (a = C_3 \vee a = C_2 \rightarrow a \neq C_1)$
3. Hinzufügen der Constraints:
   $\varphi' = \varphi' \wedge (C_1 \neq C_2) \wedge (C_1 \neq C_3) \wedge (C_2 \neq C_3)$

Uninterpretierte Funktionen

- Funktions- und Prädikatssymbole, die nicht interpretiert werden
- Können mit interpretierten Symbolen gemischt werden
- Einzige Eigenschaft: Funktionale Konsistenz

Definition (Funktionale Konsistenz)

Für jedes uninterpretierte Funktions- oder Prädikatssymbol S gilt:

$$\forall t_1, \ldots, t_n, t'_1, \ldots, t'_n \left[\left(\bigwedge_{i \in \{1,\ldots,n\}} t_i = t'_i \right) \rightarrow S(t_1, \ldots, t_n) = S(t'_1, \ldots, t'_n) \right]$$

EBNF-Grammatik für EUF

Term	$=$	*Termvariable*	über einer Domäne $\mathcal{D}$
	$\mid$	*Funktionssymbol*({Term})	Funktion
Formel	$=$	Term $=$ Term	Gleichung
	$\mid$	*Prädikatssymbol*({Term})	Prädikat
	$\mid$	$\ldots$	

Verwendung von uninterpretierten Funktionen

- φ^{UF} ist φ, wobei alle Vorkommen von Funktionen und Prädikaten durch uninterpretierte ersetzt werden
- Beliebte Technik um einfachere Entscheidungsverfahren einsetzen zu können
- Schwächt die Formel ab, d.h. $\models \varphi^{\mathrm{UF}} \rightarrow \models \varphi$

Beispiel (Schwächung durch UF)

- **Originalformel:** $\models_{\mathbb{Z}} (x \cdot y) - (y \cdot x) = 0$
- **UF Formel:** $\models_{\mathbb{Z}} G(F(x,y), F(y,x)) = 0$ gilt jedoch nicht

Ist die UF-Formel keine Tautologie, so kann man wieder graduell verfeinern (Abstraction-Refinement-Loop)

Beispiel aus der Software Verifikation

Beispiel (Äquivalenz von Programmen)

Beweis, dass zwei (z.B. C) Programme äquivalent sind:

```
int power3(int in) {
   int i, out_a;
   out_a = in;
   for (i=0; i<2; i++) {
      out_a = out_a * in;
   }
   return out_a;
}
```

```
int power3_new(int in) {
   int out_b;

   out_b = (in * in) * in;


   return out_b;
}
```

Vorgehen

1. `for` Schleifen entrollen, Entfernen von überflüssigen Anweisungen
2. in SSA bringen
3. Übersetzen in Formel (EUF)

Beispiel aus der Softwareverifikation

```
int power3(int in) {
   int i, out_a;
   out_a = in;
   for (i=0; i<2; i++) {
      out_a = out_a * in;
   }
   return out_a;
}
```

```
int power3_new(int in) {
   int out_b;

   out_b = (in * in) * in;


   return out_b;
}
```

Schritt 1: for Schleifen entrollen

```
out_a = in;
out_a = out_a * in;
out_a = out_a * in;
```

```
out_b = (in * in) * in;
```

Beispiel aus der Softwareverifikation

Schritt 1: for Schleifen entrollen

```
out_a = in;
out_a = out_a * in;
out_a = out_a * in;
```

```
out_b = (in * in) * in;
```

Schritt 2: In SSA bringen

$$\begin{aligned} oa_0 &= in \wedge \\ oa_1 &= oa_0 \cdot in \wedge \\ oa_2 &= oa_1 \cdot in \end{aligned}$$

$$ob_0 = (in \cdot in) \cdot in$$

Schritt 3: Übersetzen in EUF

$$\begin{aligned} \varphi_a^{UF} := a_0 &= i \wedge \\ a_1 &= G(a_0, i) \wedge \\ a_2 &= G(a_1, i) \end{aligned}$$

$$\varphi_b^{UF} := b_0 = G(G(i, i), i)$$

Zu zeigen: $\varphi_a^{UF} \wedge \varphi_b^{UF} \rightarrow a_2 = b_0$

Reduktionen von EUF nach GL

- Wir verlassen uns auf die stärkste Eigenschaft von EUF: Funktionale Konsistenz (FK)
- Damit können wir Formeln in EUF reduzieren auf Formeln in GL
- Zwei Verfahren: Ackermann Reduktion / Bryant Reduktion

Einschränkungen

Für eine übersichtlichere Darstellung

- Nur eine uninterpretierte Funktion
- Funktion hat nur einen Parameter

Kann sehr leicht erweitert werden

Ackermann Reduktion für eine Formel φ^{UF}

Füge explizite Constraints hinzu, die die funktionale Konsistenz erzwingen:

- $\texttt{fc}^E$ ist Konjunktion von FK Bedingungen
- $\texttt{flat}^E$ ist eine Version von φ^{UF}

Ausgabe ist dann: $\varphi^E = \texttt{fc}^E \rightarrow \texttt{flat}^E$.

