Formale Spezifikation und Verifikation

Wintersemester 2024

Prof. Dr. Gidon Ernst
gidon.ernst@lmu.de

Software and Computational Systems Lab
Ludwig-Maximilians-Universität München, Germany

December 16, 2024

Automatisiertes Beweisen (SAT/SMT)

Ritter und Schurken (nach Raymond Smullyan)

Auf der Insel der Ritter und Schurken sprechen Ritter immer die Wahrheit, während Schurken immer lügen. Du triffst Alex und Chris, jeder ist entweder ein Ritter oder ein Schurke, aber man sieht es ihnen nicht an.

Alex sagt: *“Wir sind beide Schurken”*

Ritter und Schurken (nach Raymond Smullyan)

Auf der Insel der Ritter und Schurken sprechen Ritter immer die Wahrheit, während Schurken immer lügen. Du triffst Alex und Chris, jeder ist entweder ein Ritter oder ein Schurke, aber man sieht es ihnen nicht an.

Alex sagt: “Genau dann wenn Chris ein Schurke ist, bin ich ein Schurke.”
Chris sagt: “Wir sind verschiedenen Typs.”

Ritter und Schurken (nach Raymond Smullyan)

Auf der Insel der Ritter und Schurken sprechen Ritter immer die Wahrheit, während Schurken immer lügen. Du triffst Alex, Chris und Erin, jeder ist entweder ein Ritter oder ein Schurke, aber man sieht es ihnen nicht an.

Alex sagt:	*"Chris ist ein Ritter, aber Erin ist ein Schurke"*
Chris sagt:	*"Zwei von uns sind Ritter"*
Erin sagt:	*"Alex und ich sind beide Ritter"*

Ritter und Schurken (nach Raymond Smullyan)

Auf der Insel der Ritter und Schurken sprechen Ritter immer die Wahrheit, während Schurken immer lügen. Du triffst Alex, Chris und Erin, jeder ist entweder ein Ritter oder ein Schurke, aber man sieht es ihnen nicht an.

Alex sagt:	*"Chris ist ein Ritter, aber Erin ist ein Schurke"*
Chris sagt:	*"Zwei von uns sind Ritter"*
Erin sagt:	*"Alex und ich sind beide Ritter"*

Modellierung als Aussagenlogisches Problem

- Propositionen `alex`, `chris`, `erin` wahr gdw. Person Ritter ist und daher die Wahrheit sagt
- $\mathtt{alex} \iff \mathtt{chris} \land \neg\mathtt{erin}$
- $\mathtt{chris} \iff (\neg\mathtt{alex} \land \mathtt{chris} \land \mathtt{erin}) \lor (\mathtt{alex} \land \neg\mathtt{chris} \land \mathtt{erin}) \lor (\mathtt{alex} \land \mathtt{chris} \land \neg\mathtt{erin})$
- $\mathtt{erin} \iff \mathtt{alex} \land \mathtt{erin}$

Lösung mit Hilfe von SMT Solvern

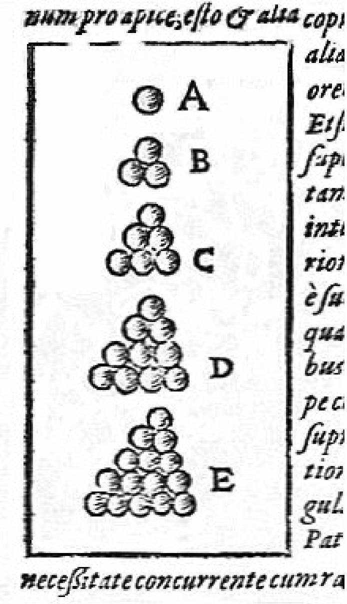
```
; SMT-LIB Format

; Einzige Lösung?

(set-option :produce-models true)

(declare-const alex  Bool)
(declare-const chris Bool)
(declare-const erin  Bool)

(assert (= alex
       (and chris (not erin))))

(assert (= chris
       (or (and (not alex) chris erin)
           (and alex (not chris) erin)
           (and alex chris (not erin)))))

(assert (= erin
        (and alex erin)))
```

Lösung mit Hilfe von SMT Solvern

```
; SMT-LIB Format

; Einzige Lösung?

(set-option :produce-models true)

(declare-const alex  Bool)
(declare-const chris Bool)
(declare-const erin  Bool)

(assert (= alex
       (and chris (not erin))))

(assert (= chris
       (or (and (not alex) chris erin)
           (and alex (not chris) erin)
           (and alex chris (not erin)))))

(assert (= erin
        (and alex erin)))
```

```
$ z3 ritter.smt2
sat
(model
  (define-fun alex () Bool
      true)
  (define-fun chris () Bool
      true)
  (define-fun erin () Bool
      false))
```

Kepler Conjecture [Hales et al, 1998 & 2014]



Computational Mathematics

Liquid tensor experiment

Liquid tensor experiment

Posted on December 5, 2020 by xenaproject

This is a guest post, written by Peter Scholze, explaining a liquid real vector space mathematical formalisation challenge. For a pdf version of the challenge, see here. For comments about formalisation, see section 6. Now over to Peter.

1. The challenge

I want to propose a challenge: Formalize the proof of the following theorem.

Theorem 1.1 *(Clausen-S.) Let $0 < p' < p \leq 1$ be real numbers, let S be a profinite set, and let V be a p-Banach space. Let $\mathcal{M}_{p'}(S)$ be the space of p'-measures on S. Then*

$$\mathrm{Ext}^i_{\mathrm{Cond(Ab)}}(\mathcal{M}_{p'}(S), V) = 0$$

for $i \geq 1$.

