

# Automatisches Beweisen—Vertiefung

## Entscheidbare FOL Fragmente

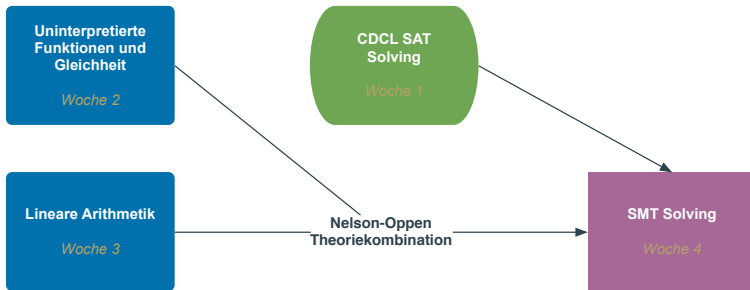
### Uninterpretierte Funktionen und Gleichheit

Christoph Zengler

Arbeitsbereich Symbolisches Rechnen  
Prof. Dr. Wolfgang Küchlin  
Universität Tübingen

22. November 2011

# Wir erinnern uns...



- 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	=	<i>Termvariable</i>	über einer Domäne $\mathcal{D}$
		<i>Konstante</i>	aus einer Domäne $\mathcal{D}$

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

- 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 Gleichheit über  $\mathcal{D}$  interpretiert

**Damit ist eine Interpretation  $M_{GL}$  fixiert.**

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

## Definition (Erfüllbarkeit in der Gleichheitslogik)

Eine Formel  $\varphi$  in GL ist erfüllbar gdw. es eine Belegung  $\beta$  gibt, so dass  $\text{holds}(M_{GL}, \beta, \varphi) = \text{true}$  gilt.



### Beispiel (Gleichungslogik)

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

Wir können statt  $\neg(\dots = \dots)$  auch  $(\dots \neq \dots)$  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 Übersetzen nach AL)
- ② Strukturinformation in Entscheidungsverfahren nutzbar

# Entfernen von Konstanten

## 🔗 Algorithmus: `removeConstants( $\varphi$ )`

**Eingabe:** Eine Formel  $\varphi$  in Gleichheitslogik

**Ausgabe:** Eine Formel  $\varphi'$  in Gleichheitslogik mit  $\varphi$  und  $\varphi'$  erfüllbarkeitsäquivalent und  $\varphi'$  konstantenfrei

- 1  $\varphi' = \varphi$
- 2 Ersetze jede Konstante  $c_i$  in  $\varphi'$  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  $\varphi'$

## 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  $\varphi'$ :  
 $\varphi' = (a = C_1 \wedge b \neq C_2 \rightarrow a \neq C_3) \leftrightarrow (a = C_3 \vee a = C_2 \rightarrow a \neq C_1)$
- 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:

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

## EBNF-Grammatik für EUF

Term	=	<i>Termvariable</i>	über einer Domäne $\mathcal{D}$
		<i>Funktionssymbol</i> ( $\{\text{Term}\}$ )	Funktion

Formel	=	Term = Term	Gleichung
		<i>Prädikatssymbol</i> ( $\{\text{Term}\}$ )	Prädikat
		...	

# Verwendung von uninterpretierten Funktionen

- $\varphi^{\text{UF}}$  ist  $\varphi$ , 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.  $\models \varphi^{\text{UF}} \rightarrow \models \varphi$

## 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 (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_b = (in * in) * in;
out_a = out_a * in;		
out_a = out_a * in;		

## Schritt 2: In SSA bringen

oa <sub>0</sub> = in ∧		ob <sub>0</sub> = (in · in) · in
oa <sub>1</sub> = oa <sub>0</sub> · in ∧		
oa <sub>2</sub> = oa <sub>1</sub> · in		

## Schritt 3: Übersetzen in EUF

$\varphi_a^{\text{UF}} := a_0 = i \wedge$		$\varphi_b^{\text{UF}} := b_0 = G(G(i, i), i)$
$a_1 = G(a_0, i) \wedge$		
$a_2 = G(a_1, i)$		