Ackermann Reduktion

- $\mathcal{T}(\varphi)$ ersetzt in einer Formel oder Funktion φ jedes Vorkommen einer Funktionsinstanz F_i durch eine neue Variable f_i

Algorithmus: $\texttt{ackermann}(\varphi)$

Eingabe: Formel φ in UF mit m Vorkommen einer uninterpretierten Funktion F
Ausgabe: Formel $\varphi^E = \texttt{fc}^E \rightarrow \texttt{flat}^E$ in GL so dass $\models \varphi \leftrightarrow \models \varphi^E$

1. Weise jedem Vorkommen (von innen nach außen) von F einen Index i zu, $\texttt{arg}(F_i)$ ist das einzige Argument dieser Instanz von F
2. $\texttt{flat}^E = \mathcal{T}(\varphi)$
3. $\texttt{fc}^E$:

$$\bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} (\mathcal{T}(\texttt{arg}(F_i)) = \mathcal{T}(\texttt{arg}(F_j))) \rightarrow f_i = f_j$$

Will man auf Erfüllbarkeit anstelle von Gültigkeit testen, so muss man

$$\texttt{fc}^E \wedge \texttt{flat}^E \text{ testen.}$$

Beispiel für die Ackermann Reduktion

Beispiel (Ackermann Reduktion)

$$\varphi = (x_1 \neq x_2) \vee (F(x_1) = F(x_2)) \vee (F(x_1) \neq F(x_3))$$

1. $\texttt{flat}^E$

$$(x_1 \neq x_2) \vee (f_1 = f_2) \vee (f_1 \neq f_3)$$

2. $\texttt{fc}^E$

$$\begin{aligned}&(x_1 = x_2 \rightarrow f_1 = f_2) \wedge \\ &(x_1 = x_3 \rightarrow f_1 = f_3) \wedge \\ &(x_2 = x_3 \rightarrow f_2 = f_3)\end{aligned}$$

3. φ^E

$$\texttt{fc}^E \rightarrow \texttt{flat}^E$$

Mehrere Funktionen

Erweiterung des Verfahrens auf mehrere Funktionen:

$$x_1 = x_2 \rightarrow \underbrace{\overbrace{F(F(\overbrace{G(x_1)}^{g_1}))}^{f_1}}_{f_2} = \underbrace{\overbrace{F(F(\overbrace{G(x_2)}^{g_2}))}^{f_3}}_{f_4}$$

$$\texttt{flat}^E := x_1 = x_2 \rightarrow f_2 = f_4$$

$$\begin{aligned}\texttt{fc}^E := \quad & x_1 = x_2 \rightarrow g_1 = g_2 \wedge \\ & g_1 = f_1 \rightarrow f_1 = f_2 \wedge \\ & g_1 = g_2 \rightarrow f_1 = f_3 \wedge \\ & g_1 = f_3 \rightarrow f_1 = f_4 \wedge \\ & f_1 = g_2 \rightarrow f_2 = f_3 \wedge \\ & f_1 = f_3 \rightarrow f_2 = f_4 \wedge \\ & g_2 = f_3 \rightarrow f_3 = f_4\end{aligned}$$

Bryant Reduktion

- Reduktion von EUF zu GL ähnlich der Ackermann Reduktion
- Arbeitet mit Fallunterscheidung anstelle von paarweisen Constraints für die Funktionale Konsistenz
- Für Entscheidungsalgorithmen kann Bryant Reduktion von Vorteil sein, da sie kleinere Datenstrukturen erzeugt (siehe später)

Kodierung mit Fallunterscheidung

$$F = \begin{pmatrix} \texttt{case}\ x_1 = x_3 & : & f_1 \\ x_2 = x_3 & : & f_2 \\ \texttt{true} & : & f_3 \end{pmatrix}$$

wird übersetzt zu

$$\begin{aligned} &(F = f_1 \wedge x_1 = x_3) \vee \\ &(F = f_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3) \vee \\ &(F = f_3 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3). \end{aligned}$$

d.h. es wird die erste Bedingung gesucht, die zu `true` auswertet.

Bryant Reduktion

- $\mathcal{T}^\star(\varphi)$ ersetzt in einer Formel oder Funktion φ jedes Vorkommen einer Funktionsinstanz F_i durch eine neue Variable $F_i^\star$

Algorithmus: $\texttt{bryant}(\varphi)$

Eingabe: Formel φ in UF mit m Vorkommen von F
Ausgabe: Formel $\varphi^E = \texttt{fc}^E \rightarrow \texttt{flat}^E$ in GL so dass $\models \varphi \leftrightarrow \models \varphi^E$

1. Weise jedem Vorkommen (von Sub-Formeln nach außen) von F einen Index i zu, $\texttt{arg}(F_i)$ wie oben
2. $\texttt{flat}^E = \mathcal{T}^\star(\varphi)$
3. Für $i \in \{1, \ldots, m\}$ ist f_i neue Variable und $F_i^\star$ wie folgt definiert:

$$F_i^\star = \begin{pmatrix} \texttt{case}\ \mathcal{T}^\star(\texttt{arg}(F_1^\star)) = \mathcal{T}^\star(\texttt{arg}(F_i^\star)) & : & f_1 \\ \vdots & : & \vdots \\ \mathcal{T}^\star(\texttt{arg}(F_{i-1}^\star)) = \mathcal{T}^\star(\texttt{arg}(F_i^\star)) & : & f_{i-1} \\ \texttt{true} & : & f_i \end{pmatrix}$$