RNA-folding Problem [Ganesh et al, 2012]

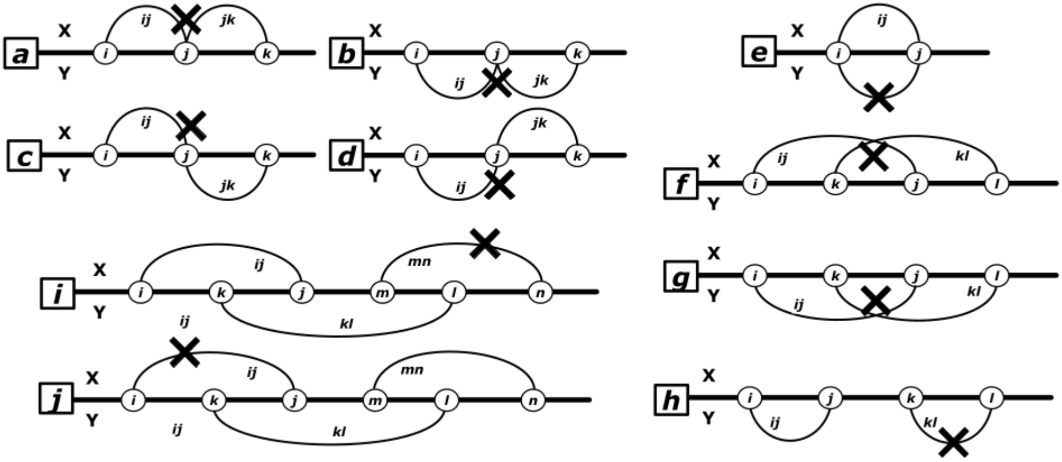


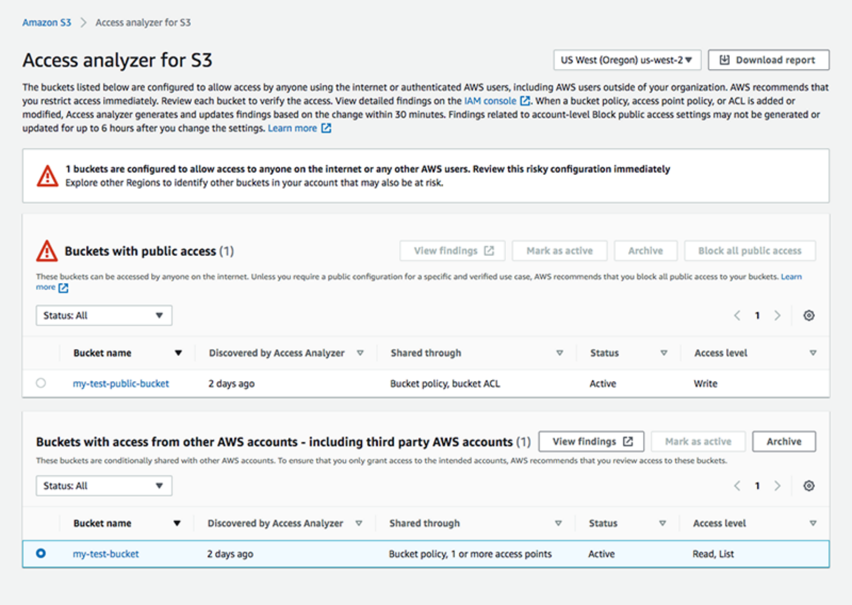
Fig. 1. RNA Constraints

Microsoft SLAM (2001–)

SLAM is a project for checking that software satisfies critical behavioral properties of the interfaces it uses and to aid software engineers in designing interfaces and software that ensure reliable and correct functioning. Static Driver Verifier is a tool in the Windows Driver Development Kit that uses the SLAM verification engine.

"Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability." **Bill Gates, April 18, 2002. Keynote address at WinHec 2002**

Amazon AWS access analyzer (2017–)



Programmsynthese

```
data BST a where
    Empty :: BST a
    Node  :: x: a -> l: BST {a | _v < x} -> r: BST {a | x < _v} -> BST a

measure keys :: BST a -> Set a where
    Empty -> []
    Node x l r -> keys l + keys r + [x]
```

Synthese von Implementierungen:

```
insert :: x: a -> t: BST a -> {BST a | keys _v == keys t + [x]}
insert = ??
```

Tools

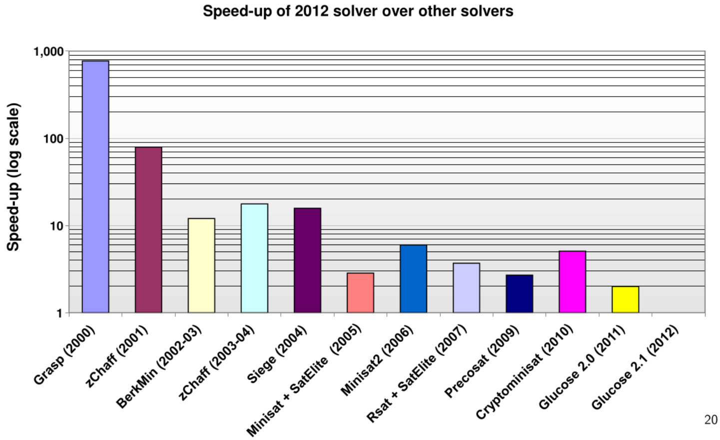
- Liquid Haskell https://ucsd-progsys.github.io/liquidhaskell-blog/
- Synquid http://comcom.csail.mit.edu/comcom/#Synquid
- Leon https://leon.epfl.ch/

Superoptimierung, z.B. [Schkufza et al, 2013]

```
# gcc -O3

.L0:
movslq ecx,rcx
leaq (rsi,rcx,4),r8
leaq 1(rcx),r9
movl (r8),eax
imull edi,eax
addl (rdx,rcx,4),eax
movl eax,(r8)
leaq (rsi,r9,4),r8
movl (r8),eax
imull edi,eax
addl (rdx,r9,4),eax
leaq 2(rcx),r9
addq 3,rcx
movl eax,(r8)
leaq (rsi,r9,4),r8
movl (r8),eax
imull edi,eax
addl (rdx,r9,4),eax
movl eax,(r8)
leaq (rsi,rcx,4),rax
imull (rax),edi
addl (rdx,rcx,4),edi
movl edi,(rax)
```

```
# STOKE

.L0:
movd edi,xmm0
shufps 0,xmm0,xmm0
movups (rsi,rcx,4),xmm1
pmullw xmm1,xmm0
movups (rdx,rcx,4),xmm1
paddw xmm1,xmm0
movups xmm0,(rsi,rcx,4)
```

