SAT-Solving und Anwendungen
CDCL SAT-Solving

Prof. Dr. Wolfgang Küchlin
Dr. Eray Gençay
Rouven Walter, M.Sc.

Universität Tübingen

2. November 2017

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Ausblick der letzten Vorlesung

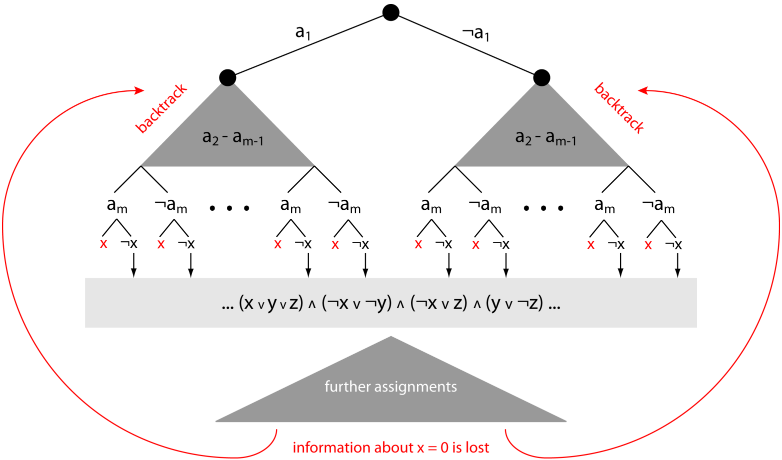
Drei große Probleme bei DPLL:

- Rekursiv, hohe Wahrscheinlichkeit für „Out Of Memory“
- Vergessen von zusätzlichen Informationen beim Backtracking (Illustration auf der nächsten Folie)
  - Beobachtung 1a: Springt man über eine gewisse Grenze beim Backtracking zurück, so „vergisst“ man unterhalb der Grenze erarbeitete lokale Information.
  - Beobachtung 1b: Information zur zukünftigen Vermeidung bereits getroffener Nullstellen lässt sich als Klauseln speichern.
  - Idee 1: Hinzufügen dieser zusätzlichen Information zur Originalformel, so dass sie beim Backtracking nicht verloren geht
- Backtracking immer nur zur letzten durch Entscheidung gesetzten Variable
  - Beobachtung 2: Die letzte Variable ist oft nicht verantwortlich für den aktuellen Konflikt, sondern dieser wurde schon durch frühere Entscheidungen verursacht.
  - Idee 2: Lokalisierung der ursprünglich für den Konflikt verantwortlichen Variable. Backtracking auch über mehrere Entscheidungsebenen hinweg zu dieser Variable weiter oben im Entscheidungsbaum

Resultat: Iterativer SAT-Solver mit **Klausellernen** (Idee 1) und **nicht-chronologischem Backtracking** (Idee 2).

Vergessen von Informationen beim Backtracking

Die Folgerung, dass $x = 0$ gelten muss, hängt nur von wenigen Klauseln in x, y und z ab. Sie wird beim Backtracking durch alle möglichen Belegungen von $a_1, \ldots, a_m$ immer wieder neu berechnet.



Wie realisieren wir diese Ideen?

Idee 1: Lernen von Informationen

Im Konfliktfall (leere Klausel, Konfliktklausel) wird der Konflikt analysiert:

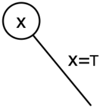
1. Speichere zu jeder Variable, ob sie durch Entscheidung oder UP belegt wurde
2. Bei UP speichere die Klausel, die die UP veranlasst hat *(Reason)*
3. Betreibe Resolution zwischen der Konfliktklausel und den einzelnen Reasons um neue Klauseln zu erhalten
4. Ziel: Ersetzen der durch UP belegten Variablen durch ihre Gründe (belegte Variablen) auf früheren decision levels, die für die UP verantwortlich waren
5. Füge die „beste“ dieser neuen Klauseln der originalen Klauselmenge hinzu

Idee 2: Nicht-chronologisches Backtracking

1. Speichere zu jeder Variablenbelegung ein *Decision Level*
2. Berechne das neue Backtracking Level aus den Decision Levels, die in der neu gelernten Klausel vorkommen. (Backtracking Level ist das numerisch größte Level, das nach Rücknahme der Variablenbelegungen übrig bleibt und auf dem die Berechnung noch vor der nächsten decision fortfährt.)