4. $\texttt{fc}^E = \bigwedge_{i=1}^{m} F_i^\star$

Beispiel für die Bryant Reduktion

Beispiel (Bryant Reduktion)

$$\varphi = (x_1 \neq x_2) \vee (F(x_1) = F(x_2)) \vee (F(x_1) \neq F(x_3))$$

1. $\texttt{flat}^E = (x_1 \neq x_2) \vee (F_1^\star = F_2^\star) \vee (F_1^\star \neq F_3^\star)$
2. $$F_1^\star = f_1$$
$$F_2^\star = \begin{pmatrix} \texttt{case}\ x_1 = x_2 & : & f_1 \\ \texttt{true} & : & f_2 \end{pmatrix}$$
$$F_3^\star = \begin{pmatrix} \texttt{case}\ x_1 = x_3 & : & f_1 \\ x_2 = x_3 & : & f_2 \\ \texttt{true} & : & f_3 \end{pmatrix}$$
3. $\texttt{fc}^E = F_1^\star \wedge F_2^\star \wedge F_3^\star$
4. $\varphi^E = \texttt{fc}^E \rightarrow \texttt{flat}^E$

Abstraction Refinement Loop

- Durch uninterpretierte Funktionen wird die Original Formel schwächer
- Ein Gegenbeispiel in der Abstraktion muss kein Gegenbeispiel für die Original Formal sein
- Graduelles Verfeinern von uninterpretierten Funktionen

Algorithmus: `abstractionRefinement`(φ)

Eingabe: Formel φ in einer Logik L, so dass es ein Entscheidungsverfahren für L mit UF gibt
Ausgabe: `true` wenn φ gültig ist, `false` sonst

1. $\varphi' = \mathcal{T}(\varphi)$
2. *if* $\models \varphi'$ *then return* `true`
3. *if* $\varphi' = \varphi$ *then return* `false`
4. Verfeinere φ' durch das Hinzufügen weiterer Constraints (oder Ersetzen uninterpretierter Funktionen durch interpretierte)
5. Gehe zurück zu Schritt 2

Beispiel für Abstraction Refinement

Beispiel (Abstraction Refinement)

- **Originalformel:** $\varphi = (x \cdot y) - (y \cdot x) = 0$ gültig in $\mathbb{Z}$
- **EUF Formel:** $\varphi' = G(F(x,y), F(y,x)) = 0$ nicht gültig
- Füge Constraints für die Kommutativität der Multiplikation hinzu

$$\varphi' := G(F(x,y), F(y,x)) = 0 \wedge F(x,y) = F(y,x)$$

immer noch nicht gültig

- Füge weitere Eigenschaft der Subtraktion hinzu:

$$\varphi' := G(F(x,y), F(y,x)) = 0 \wedge F(x,y) = F(y,x) \wedge G(z,z) = 0$$

nun ebenfalls gültig in $\mathbb{Z}$.

Kongruenzhülle

Algorithmus: congruenceClosure(φ)

Eingabe: 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:* Für jede Gleichung $t_1 = t_2$ sind t_1 und t_2 in der selben Äquivalenzklasse. Alle anderen Terme sind in einer eigenen Äquivalenzklasse.
   2. *Zusammenfassen:* Teilen zwei Klassen einen Term, werden sie vereinigt. (Bis zum Fixpunkt)
   3. *Funktionale Konsistenz:* Sind t_1 und t_2 in der selben Klasse dann vereinige auch die Klassen $f(t_1)$ und $f(t_2)$ (Bis zum Fixpunkt)
2. Ausgabe: UNSAT wenn es eine Ungleichung $t_1 \neq t_2$ in φ gibt, so dass t_1 und t_2 in der selben Äquivalenzklasse sind, SAT sonst.

Bemerkung: Algorithmus leicht erweiterbar auf UF mit Stelligkeit > 1

Kongruenzhülle—Beispiel

Beispiel (Congruence Closure)

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

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

Grundlegende Konzepte

Von nun an…

- Formel in NNF
- Konstanten eliminiert
- Keine uninterpretierten Funktionen mehr (Ackermann oder Bryant)

Definition (Gleichungs- & Ungleichungsmengen)

- $E_{=}(\varphi)$: Menge der positiven Atome in φ
- $E_{\neq}(\varphi)$: Menge der negativen Atome in φ

Beispiel ($E_{=}$ & $E_{\neq}$)

$$\varphi := (x_1 = x_2 \wedge x_2 = x_3) \vee (x_4 = x_5 \wedge x_5 \neq x_1) \wedge f_1 \neq f_3 \wedge (x_1 \neq x_3 \vee f_1 = f_3)$$

- $E_{=}(\varphi) = \{(x_1 = x_2), (x_2 = x_3), (x_4 = x_5), (f_1 = f_3)\}$
- $E_{\neq}(\varphi) = \{(x_5 \neq x_1), (f_1 \neq f_3), (x_1 \neq x_3)\}$

Der Gleichungsgraph

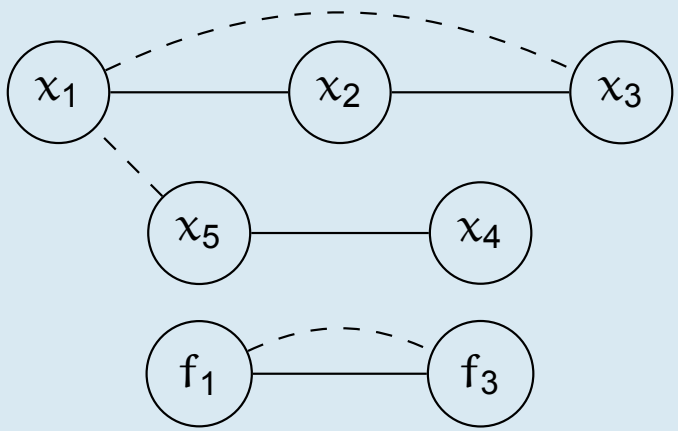
Definition (Gleichungsgraph)

Zu einer NNF Formel φ in GL gibt es einen Gleichungsgraph $G^E(\varphi)$:

- Ungerichteter Graph $(V, E_=, E_\neq)$
- Knoten in V entsprechen den Variablen in φ
- Kanten in $E_=$ entsprechen den Relationen in $E_=(\varphi)$
- Kanten in $E_\neq$ entsprechen den Relationen in $E_\neq(\varphi)$

Beispiel (Gleichungsgraph)

$$\varphi := (x_1 = x_2 \wedge x_2 = x_3) \vee (x_4 = x_5 \wedge x_5 \neq x_1) \wedge f_1 \neq f_3 \wedge (x_1 \neq x_3 \vee f_1 = f_3)$$



Gleichungs- & Ungleichungspfad

Definition (Gleichungspfad)

Ein Gleichungspfad in einem Graphen G^E ist ein Pfad, der nur aus Kanten in $E_=$ besteht.