SAT Revolution (Aussagenlogik)



Moshe Y. Vardi, Rice University

"SAT Solver lösen NP-schwierige Probleme als ob diese in P wären" (oft)

SMT Revolution (Prädikatenlogik)

Z3: An efficient SMT solver

L De Moura, N Bjørner - International conference on Tools and Algorithms …, 2008 - Springer

Abstract Satisfiability Modulo Theories (**SMT**) problem is a decision problem for logical first order formulas with respect to combinations of background theories such as: arithmetic, bit-vectors, arrays, and uninterpreted functions. **Z3** is a new and **efficient SMT Solver** freely …

Zitiert von: 6341 Ähnliche Artikel Alle 19 Versionen

SMT-COMP 2019:
Alt-Ergo, AProVe, Boolector, COLIBRI, CVC 4,
MathSAT, Minkeyrink, OpenSMT2, Q3B, SMTInterpol,
SMT-RAT, SPASS, STP, Vampire, veriT, Yices, Z3

SMT Revolution (Prädikatenlogik)

SAT/SMT

Satisfiability von Aussagenlogischen Formeln ϕ:
Gibt es eine Belegung s der Propositionen, sodass $s \models \phi$?

SMT = Satisfiability modulo Theory (SAT + Theorien)
Gibt es eine Belegung s aller Variablen, sodass $s \models \phi$?

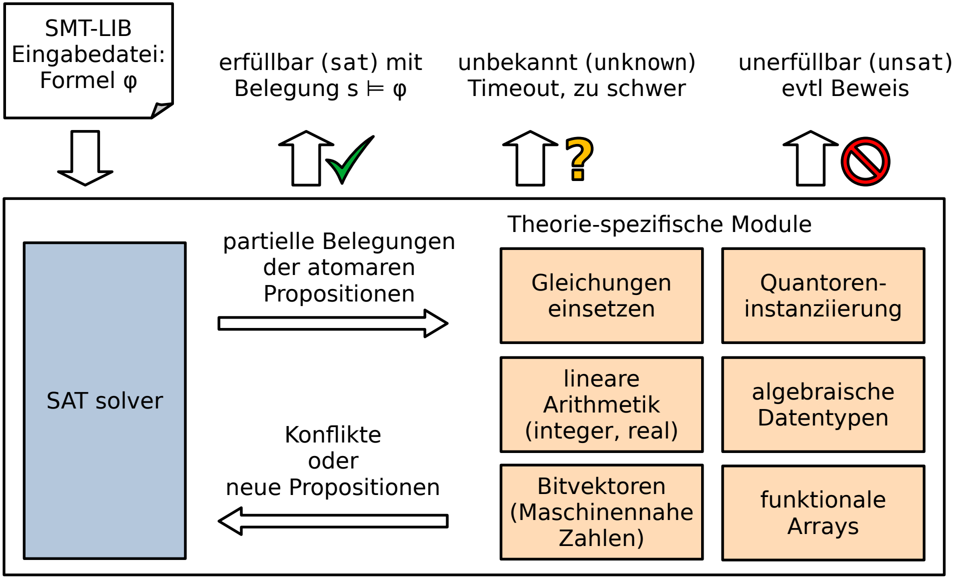
Theorie = mathematischer Datentyp + Operationen, z.B.

- Zahlen $\mathbb{Z}$ oder $\mathbb{R}$ mit $+, -, \cdot$
- Mengen, Sequenzen, funktionale Arrays, Baumstrukturen
- Quantoren $\forall, \exists$, mathematische Funktionen

Anwendungen, z.B. wenn Belegung s = Programmzustand

- Erfüllbarkeit = Erreichbarkeit von (Fehler-)Zuständen
- Allgemeingültigkeit von Implikationen $P \Rightarrow Q$ (Hoare: Konsequenzregel)

Satisfiability Modulo Theory



Automatisiertes Beweisen (SAT/SMT)

Aussagenlogik

Aussagenlogik: Syntax von Formeln

Für eine Menge $\mathcal{A} = \{A, B, C, \ldots\}$ an *atomaren Propositionen* definieren wir die Grammatik von Formeln:

$$
\begin{array}{rcll}
\text{Formeln } \phi & ::= & \texttt{true} & \\
& \mid & \texttt{false} & \\
& \mid & A & \text{atomare Propositionen } A \in \mathcal{A} \\
& \mid & \neg\, \phi & \text{Negation (nicht)} \\
& \mid & \phi_1 \wedge \phi_2 & \text{Konjunktion (beide)} \\
& \mid & \phi_1 \vee \phi_2 & \text{Disjunktion (mindestens eines)} \\
& \mid & \phi_1 \implies \phi_2 & \text{Implikation (wenn-dann)} \\
& \mid & \phi_1 \iff \phi_2 & \text{Äquivalenz (genau-dann-wenn)}
\end{array}
$$

- Die Symbole $\neg, \wedge, \vee, \implies, \iff$ heißen *Junktoren*
- "Punkt-vor-Strich": Präzedenzen in dieser Reihenfolge, evtl Klammern setzen
- $A \implies B \implies C$ ist $A \implies (B \implies C)$

Aussagenlogik: Semantik

Eine Belegung $s : \mathcal{A} \mapsto \mathbb{B}$ weist jeder atomaren Proposition $A \in \mathcal{A}$ einen Wahrheitswert $s(A)$ zu. Beispielbelegung für $\mathcal{A} = \{\texttt{alex}, \texttt{chris}, \texttt{erin}\}$

$$s = \{\texttt{alex} \mapsto \mathit{false}, \texttt{chris} \mapsto \mathit{true}, \texttt{erin} \mapsto \mathit{true}\}$$