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

# 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 $\varphi^{\text{UF}}$

Füge explizite Constraints hinzu, die die funktionale Konsistenz erzwingen:

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

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

# Ackermann Reduktion

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

## 🔗 Algorithmus: ackermann( $\varphi$ )

**Eingabe:** Formel  $\varphi$  in UF mit  $m$  Vorkommen einer uninterpretierten Funktion  $F$

**Ausgabe:** Formel  $\varphi^E = fc^E \rightarrow flat^E$  in GL so dass  $\models \varphi \leftrightarrow \models \varphi^E$

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

- 3  $fc^E$ :

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

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

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

# 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  $\text{flat}^E$

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

2  $\text{fc}^E$

$$(x_1 = x_2 \rightarrow f_1 = f_2) \wedge$$

$$(x_1 = x_3 \rightarrow f_1 = f_3) \wedge$$

$$(x_2 = x_3 \rightarrow f_2 = f_3)$$

3  $\varphi^E$

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

# Mehrere Funktionen

Erweiterung des Verfahrens auf mehrere Funktionen:

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

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

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

- 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

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

wird übersetzt zu

$$(F = f_1 \wedge x_1 = x_3) \vee$$

$$(F = f_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3) \vee$$

$$(F = f_3 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3).$$

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



# Bryant Reduktion

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

## 🔗 Algorithmus: bryant( $\varphi$ )

**Eingabe:** Formel  $\varphi$  in UF mit  $m$  Vorkommen von  $F$

**Ausgabe:** Formel  $\varphi^E = fc^E \rightarrow flat^E$  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,  $\arg(F_i)$  wie oben
- 2  $flat^E = \mathcal{T}^*(\varphi)$
- 3 Für  $i \in \{1, \dots, m\}$  ist  $f_i$  neue Variable und  $F_i^*$  wie folgt definiert:

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

- 4  $fc^E = \bigwedge_{i=1}^m F_i^*$

## 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  $\text{flat}^E = (x_1 \neq x_2) \vee (F_1^* = F_2^*) \vee (F_1^* \neq F_3^*)$

②

$$\begin{aligned} F_1^* &= f_1 \\ F_2^* &= \begin{pmatrix} \text{case } x_1 = x_2 & : & f_1 \\ \text{true} & : & f_2 \end{pmatrix} \\ F_3^* &= \begin{pmatrix} \text{case } x_1 = x_3 & : & f_1 \\ x_2 = x_3 & : & f_2 \\ \text{true} & : & f_3 \end{pmatrix} \end{aligned}$$

③  $\text{fc}^E = F_1^* \wedge F_2^* \wedge F_3^*$

④  $\varphi^E = \text{fc}^E \rightarrow \text{flat}^E$

# Abstraction Refinement Loop

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

## Algorithmus: `abstractionRefinement( $\varphi$ )`

**Eingabe:** Formel  $\varphi$  in einer Logik L, so dass es ein Entscheidungsverfahren für L mit UF gibt

**Ausgabe:** true wenn  $\varphi$  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  $\varphi'$  durch das Hinzufügen weiterer Constraints (oder Ersetzen uninterpretierter Funktionen durch interpretierte)
- 5 Gehe zurück zu Schritt 2



## 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  $\mathbb{Z}$ .**

## 🔗 Algorithmus: congruenceClosure( $\varphi$ )

**Eingabe:** Konjunktion  $\varphi$  von Gleichungen und Ungleichungen über Variablen und uninterpretierten Funktionen

**Ausgabe:** SAT, wenn  $\varphi$  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  $\varphi$  gibt, so dass  $t_1$  und  $t_2$  in der selben Äquivalenzklasse sind, SAT sonst.