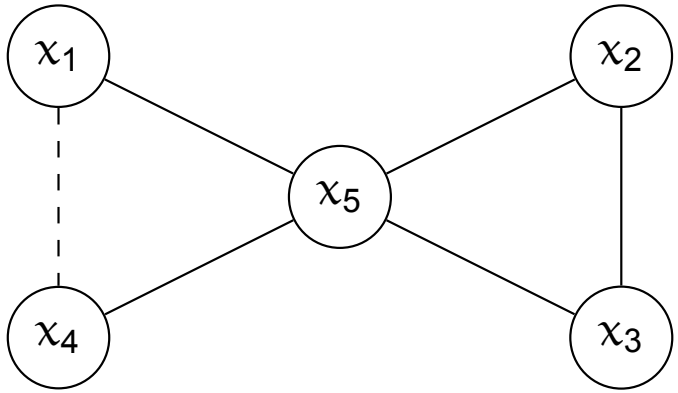
Definition (Ungleichungspfad)

Ein Ungleichungspfad in einem Graphen G^E ist ein Pfad, der nur aus Kanten in $E_=$ und einer Kante in $E_\neq$ besteht.

Notationen

- $x =^* y$: Es gibt einen Gleichungspfad zwischen x und y
- $x \neq^* y$: Es gibt einen Ungleichungspfad zwischen x und y
- einfacher . . . pfad: Der Pfad muss zyklenfrei sein

Beispiele



Beispiel (Pfade)

- $x_2 =^* x_4$ einfacher Gleichungspfad (über $x_2 \to x_5 \to x_4$)
- $x_2 \neq^* x_4$ einfacher Ungleichungspfad (über $x_2 \to x_5 \to x_1 \to x_4$)

Idee dahinter

Auf die Boolesche Struktur der Formel wird keine Rücksicht genommen:

- Gilt $x =^* y$, so *kann es sein*, dass $x = y$ gelten muss
- Gilt $x \neq^* y$, so *kann es sein*, dass $x \neq y$ gelten muss

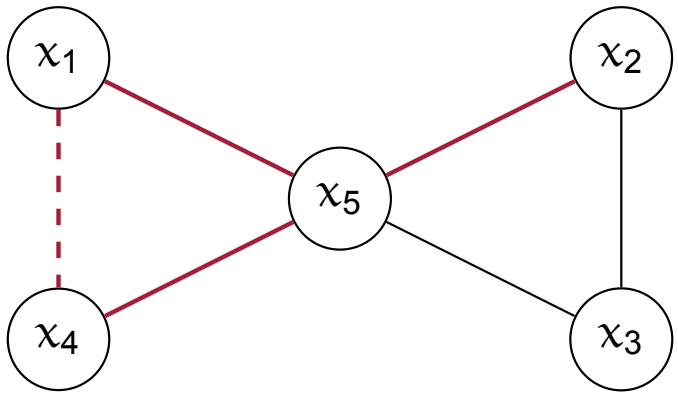
Widersprüchliche Zyklen

Gilt sowohl $x =^* y$ als auch $x \neq^* y$ (wie im letzen Beispiel), so sprechen wir von einem widersprüchlichem Zyklus.

Definition (Widersprüchlicher Zyklus)

Ein widersprüchlicher Zyklus ist ein Zyklus in einem Gleichungsgraphen mit genau einer Kante aus $E_{\neq}$.

- Für jedes Paar von Kanten x, y in einem widersprüchlichen Zyklus gilt sowohl $x =^* y$ also auch $x \neq^* y$
- Sehr wichtiges Konzept! Alle zukünftigen Entscheidungsverfahren basieren darauf.



- Einfacher widersprüchlicher Zyklus: Außer dem Start- und Endknoten kommt kein Knoten doppelt vor

Simplifikationen

Unabhängig vom Entscheidungsverfahren können GL Formeln vorher vereinfacht werden