Eine Belegung s erfüllt eine Formel ϕ, geschrieben $s \models \phi$, falls

$s \models \texttt{true}$ immer $\qquad s \models \texttt{false}$ niemals

$s \models A$ genau wenn $s(A) = \mathit{true}$

$s \models \neg\phi$ genau falls nicht $s \models \phi$

$s \models \phi_1 \wedge \phi_2$ genau wenn $s \models \phi_1$ und $s \models \phi_2$

$s \models \phi_1 \vee \phi_2$ genau wenn $s \models \phi_1$ oder $s \models \phi_2$

$s \models \phi_1 \implies \phi_2$ genau wenn $s \models \phi_1$ impliziert dass $s \models \phi_2$

$s \models \phi_1 \iff \phi_2$ genau wenn $s \models \phi_1$ und $s \models \phi_2$ oder beide nicht gelten

Hier: Junktoren durch natürliche Sprache definiert (alternativ: Wahrheitstabellen)
Man sagt auch s ist "Modell" von ϕ

Aussagenlogik: Begriffe und Definitionen

Gegeben: Definition der Semantik $s \models \phi$ von ϕ für Belegungen s

Erfüllbarkeit und Allgemeingültigkeit

- Falls $s \models \phi$ sagt man: s erfüllt ϕ, ϕ gilt in s
- ϕ ist *erfüllbar*, falls es ein s gibt mit $s \models \phi$
- ϕ ist *allgemeingültig*, geschrieben $\models \phi$, falls für alle s gilt $s \models \phi$
  Solche Formeln heißen auch *Tautologien*
- ϕ ist *unverfüllbar*, falls es kein s gibt mit $s \models \phi$
- Zusammenhang: ϕ ist allgemeingültig falls $\neg\phi$ unerfüllbar

Aussagenlogik: Begriffe und Definitionen

Gegeben: Definition der Semantik $s \models \phi$ von ϕ für Belegungen s

Erfüllbarkeit und Allgemeingültigkeit

- Falls $s \models \phi$ sagt man: s erfüllt ϕ, ϕ gilt in s
- ϕ ist *erfüllbar*, falls es ein s gibt mit $s \models \phi$
- ϕ ist *allgemeingültig*, geschrieben $\models \phi$, falls für alle s gilt $s \models \phi$
  Solche Formeln heißen auch *Tautologien*
- ϕ ist *unverfüllbar*, falls es kein s gibt mit $s \models \phi$
- Zusammenhang: ϕ ist allgemeingültig falls $\neg\phi$ unerfüllbar

Äquivalenz: ϕ und ψ sind *äquivalent*

- $s \models \phi$ genau wenn $s \models \psi$ für alle Belegungen s
- alternativ: $\phi \iff \psi$ ist allgemeingültig, d.h. $s \models \phi \iff \psi$

Beispiele (I)

- `true` ist Tautologie und auch erfüllbar (mit jedem s)
- `false` ist unerfüllbar
- A ist erfüllbar (mit $s = \{A \mapsto \mathit{true}, \ldots\}$)
- $A \wedge \neg B$ ist ebenfalls erfüllbar (mit $s = \{A \mapsto \mathit{true}, B \mapsto \mathit{false}, \ldots\}$)
- $A \wedge \neg A$ ist nicht erfüllbar und damit äquivalent zu `false`
  die Aussage ist in sich widersprüchlich
- $A \vee \neg A$ ist allgemeingültig und damit äquivalent zu `true`
  Satz des ausgeschlossenen Dritten

Beispiele (II)

Gegeben eine Belegung

$$s = \{\texttt{alex} \mapsto \mathit{false}, \texttt{chris} \mapsto \mathit{true}, \texttt{erin} \mapsto \mathit{true}\}$$

Ritter und Schurken

- $\phi_1 = \big(\texttt{alex} \iff \texttt{chris} \land \neg\texttt{erin}\big)$
- $\phi_2 = \big(\texttt{chris} \iff (\neg\texttt{alex}\land\texttt{chris}\land\texttt{erin})\lor(\texttt{alex}\land\neg\texttt{chris}\land\texttt{erin})\lor(\texttt{alex}\land\texttt{chris}\land\neg\texttt{erin})\big)$
- $\phi_3 = \big(\texttt{erin} \iff \texttt{alex} \land \texttt{erin}\big)$

Dann gilt $s \models \phi_1$ und $s \models \phi_2$, aber nicht $s \models \phi_3$

Wie finden wir erfüllende Belegungen s für ϕ bzw. beweisen dass es keine gibt?

✗ Alle $2^{|\mathcal{A}|}$ Möglichkeiten durchprobieren

✓ Berechnung auf Formeln anstatt mit Belegungen

✓ Clevere Algorithmen: Vereinfachen & Abkürzungen

Beispiele (III)

Ritter und Schurken

- $\phi_1 = \big(\texttt{alex} \iff \texttt{chris} \land \neg\texttt{erin}\big)$
- $\phi_2 = \big(\texttt{chris} \iff (\neg\texttt{alex}\land\texttt{chris}\land\texttt{erin})\lor(\texttt{alex}\land\neg\texttt{chris}\land\texttt{erin})\lor(\texttt{alex}\land\texttt{chris}\land\neg\texttt{erin})\big)$
- $\phi_3 = \big(\texttt{erin} \iff \texttt{alex} \land \texttt{erin}\big)$