*Bemerkung: Algorithmus leicht erweiterbar auf UF mit Stelligkeit  $> 1$*

## 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  $\varphi$

$\Rightarrow \varphi$  ist UNSAT

## 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  $\varphi$
- $E_\neq(\varphi)$ : Menge der negativen Atome in  $\varphi$

## Beispiel ( $E_=(\varphi)$ & $E_\neq(\varphi)$ )

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

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

# Der Gleichungsgraph

## Definition (Gleichungsgraph)

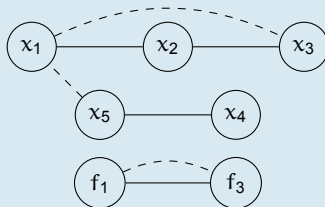
Zu einer NNF Formel  $\varphi$  in GL gibt es einen **Gleichungsgraph**  $G^E(\varphi)$ :

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



## Beispiel (Gleichungsgraph)

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





## Definition (Gleichungspfad)

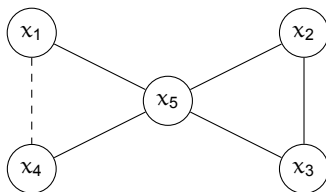
Ein **Gleichungspfad** in einem Graphen  $G^E$  ist ein Pfad, der nur aus Kanten in  $E_=$  besteht.

## 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



## Beispiel (Pfade)

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



## 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

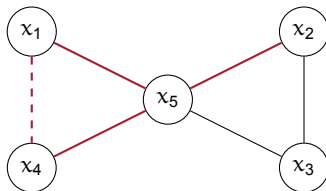
# Widersprüchliche Zyklen

Gilt sowohl  $x =^* y$  als auch  $x \neq^* y$  (wie im letzten 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

Unabhängig vom Entscheidungsverfahren können GL Formeln vorher vereinfacht werden

## 🔗 Algorithmus: `simplifyEQFormula( $\varphi$ )`

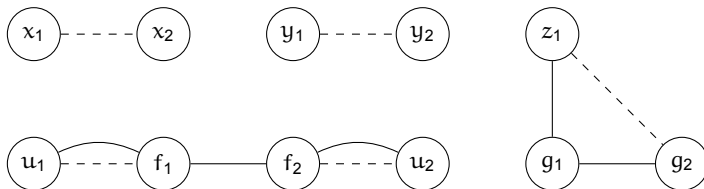
**Eingabe:** Eine Formel  $\varphi$  in GL (und NNF)

**Ausgabe:** Eine Formel  $\varphi'$ , erfüllbarkeitsäquivalent zu  $\varphi$ , und  $|\varphi'| \leq |\varphi|$

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

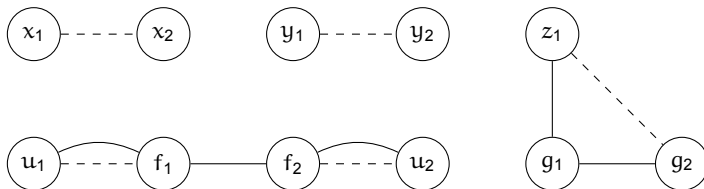
# Beispiel für die Simplifikation

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



# Beispiel für die Simplifikation

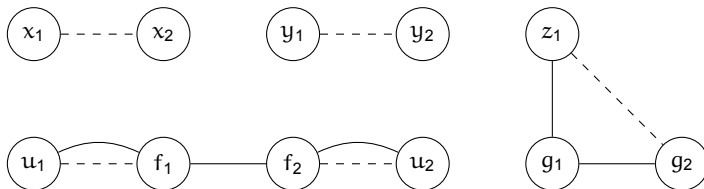
$$\begin{aligned}\varphi = & (u_1 \neq f_1 \vee y_1 \neq y_2 \vee f_1 = f_2) \wedge \\ & (x_1 \neq x_2 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge \\ & u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2\end{aligned}$$



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

# Beispiel für die Simplifikation

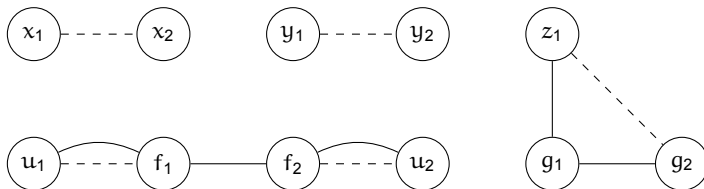
$$\begin{aligned}\varphi = & (u_1 \neq f_1 \vee y_1 \neq y_2 \vee f_1 = f_2) \wedge \\ & (x_1 \neq x_2 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge \\ & u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2\end{aligned}$$