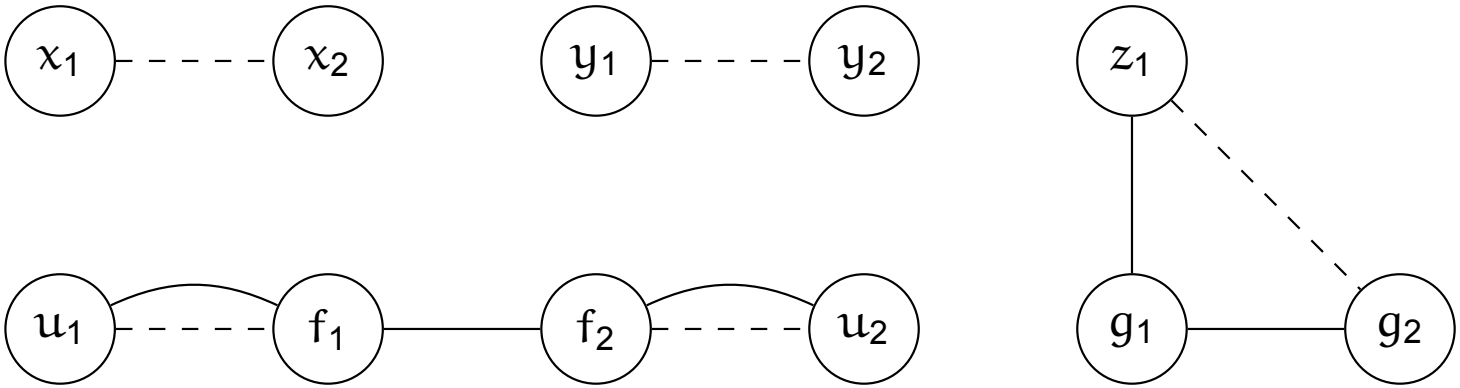
Algorithmus: `simplifyEQFormula`(φ)

Eingabe: Eine Formel φ in GL (und NNF)
Ausgabe: Eine Formel φ', erfüllbarkeitsäquivalent zu φ, und $|\varphi'| \leqslant |\varphi|$

1. $\varphi' := \varphi$
2. Konstruiere den Gleichungsgraph $G^E(\varphi')$
3. Ersetze jedes pure Atom in φ', dessen Kante nicht in einem einfachen widersprüchlichem Zyklus vorkommt, durch $\top$
4. Vereinfache φ' bezüglich der Booleschen Konstanten
5. Hat Schritt 4) die Formel verändert, gehe zurück zu Schritt 2)
6. Gib φ' zurück

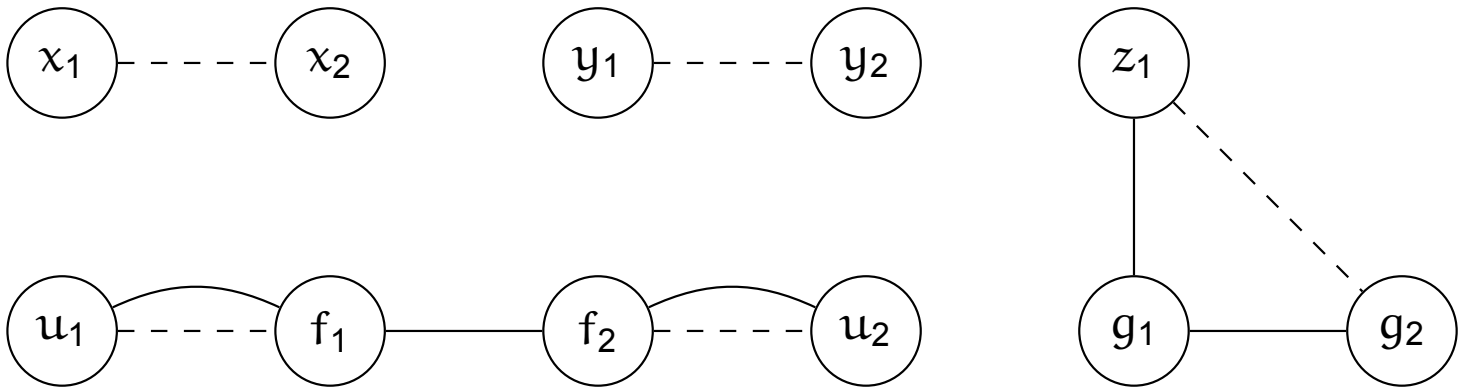
Beispiel für die Simplifikation

$$\begin{aligned}\varphi = &(u_1 \neq f_1 \vee y_1 \neq y_2 \vee f_1 = f_2) \wedge \\ &(x_1 \neq x_2 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge \\ &u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2\end{aligned}$$



Beispiel für die Simplifikation

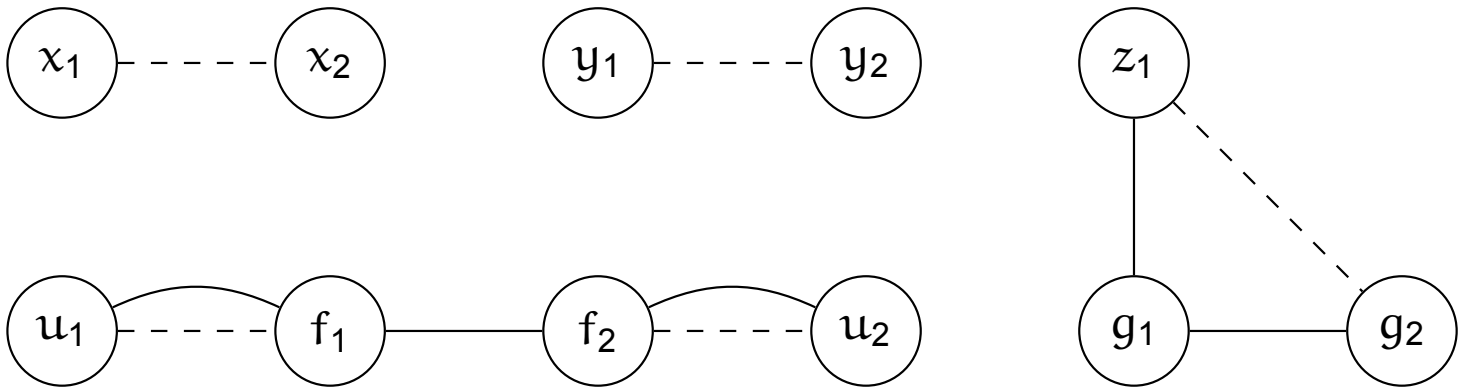
$$\begin{aligned}\varphi = &(u_1 \neq f_1 \vee y_1 \neq y_2 \vee f_1 = f_2) \wedge \\ &(x_1 \neq x_2 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge \\ &u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2\end{aligned}$$