Mit Nachdenken suchen wir eine Erfüllende Belegungen

- Einsetzen der Äquivalenz ϕ_1 in ϕ_3 $\quad \phi_3' = (\texttt{erin} \iff (\texttt{chris} \land \neg\texttt{erin}) \land \texttt{erin})$
- Umklammern, Vereinfachen $\quad \phi_3'' = (\texttt{erin} \iff \texttt{false})$
- Einsetzen der Äquivalenz ϕ_3'' in ϕ_1 $\quad \phi_1' = (\texttt{alex} \iff \texttt{chris})$
- Einsetzen der Äquivalenz ϕ_3'' in ϕ_2 $\quad \phi_2' = (\texttt{chris} \iff \texttt{alex} \land \texttt{chris})$
- Vereinfachen ergibt wieder $\quad \phi_2'' = (\texttt{chris} \iff \texttt{alex}) = \phi_1'$

Beispiele (III)

Ritter und Schurken

- $\phi_1 = \big(\texttt{alex} \iff \texttt{chris} \land \neg\texttt{erin}\big)$
- $\phi_2 = \big(\texttt{chris} \iff (\neg\texttt{alex}\land\texttt{chris}\land\texttt{erin})\lor(\texttt{alex}\land\neg\texttt{chris}\land\texttt{erin})\lor(\texttt{alex}\land\texttt{chris}\land\neg\texttt{erin})\big)$
- $\phi_3 = \big(\texttt{erin} \iff \texttt{alex} \land \texttt{erin}\big)$

Mit Nachdenken suchen wir eine Erfüllende Belegungen

- Einsetzen der Äquivalenz ϕ_1 in ϕ_3 — $\phi_3' = (\texttt{erin} \iff (\texttt{chris} \land \neg\texttt{erin}) \land \texttt{erin})$
- Umklammern, Vereinfachen — $\phi_3'' = (\texttt{erin} \iff \texttt{false})$
- Einsetzen der Äquivalenz ϕ_3'' in ϕ_1 — $\phi_1' = (\texttt{alex} \iff \texttt{chris})$
- Einsetzen der Äquivalenz ϕ_3'' in ϕ_2 — $\phi_2' = (\texttt{chris} \iff \texttt{alex} \land \texttt{chris})$
- Vereinfachen ergibt wieder — $\phi_2'' = (\texttt{chris} \iff \texttt{alex}) = \phi_1'$

Lösungen

- $s_1 = \{\texttt{alex} \mapsto \mathit{false}, \texttt{chris} \mapsto \mathit{false}, \texttt{erin} \mapsto \mathit{false}\}$
- $s_2 = \{\texttt{alex} \mapsto \mathit{true}, \texttt{chris} \mapsto \mathit{true}, \texttt{erin} \mapsto \mathit{false}\}$

Beispiele (IV) — Äquivalenzen

Vereinfachungsmöglichkeiten

- $A \land \texttt{true} \iff A$ und $A \lor \texttt{true} \iff \texttt{true}$
- $A \land \texttt{false} \iff \texttt{false}$ und $A \lor \texttt{false} \iff A$
- $A \land A \iff A$, usw. . .

Eigenschaften der Implikation

- $(A \implies B) \iff (\neg A \lor B)$
- $\neg A \iff (A \implies \texttt{false})$
- $(A \implies (B \implies C)) \iff (A \land B \implies C)$

Regeln von De Morgan: Negation nach innen ziehen

- $\neg(A \land B) \iff (\neg A \lor \neg B)$
- $\neg(A \lor B) \iff (\neg A \land \neg B)$

Distributionsgesetze

- $A \land (B \lor C) \iff (A \land B) \lor (A \land C)$
- $A \lor (B \land C) \iff (A \lor B) \land (A \lor C)$

Normalformen: Motivation

Pausieren und Nachdenken: Finde erfüllende Belegung für

- ϕ: $(A \land B \land \neg C) \lor (\neg A \land \neg B)$
- ψ: $(A \lor B \lor \neg C) \land (\neg A \lor \neg B)$

Normalformen: Motivation

Pausieren und Nachdenken: Finde erfüllende Belegung für

- ϕ: $(A \land B \land \neg C) \lor (\neg A \land \neg B)$
- ψ: $(A \lor B \lor \neg C) \land (\neg A \lor \neg B)$

Für ϕ sind die linke/rechte Teilformel unabhängig,
wir können wir mögliche Lösungen direkt ablesen:

- $s_1 = \{A \mapsto \mathit{true}, B \mapsto \mathit{true}, C \mapsto \mathit{false}\}$

Normalformen: Motivation

Pausieren und Nachdenken: Finde erfüllende Belegung für

- ϕ: $(A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B)$
- ψ: $(A \vee B \vee \neg C) \wedge (\neg A \vee \neg B)$

Für ϕ sind die linke/rechte Teilformel unabhängig,
wir können wir mögliche Lösungen direkt ablesen:

- $s_1 = \{A \mapsto \mathit{true}, B \mapsto \mathit{true}, C \mapsto \mathit{false}\}$
- $s_{2/3} = \{A \mapsto \mathit{false}, B \mapsto \mathit{false}, C \mapsto b\}$ für $b \in \mathbb{B}$ (C spielt keine Rolle)

Normalformen: Motivation

Pausieren und Nachdenken: Finde erfüllende Belegung für

- ϕ: $(A \land B \land \neg C) \lor (\neg A \land \neg B)$
- ψ: $(A \lor B \lor \neg C) \land (\neg A \lor \neg B)$

Für ϕ sind die linke/rechte Teilformel unabhängig,
wir können wir mögliche Lösungen direkt ablesen:

- $s_1 = \{A \mapsto \mathit{true}, B \mapsto \mathit{true}, C \mapsto \mathit{false}\}$
- $s_{2/3} = \{A \mapsto \mathit{false}, B \mapsto \mathit{false}, C \mapsto b\}$ für $b \in \mathbb{B}$ (C spielt keine Rolle)

In ψ müssen beide Teilformeln erfüllt sein und es gibt Abhängigkeiten

- Wähle z.B. $A \mapsto \mathit{true}$ dann muss zwingend $B \mapsto \mathit{false}$

Normalformen: Motivation

Pausieren und Nachdenken: Finde erfüllende Belegung für

- ϕ: $(A \land B \land \neg C) \lor (\neg A \land \neg B)$
- ψ: $(A \lor B \lor \neg C) \land (\neg A \lor \neg B)$