Aufbau der Datenstruktur

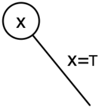
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Level	Variable	Wert	Grund
1	x	T	decision

Aufbau der Datenstruktur

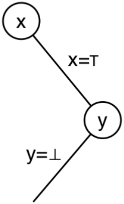
$\{\neg u, w\}$
$\{\neg u, \neg m, h, \neg w\}$
$\{\neg u, \neg f, m, \neg w\}$
$\{\neg f, \neg g, \neg h\}$
$\{y, f\}$
$\{\neg f, g\}$
.
.
.
$\{\neg x, a\}$
$\{\neg x, b, \neg a\}$
$\{\neg x, \neg a, \neg b, c\}$
$\{\neg a, \neg d\}$
$\{\neg b, d, \neg e\}$
.
.
.



Level	Variable	Wert	Grund
1	x	T	decision
	a	T	$\{\neg x, a\}$
	b	T	$\{\neg x, b, \neg a\}$
	c	T	$\{\neg x, \neg a, \neg b, c\}$
	d	$\perp$	$\{\neg a, \neg d\}$
	e	$\perp$	$\{\neg b, d, \neg e\}$

Aufbau der Datenstruktur

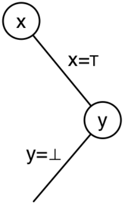
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision

Aufbau der Datenstruktur

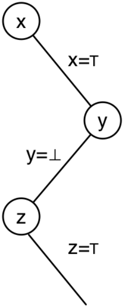
$\{\neg u, w\}$
$\{\neg u, \neg m, h, \neg w\}$
$\{\neg u, \neg f, m, \neg w\}$
$\{\neg f, \neg g, \neg h\}$
$\{y, f\}$
$\{\neg f, g\}$
.
.
.
$\{\neg x, a\}$
$\{\neg x, b, \neg a\}$
$\{\neg x, \neg a, \neg b, c\}$
$\{\neg a, \neg d\}$
$\{\neg b, d, \neg e\}$
.
.
.



Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	$\{\neg x, a\}$
	b	$\top$	$\{\neg x, b, \neg a\}$
	c	$\top$	$\{\neg x, \neg a, \neg b, c\}$
	d	$\bot$	$\{\neg a, \neg d\}$
	e	$\bot$	$\{\neg b, d, \neg e\}$
2	y	$\bot$	decision
	f	$\top$	$\{y, f\}$
	g	$\top$	$\{\neg f, g\}$
	h	$\bot$	$\{\neg f, \neg g, \neg h\}$

Aufbau der Datenstruktur

{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.

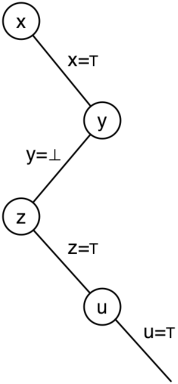


Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision

z kommt in den gegebenen Klauseln nicht vor, kann jedoch in ... vorkommen.

Aufbau der Datenstruktur

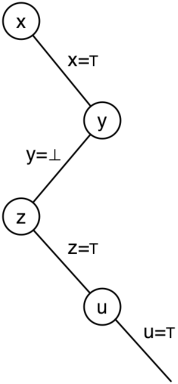
$\{\neg u, w\}$
$\{\neg u, \neg m, h, \neg w\}$
$\{\neg u, \neg f, m, \neg w\}$
$\{\neg f, \neg g, \neg h\}$
$\{y, f\}$
$\{\neg f, g\}$
.
.
.
$\{\neg x, a\}$
$\{\neg x, b, \neg a\}$
$\{\neg x, \neg a, \neg b, c\}$
$\{\neg a, \neg d\}$
$\{\neg b, d, \neg e\}$
.
.
.



Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	$\{\neg x, a\}$
	b	$\top$	$\{\neg x, b, \neg a\}$
	c	$\top$	$\{\neg x, \neg a, \neg b, c\}$
	d	$\bot$	$\{\neg a, \neg d\}$
	e	$\bot$	$\{\neg b, d, \neg e\}$
2	y	$\bot$	decision
	f	$\top$	$\{y, f\}$
	g	$\top$	$\{\neg f, g\}$
	h	$\bot$	$\{\neg f, \neg g, \neg h\}$
3	z	$\top$	decision
4	u	$\top$	decision