1. Einfache widersprüchliche Zyklen: $\{z_1, g_1, g_2\}, \{u_1, f_1\}, \{f_2, u_2\}$

Beispiel für die Simplifikation

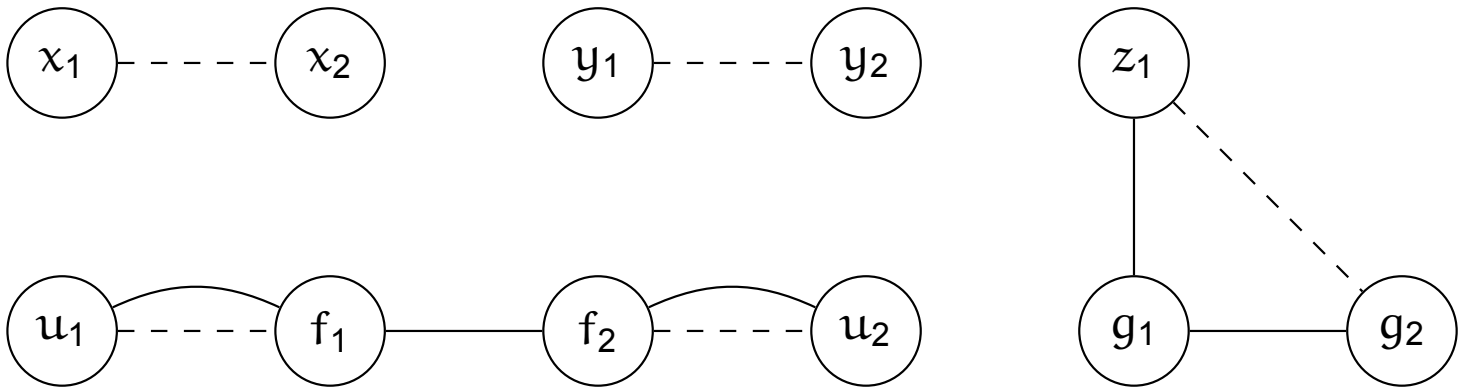
$$\begin{aligned}\varphi = &(u_1 \neq f_1 \vee y_1 \neq y_2 \vee f_1 = f_2) \wedge \\ &(x_1 \neq x_2 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge \\ &u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2\end{aligned}$$



1. Einfache widersprüchliche Zyklen: $\{z_1, g_1, g_2\}, \{u_1, f_1\}, \{f_2, u_2\}$
2. Pure Atome, deren Kante in keinem einfachen widersprüchlichen Zyklus vorkommt: $x_1 \neq x_2, y_1 \neq y_2, f_1 = f_2$

Beispiel für die Simplifikation

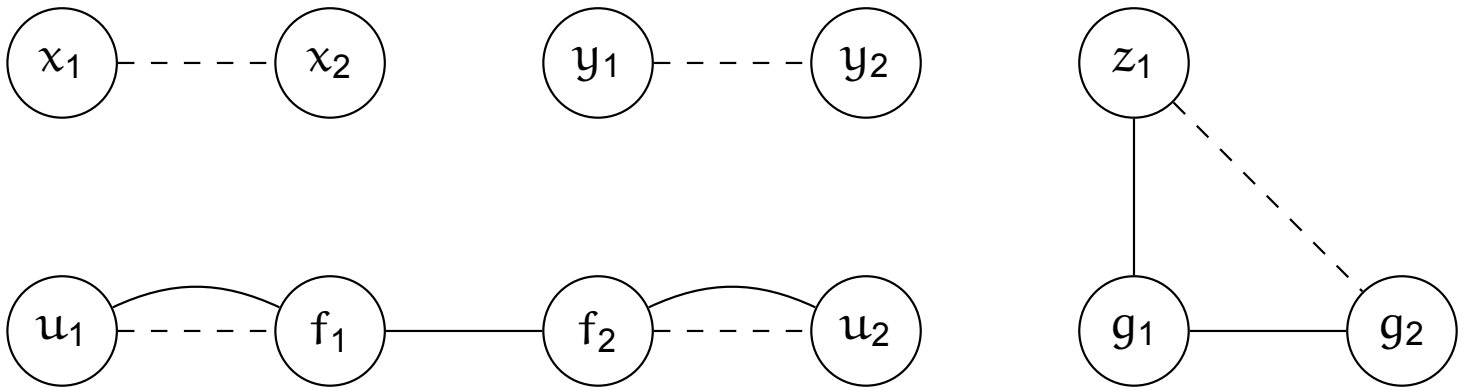
$$\begin{aligned}\varphi = &(u_1 \neq f_1 \vee \top \vee \top) \wedge \\ &(\top \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge \\ &u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2\end{aligned}$$



1. Einfache widersprüchliche Zyklen: $\{z_1, g_1, g_2\}, \{u_1, f_1\}, \{f_2, u_2\}$
2. Pure Atome, deren Kante in keinem einfachen widersprüchlichen Zyklus vorkommt: $x_1 \neq x_2, y_1 \neq y_2, f_1 = f_2$
3. Ersetze entsprechende Atome in der Formel mit $\top$