Für ϕ sind die linke/rechte Teilformel unabhängig,
wir können wir mögliche Lösungen direkt ablesen:

- $s_1 = \{A \mapsto \mathit{true}, B \mapsto \mathit{true}, C \mapsto \mathit{false}\}$
- $s_{2/3} = \{A \mapsto \mathit{false}, B \mapsto \mathit{false}, C \mapsto b\}$ für $b \in \mathbb{B}$ (C spielt keine Rolle)

In ψ müssen beide Teilformeln erfüllt sein und es gibt Abhängigkeiten

- Wähle z.B. $A \mapsto \mathit{true}$ dann muss zwingend $B \mapsto \mathit{false}$
- Alternativ $B \mapsto \mathit{true}$ dann muss zwingend $A \mapsto \mathit{false}$

Zum Nachdenken: Sind die Formeln äquivalent?

Negationsnormalform (NNF)

Definition: Ein *Literal* l ist eine Proposition A oder eine negierte Proposition $\neg A$

Definition: Formeln in *Negationsnormalform* (NNF)

- `true`, `false` und Literale A, $\neg A$ sind in NNF
- Wenn ϕ, ψ in NNF sind dann auch $\phi \land \psi$ und $\phi \lor \psi$

Berechnung: Anwenden der folgenden Regeln bis NNF hergestellt ist

- $\neg$`true` $\leadsto$ `false` und $\neg$`false` $\leadsto$ `true`
- $\phi \implies \psi \quad \leadsto \quad \neg\phi \lor \psi$
- $\phi \iff \psi \quad \leadsto \quad (\phi \land \psi) \lor (\neg\phi \land \neg\psi)$ oder
  $\leadsto \quad (\neg\phi \lor \psi) \land (\phi \lor \neg\psi)$
- $\neg(\phi \land \psi) \quad \leadsto \quad \neg\phi \lor \neg\psi$
- $\neg(\phi \lor \psi) \quad \leadsto \quad \neg\phi \land \neg\psi$

Disjunktive Normalform (DNF)

Definition: Eine Formel ϕ ist in *disjunktiver Normalform* (DNF), falls ϕ als Disjunktion ("oder") zwischen Konjunktionen von Literalen gegeben ist

Beispiel

- ϕ: $(A \land B \land \neg C) \lor (\neg A \land \neg B)$

Berechnung aus NNF durch Distributionsregel

- $\phi \land (\psi \lor \chi) \quad \leadsto \quad (\phi \land \psi) \lor (\phi \land \chi)$

Disjunktive Normalform (DNF)

Definition: Eine Formel ϕ ist in *disjunktiver Normalform* (DNF), falls ϕ als Disjunktion ("oder") zwischen Konjunktionen von Literalen gegeben ist

Beispiel

- ϕ: $(A \land B \land \neg C) \lor (\neg A \land \neg B)$

Berechnung aus NNF durch Distributionsregel

- $\phi \land (\psi \lor \chi) \quad \rightsquigarrow \quad (\phi \land \psi) \lor (\phi \land \chi)$

Eigenschaften

✓ Erfüllbarkeit lässt sich in $O(n)$ prüfen (n Größe der Formel):
Es reicht aus wenn mindestens ein Disjunktionsterm nicht widersprüchlich ist

✗ Berechnung von DFN benötigt exponentiell viel Zeit/Platz
($O(2^m)$ der Größe m der *ursprünglichen* Formel)

Konjunktive Normalform (KNF)

Definition: Eine *Klausel* ist eine Menge von Literalen $\{l_1, \ldots, l_n\}$

- Bedeutung: $l_1 \vee \cdots \vee l_n$

Konjunktive Normalform (KNF)

Definition: Eine *Klausel* ist eine Menge von Literalen $\{l_1, \ldots, l_n\}$

- Bedeutung: $l_1 \vee \cdots \vee l_n$

Definition: Eine Formel ϕ ist in *konjunktiver Normalform* (KNF), wenn ϕ als Menge von Klauseln gegeben ist, d.h. ϕ ist als Konjunktion ("und") zwischen Disjunktionen von Literalen gegeben ist

Beispiel

- ϕ: $(A \vee B \vee \neg C) \wedge (\neg A \vee \neg B)$

Konjunktive Normalform (KNF)

Definition: Eine *Klausel* ist eine Menge von Literalen $\{l_1, \dots, l_n\}$

- Bedeutung: $l_1 \lor \cdots \lor l_n$

Definition: Eine Formel ϕ ist in *konjunktiver Normalform* (KNF), wenn ϕ als Menge von Klauseln gegeben ist, d.h. ϕ ist als Konjunktion ("und") zwischen Disjunktionen von Literalen gegeben ist

Beispiel

- ϕ: $(A \lor B \lor \neg C) \land (\neg A \lor \neg B)$

Berechnung und Diskussion

✗ Per Distributionsregel aus NNF ist exponentiell teuer (analog zur DNF)

✓ Tseitin-Transformation: mit zusätzlichen Propositionen für Unterformeln in polynomieller Zeit und *linearer* Größe (siehe FSV vom SoSe 2015)

✓ Erfüllbarkeit von Formeln in KNF lässt sich "oft" effizient prüfen ($\rightarrow$ DPLL)

Automatisiertes Beweisen (SAT/SMT)

DPLL: Effizienter Erfüllbarkeitstest für Aussagenlogik

Grundlegender Algorithmus

After *Davis, Putnam, Logemann and Loveland, 1960*

Gegeben: ϕ in konjunktiver Normalform (KNF)
Start mit leerer Belegung $\mathtt{DPLL}(\phi, \varnothing)$