- 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

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

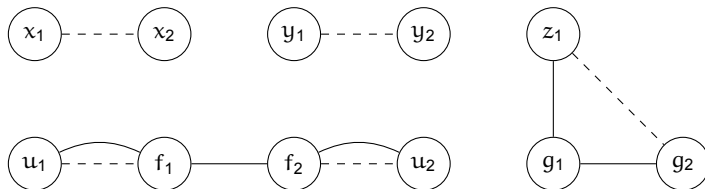


- 1 Einfache widersprüchliche Zyklen:  $\{z_1, g_1, g_2\}$ ,  $\{u_1, f_1\}$ ,  $\{f_2, u_2\}$
- 2 Pure Atome, deren Kante in keinem einfachen widersprüchlichen Zyklus vorkommt:  $x_1 \neq x_2$ ,  $y_1 \neq y_2$ ,  $f_1 = f_2$
- 3 Ersetze entsprechende Atome in der Formel mit  $\top$



# Beispiel für die Simplifikation

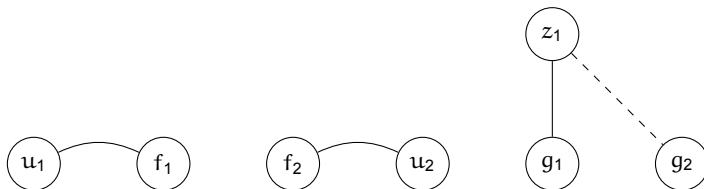
$$\varphi = u_1 = f_1 \wedge u_2 = f_2 \wedge z_1 = g_1 \wedge z_1 \neq g_2$$



- 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)

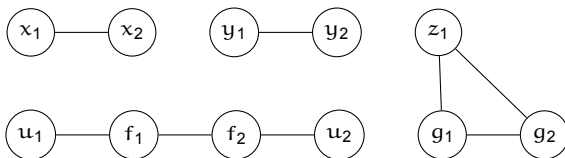
Reduziere eine Formel in GL mit einem Graph-basiertem Algorithmus in Aussagenlogik und benutze dann einen SAT Solver.

## Definition (Nonpolarer Gleichungsgraph)

Zu einer NNF Formel  $\varphi$  in GL, ist der **nonpolare Gleichungsgraph**  $G_{NP}^E(\varphi) = (V, E)$  wie folgt definiert:

- $G_{NP}^E$  ist ungerichtet
- Die Knoten in  $V$  entsprechen den Variablen in  $\varphi$
- Die Kanten in  $E$  entsprechen den Gleichungen und Ungleichungen in  $\varphi$

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



## Vorgehen

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

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

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

Ersetze jede Atomare Formel in  $\varphi$  mit einer aussagenlogischen Variablen (gleiche Formel, gleiche Variable)

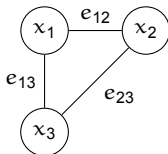
### $\mathcal{B}_{\text{trans}}$ —Transitivitätsbedingungen

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

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

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



2

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

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

# Verbesserung der Reduktion—1

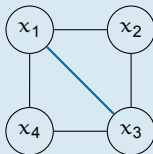
## ⚠ Problem!

- 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  $\{x_1, x_2, x_3, x_4\}$
- $x_1 \rightarrow x_3$  ist Sehne

# Verbesserung der Reduktion—2

## Theorem (Bryant & Velev)

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

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

## Definition (Chordaler Graph)

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

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

## 🔗 Algorithmus: eq2p1( $\varphi$ )

**Eingabe:** NNF Formel  $\varphi$  in GL

**Ausgabe:** AL Formel  $\varphi'$ , die erfüllbarkeitsäquivalent zu  $\varphi$  ist

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

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

- 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)



## 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.*