Beispiel für die Simplifikation

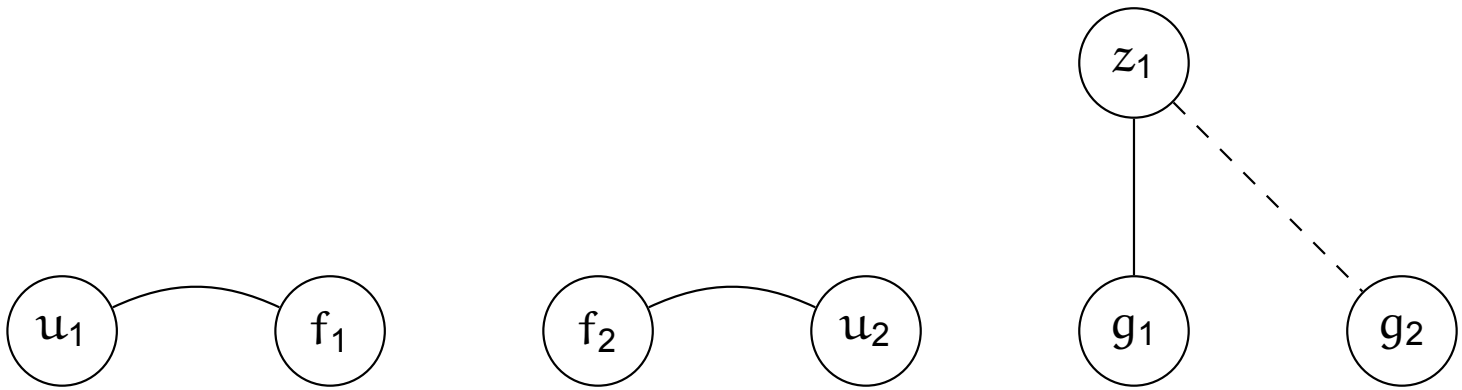
$$\varphi = u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2$$



1. Einfache widersprüchliche Zyklen: $\{z_1, g_1, g_2\}, \{u_1, f_1\}, \{f_2, u_2\}$
2. Pure Atome, deren Kante in keinem einfachen widersprüchlichen Zyklus vorkommt: $x_1 \neq x_2$, $y_1 \neq y_2$, $f_1 = f_2$
3. Ersetze entsprechende Atome in der Formel mit $\top$
4. Vereinfache die Formel

Beispiel für die Simplifikation

$$\varphi = \top$$



1. Einfache widersprüchliche Zyklen: $\{z_1, g_1, g_2\}, \{u_1, f_1\}, \{f_2, u_2\}$
2. Pure Atome, deren Kante in keinem einfachen widersprüchlichen Zyklus vorkommt: $x_1 \neq x_2, y_1 \neq y_2, f_1 = f_2$
3. Ersetze entsprechende Atome in der Formel mit $\top$
4. Vereinfache die Formel
5. Es gibt keine einfachen widersprüchlichen Zyklen mehr, d.h. alle Kanten können mit $\top$ ersetzt werden

Reduktion von GL in AL

Idee! (Bryant & Velev, 2000)

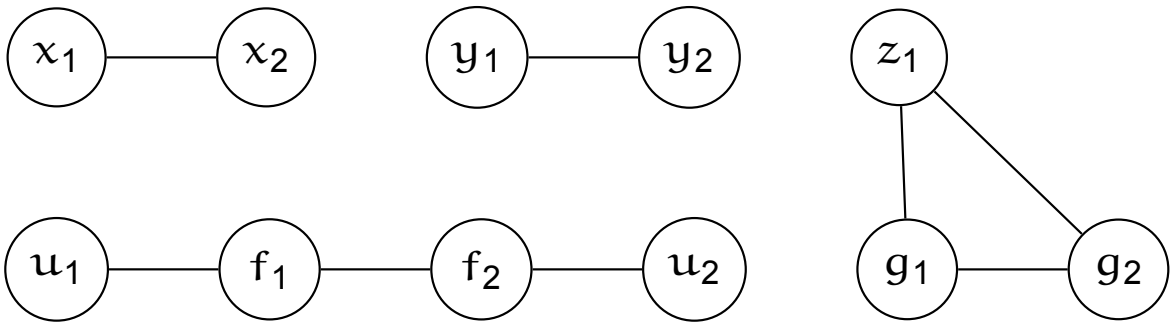
Reduziere eine Formel in GL mit einem Graph-basiertem Algorithmus in Aussagenlogik und benutze dann einen SAT Solver.

Definition (Nonpolarer Gleichungsgraph)

Zu einer NNF Formel φ in GL, ist der nonpolare Gleichungssgraph $G^E_{NP}(\varphi) = (V, E)$ wie folgt definiert:

- G^E_{NP} ist ungerichtet
- Die Knoten in V entsprechen den Variablen in φ
- Die Kanten in E entsprechen den Gleichungen und Ungleichungen in φ

D.h. $E = E_= \cup E_\neq$



Vorgehen des Algorithmus

Vorgehen

Für eine NNF Eingabeformel φ in GL generiere zwei Teilformeln $e(\varphi)$ und $\mathcal{B}_{trans}$ mit

$$\varphi \text{ erfüllbar gdw. } e(\varphi) \wedge \mathcal{B}_{trans} \text{ erfüllbar}$$

$e(\varphi)$—Das Aussagenlogische Skelett von φ

Ersetze jede Atomare Formel in φ mit einer aussagenlogischen Variablen (gleiche Formel, gleiche Variable)