Algorithm $\mathtt{DPLL}(\phi, s)$

> **if** $s \models \phi$ **return** `sat` with s
> **if** $s \models \neg\phi$ **return** `unsat`
>
> **choose** A with $A \notin s$
>
> $s' := \mathtt{DPLL}(\phi, s + \{A \mapsto \mathit{true}\})$
>
> **if** s' is `unsat`
> **then return** $\mathtt{DPLL}(\phi, s + \{A \mapsto \mathit{false}\})$
> **else return** s'

DPLL: Unit-Propagation (UP)

$\phi = (A \vee B) \wedge (\neg B)$

Beobachtung:

- um die Klausel $(\neg B)$ wahr zu machen, muss gelten $B \mapsto \mathit{false}$
- damit vereinfacht sich ϕ zu (A)
- wiederum nur eine Möglichkeit $A \mapsto \mathit{true}$

Eine Klausel mit nur einem Literal nennt man *Unit-Clause*

DPLL: Unit-Propagation (UP)

$\phi = (A \vee B) \wedge (\neg B)$

Beobachtung:

- um die Klausel $(\neg B)$ wahr zu machen, muss gelten $B \mapsto \mathit{false}$
- damit vereinfacht sich ϕ zu (A)
- wiederum nur eine Möglichkeit $A \mapsto \mathit{true}$

Eine Klausel mit nur einem Literal nennt man *Unit-Clause*

Optimierung: Unit-Propagation vor Verzweigung

while there is a clause with a single unassigned literal l
  $s := s + \{l \mapsto \mathit{true}\}$

(Wobei $\{\neg A \mapsto \mathit{true}\}$ als $\{A \mapsto \mathit{false}\}$ verstanden werden muss)

DPLL: Pure Literal Elimination (PLE)

$$\phi = (A \vee B) \wedge (C \vee D) \wedge (A \vee \neg D)$$

Beobachtung:

- A kommt nur positiv vor (kein $\neg A$)
- es schadet nie zu wählen $A \mapsto \mathit{true}$ (analog für negierte Propositionen)
- damit vereinfacht sich ϕ zu $(C \vee D)$
  (weiter lösbar mit Unit-Propagation/Pure-Literal)
- Analog falls nur $\neg A$ vorkommt, aber nicht A

DPLL: Pure Literal Elimination (PLE)

$$\phi = (A \lor B) \land (C \lor D) \land (A \lor \neg D)$$

Beobachtung:

- A kommt nur positiv vor (kein $\neg A$)
- es schadet nie zu wählen $A \mapsto \mathit{true}$ (analog für negierte Propositionen)
- damit vereinfacht sich ϕ zu $(C \lor D)$
  (weiter lösbar mit Unit-Propagation/Pure-Literal)
- Analog falls nur $\neg A$ vorkommt, aber nicht A

Optimierung: Pure Literal Elimination vor Verzweigung

while there is a pure unassigned literal l
  $s := s + \{l \mapsto \mathit{true}\}$

DPLL: Diskussion

Vorverarbeitung der Eingabeformeln

- Transformation nach KNF (Tseitin Transformation)
- Oft auch zunächst Simplifikation, z.B. $A \wedge \texttt{true} \rightsquigarrow A$

KNF ermöglicht u.a. das Erkennen von Unit-Clauses und Pure-Literals

Implementierungsdetails ($\rightarrow$ SAT Vorlesung)

- Finden nicht zugewiesener Literale ("watched literals")
- Welche Variablen zuerst zuweisen?
  Heuristiken! Hat *enormen* Einfluss auf Performance
- Lernen von Lemmas aus Konflikten ("conflict driven clause learning")
  Resolutionsregel: $(\phi \vee A) \wedge (\psi \vee \neg A) \Longrightarrow \phi \vee \psi$

DPLL: Beispiel (I)

Zeilenweise Notation der Anwendung des Algorithmus auf Papier,
hier syntaktischer Ansatz (kein explizites Hantieren mit Belegungen s)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \land (\neg A \lor B \lor C) \land (B \lor \neg C)$$

Anwendbare Regeln: UP auf A, PLE auf B

DPLL: Beispiel (I)

Zeilenweise Notation der Anwendung des Algorithmus auf Papier,
hier syntaktischer Ansatz (kein explizites Hantieren mit Belegungen s)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \land (\neg A \lor B \lor C) \land (B \lor \neg C)$$

Anwendbare Regeln: UP auf A, PLE auf B

Wähle $A = \texttt{true}$ (UP) und setze ein

$$(\texttt{true}) \land (\texttt{false} \lor B \lor C) \land (B \lor \neg C)$$

und vereinfache

$$(B \lor C) \land (B \lor \neg C)$$

DPLL: Beispiel (I)

Zeilenweise Notation der Anwendung des Algorithmus auf Papier, hier syntaktischer Ansatz (kein explizites Hantieren mit Belegungen s)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \wedge (\neg A \vee B \vee C) \wedge (B \vee \neg C)$$

Anwendbare Regeln: UP auf A, PLE auf B

Wähle $A = \texttt{true}$ (UP) und setze ein

$$(\texttt{true}) \wedge (\texttt{false} \vee B \vee C) \wedge (B \vee \neg C)$$

und vereinfache

$$(B \vee C) \wedge (B \vee \neg C)$$

Wähle $B = \texttt{true}$ (PLE) und setze ein

$$(\texttt{true} \vee C) \wedge (\texttt{true} \vee \neg C)$$

und vereinfache

$$\texttt{true}$$

DPLL: Beispiel (I)

Zeilenweise Notation der Anwendung des Algorithmus auf Papier, hier syntaktischer Ansatz (kein explizites Hantieren mit Belegungen s)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \land (\neg A \lor B \lor C) \land (B \lor \neg C)$$

Anwendbare Regeln: UP auf A, PLE auf B

Wähle $A = \texttt{true}$ (UP) und setze ein

$$(\texttt{true}) \land (\texttt{false} \lor B \lor C) \land (B \lor \neg C)$$

und vereinfache

$$(B \lor C) \land (B \lor \neg C)$$

Wähle $B = \texttt{true}$ (PLE) und setze ein

$$(\texttt{true} \lor C) \land (\texttt{true} \lor \neg C)$$

und vereinfache

$$\texttt{true}$$

Ergebniss: Erfüllbar mit $s(A) = \mathit{true}$, $s(B) = \mathit{true}$, $s(C)$ beliebig

DPLL: Beispiel (II)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \land (\neg A \lor B) \land (\neg A \lor \neg B)$$

DPLL: Beispiel (II)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \land (\neg A \lor B) \land (\neg A \lor \neg B)$$

Wähle $A = \texttt{true}$ (UP) und setze ein

$$(\texttt{true}) \land (\texttt{false} \lor B) \land (\texttt{false} \lor \neg B)$$

und vereinfache

$$(\neg B) \land (B)$$

DPLL: Beispiel (II)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \land (\neg A \lor B) \land (\neg A \lor \neg B)$$

Wähle $A = \texttt{true}$ (UP) und setze ein

$$(\texttt{true}) \land (\texttt{false} \lor B) \land (\texttt{false} \lor \neg B)$$

und vereinfache

$$(\neg B) \land (B)$$

Wähle $B = \texttt{false}$ (UP erste Klausel) und setze ein

$$(\texttt{true}) \land (\texttt{false})$$

und vereinfache

$$\texttt{false}$$

DPLL: Beispiel (II)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A) \wedge (\neg A \vee B) \wedge (\neg A \vee \neg B)$$

Wähle $A =$ `true` (UP) und setze ein

$$(\texttt{true}) \wedge (\texttt{false} \vee B) \wedge (\texttt{false} \vee \neg B)$$

und vereinfache

$$(\neg B) \wedge (B)$$

Wähle $B =$ `false` (UP erste Klausel) und setze ein

$$(\texttt{true}) \wedge (\texttt{false})$$

und vereinfache

`false`

Es gab zuvor keine echte Wahlmöglichkeit, also
Ergebniss: Nicht erfüllbar

DPLL: Beispiel (III)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A \vee \neg B) \wedge (\neg A \vee C) \wedge (B \vee \neg C) \wedge (\neg A \vee \neg B)$$

DPLL: Beispiel (III)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A \vee \neg B) \wedge (\neg A \vee C) \wedge (B \vee \neg C) \wedge (\neg A \vee \neg B)$$

UP/PLE nicht anwendbar, also Verzweigen

- $A = \texttt{true}$ $\qquad (C) \wedge (B \vee \neg C) \wedge (\neg B)$

DPLL: Beispiel (III)

Ziel: Erfüllende Belegungen mit DPPL für

$(A \vee \neg B) \wedge (\neg A \vee C) \wedge (B \vee \neg C) \wedge (\neg A \vee \neg B)$

UP/PLE nicht anwendbar, also Verzweigen

- $A = \texttt{true}$ $\quad (C) \wedge (B \vee \neg C) \wedge (\neg B)$
  - $C = \texttt{true}$ (UP) $\quad (B) \wedge (\neg B)$

DPLL: Beispiel (III)

Ziel: Erfüllende Belegungen mit DPPL für

$(A \lor \neg B) \land (\neg A \lor C) \land (B \lor \neg C) \land (\neg A \lor \neg B)$

UP/PLE nicht anwendbar, also Verzweigen

- $A =$ `true` $\quad (C) \land (B \lor \neg C) \land (\neg B)$
  - $C =$ `true` (UP) $\quad (B) \land (\neg B)$
  - $B =$ `true` (UP) $\quad$ `false`

DPLL: Beispiel (III)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A \vee \neg B) \wedge (\neg A \vee C) \wedge (B \vee \neg C) \wedge (\neg A \vee \neg B)$$

UP/PLE nicht anwendbar, also Verzweigen

- $A =$ `true` — $(C) \wedge (B \vee \neg C) \wedge (\neg B)$
  - $C =$ `true` (UP) — $(B) \wedge (\neg B)$
  - $B =$ `true` (UP) — `false`
- $A =$ `false` — $(\neg B) \wedge (B \vee \neg C)$

DPLL: Beispiel (III)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A \lor \neg B) \land (\neg A \lor C) \land (B \lor \neg C) \land (\neg A \lor \neg B)$$

UP/PLE nicht anwendbar, also Verzweigen

- $A =$ `true` — $(C) \land (B \lor \neg C) \land (\neg B)$
  - $C =$ `true` (UP) — $(B) \land (\neg B)$
  - $B =$ `true` (UP) — `false`
- $A =$ `false` — $(\neg B) \land (B \lor \neg C)$
  - $B =$ `false` (UP) — $(\neg C)$

DPLL: Beispiel (III)

Ziel: Erfüllende Belegungen mit DPPL für

$$(A \lor \neg B) \land (\neg A \lor C) \land (B \lor \neg C) \land (\neg A \lor \neg B)$$

UP/PLE nicht anwendbar, also Verzweigen

- $A =$ `true` — $(C) \land (B \lor \neg C) \land (\neg B)$
  - $C =$ `true` (UP) — $(B) \land (\neg B)$
  - $B =$ `true` (UP) — `false`
- $A =$ `false` — $(\neg B) \land (B \lor \neg C)$
  - $B =$ `false` (UP) — $(\neg C)$
  - $C =$ `false` (UP) — `true`

Ergebniss: Erfüllbar (alle Propositionen mit *false* belegt)

Mögliche Heuristik: wähle Literale so, dass *viele* Klauseln wahr werden

Was Sie können und wissen sollten

- DPLL Algorithmus auf Papier durchführen (analog zu den Beispielen)
- Wie helfen die Regeln "Unit-Propagation" und "Pure Literal Eliminiation" schnell zu einem Ergebnis zu kommen?
  (= Ausnutzen von Abkürzungen in der Suche nach erfüllenden Belegungen)
- Zum Nachdenken: Laufzeitkomplexität von DPLL in der Anzahl der Literale?
- Ausblick: Vorlesung SAT-Solving im WS von Jan Johannsen

License