$\mathcal{B}_{trans}$—Transitivitätsbedinungen

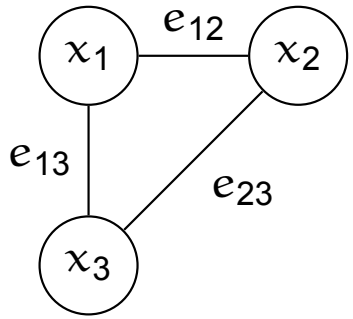
Füge für jeden Zyklus im nonpolaren Gleichungsgraphen einen Constraint hinzu:

- Sind alle Kanten bis auf eine mit `true` belegt, so kann die letzte Kante nicht mit `false` belegt werden

Beispiel

$$\varphi := x_1 = x_2 \wedge (((x_2 = x_3) \wedge (x_1 \neq x_3)) \vee (x_1 \neq x_2))$$

1. $e(\varphi) = e_{12} \wedge ((e_{23} \wedge \neg e_{13}) \vee \neg e_{12})$



2. $$\begin{aligned}\mathcal{B}_{trans} = &(e_{12} \wedge e_{23} \rightarrow e_{13}) \wedge \\ &(e_{12} \wedge e_{13} \rightarrow e_{23}) \wedge \\ &(e_{23} \wedge e_{13} \rightarrow e_{12})\end{aligned}$$

φ ist ebenso wie $e(\varphi) \wedge \mathcal{B}_{trans}$ unerfüllbar

Verbesserung der Reduktion—1

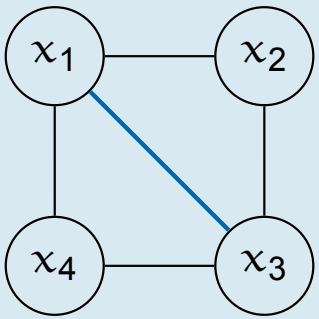
⚠ Problem!

- Für jeden Zyklus mit n Kanten müssen n Constraints hinzugefügt werden.
- Es kann eine exponentielle Anzahl an Zyklen geben

Definition (Sehne/Chord)

Eine Sehne/Chord eines Zyklus ist eine Kante, die zwei nicht-benachbarte Knoten in dem Zyklus verbindet.

Beispiel (Sehne in einem Graph)



- Betrachte Zyklus $\{x_1, x_2, x_3, x_4\}$
- $x_1 \to x_3$ ist Sehne

Verbesserung der Reduktion—2

Theorem (Bryant & Velev)

Es ist ausreichend, wenn $\mathcal{B}_{\mathrm{trans}}$ nur Transitivitätsbedingungen für einfache sehnen-freie Zyklen enthält.

Es kann aber immer noch exponentiell viele einfache sehnen-freie Zyklen geben...

Definition (Chordaler Graph)

Ein Chordaler Graph ist ein ungerichteter Graph, in dem es keinen Zyklus $\geqslant 4$ gibt, der sehnen-frei ist.

- Jeder Graph kann in Polynomzeit in einen chordalen Graphen transformiert werden
- In einem chordalen Graph gibt es nur noch einfache sehnen-freie Zyklen der Größe 3
- Pro Dreieck im Graphen brauchen wir 3 Constraints

$\Rightarrow$ Größe kubisch in der Anzahl von Variablen

Der Algorithmus

Algorithmus: eq2p1(φ)

Eingabe: NNF Formel φ in GL
Ausgabe: AL Formel φ', die erfüllbarkeitsäquivalent zu φ ist

1. Konstruiere das Aussagenlogische Skelett $e(\varphi)$
2. Konstruiere den nonpolaren Gleichungsgraphen $G^E_{NP}(\varphi)$
3. Mache $G^E_{NP}(\varphi)$ chordal
4. $\mathcal{B}_{trans} = \top$
5. Für jedes Dreieck (e_{12}, e_{13}, e_{23}) in $G^E_{NP}(\varphi)$:

$$\begin{aligned}\mathcal{B}_{trans} = \mathcal{B}_{trans} \wedge & (e_{12} \wedge e_{23} \rightarrow e_{13}) \wedge \\ & (e_{12} \wedge e_{13} \rightarrow e_{23}) \wedge \\ & (e_{23} \wedge e_{13} \rightarrow e_{12})\end{aligned}$$

6. Gib $e(\varphi) \wedge \mathcal{B}_{trans}$ zurück

Das große Bild

Wie entscheide ich die Erfüllbarkeit einer beliebigen Formel in EUF?

1. Bringe die Formel in NNF
2. Entferne alle Konstanten aus der Formel
3. Reduziere die Formel auf GL
   - Ackermann Reduktion
   - Bryant Reduktion
4. Simplifiziere die entstandene Reduktion mit Hilfe des Gleichungsgraphen
5. Wende den Algorithmus `eq2pl` auf die Formel an
6. Entscheide über die Erfüllbarkeit der Formel mit einem SAT Solver

---

Alternative

1. Besuche weiter die Vorlesung
2. Benutze einen SMT Solver (SAT Solver + Kongruenzhülle)

Literaturtipps

Literaturhinweis

- *D. Kroening & O. Strichman.* **Decision Procedures: An Algorithmic Point of View** **Chapter 3 & 4.** Springer, 2008.

- *W. Ackermann.* **Solvable cases of the Decision Problem.** Studies in Logic and the Foundations of Mathematics. Amsterdam, 1954.
- *R. Bryant & S. German & M. Velev.* **Exploiting positive equality with binary moment diagrams.** Proceedings of CAV 1999. Springer, 1999.
- *R. Bryant & M. Velev.* **Boolean Satisfiability with transitivity constraints.** Proceedings of CAV 2000. Springer, 2000.