Aufbau der Datenstruktur

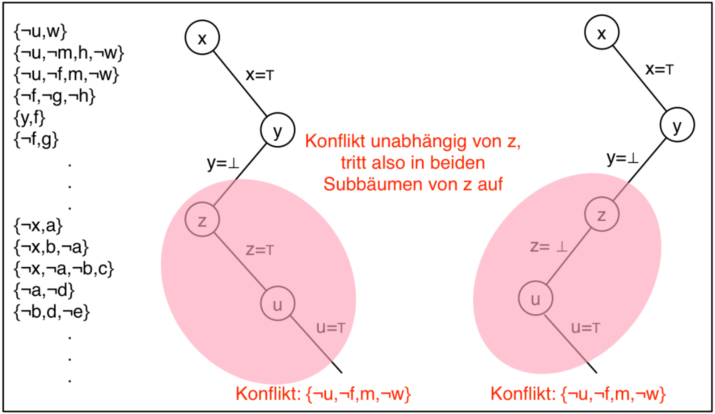
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Konflikt: {¬u,¬f,m,¬w}

Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}

Warum Lernen?



Problem: Bei Backtracking über u hinaus gehen die Informationen über u verloren.
Genauer: wenn auch $u = 0$ zu einem Konflikt führt, wird $z = 0$ probiert. Dieses führt wieder zu dem gleichen Konflikt mit $u = 1$.
Lösung: Hinzufügen einer zusätzlichen Klausel um diese Information zu "lernen"

Logische Analyse der Konfliktsituation

An einem Konflikt sind immer zwei Unit-Klauseln beteiligt (wäre eine davon nicht unit, so wäre die Unit-Propagation durch die andere konfliktlos). Diese sind als Folge einer Entscheidung entstanden (oder waren von Beginn an vorhanden), denn zum Zeitpunkt einer Entscheidung selbst gibt es keine Unit-Klauseln.

Bei den Unit-Klauseln handelt es sich um:

- Eine **Reason Clause** $R = R_r \cup \{\lambda\}$. Die Literale im Rest R_r sind mit der gegenwärtigen Belegung sämtlich $= 0$ und deshalb muss die Variable in λ jetzt (durch Unit Propagation) zwingend so belegt werden, dass $\lambda = 1$. Diese Klausel ist also der Grund für $\lambda = 1$.
- Die **Conflict Clause** $K = K_r \cup \{\neg\lambda\}$. Die Literale im Rest K_r sind mit der gegenwärtigen Belegung sämtlich $= 0$. Durch die Unit-Propagation $\lambda = 1$ der Reason Clause wird die Konfliktklausel nun zu einer Empty Clause, d.h. die Klausel wird unter dieser Variablenbelegung insgesamt $= 0$. Die Konfliktklausel kann man alternativ als Reason Clause für $\lambda = 0$ ansehen, womit dann umgekehrt R zu einer Konfliktklausel wird.

K und R enthalten also zwingend die komplementären Literale λ und $\neg\lambda$ und haben deshalb eine Resolvente $N = K_r \cup R_r$ mit $N = 0$ unter der gegenwärtigen Belegung. (N ist nicht tautologisch. K_r und R_r können keine komplementären Literale enthalten, da sonst eines der beiden $= 1$ wäre.)

Welche Klausel wird hinzugefügt (gelernt)?

Ziel ist:

- eine möglichst kurze Klausel ohne überflüssige Literale. (Je kürzer die Klausel desto genereller und mächtiger das Wissen bzw. der Constraint. Denn für jede Klausel C und Literal λ gilt: $C \models C \cup \{\lambda\}$.)
- Klausel soll nach Backtracking unit sein (damit wird das neue Wissen sofort eingesetzt und erspart eine weitere Entscheidung)

Zwei verschiedene Sichtweisen auf das Lernen:

- Resolution
  - Die aus dem Konflikt erzeugte Resolvente ist eine logische Konsequenz der Klauselmenge
  - Die Resolvente kann als neue Klausel hinzugefügt werden.
  - Falls die Resolvente erfüllt ist, wird der Konflikt vermieden, denn entweder Konfliktklausel oder Reason Klausel sind dann erfüllt.
- Implikationsgraph
  - Knoten sind Variablenbelegungen (x bedeutet, $x \mapsto 1$, $\neg x$ bedeutet $x \mapsto 0$)
  - Kante von A nach B bedeutet, dass A ein Grund war, dass B belegt wurde
  - Decision Variables haben keine eingehenden Kanten
  - War der Grund für die Implikation einer Variable y eine Klausel $\{y, x_1, \ldots, x_n\}$, so hat der Knoten (y) n eingehende Kanten

Aufbau des Implikationsgraphen

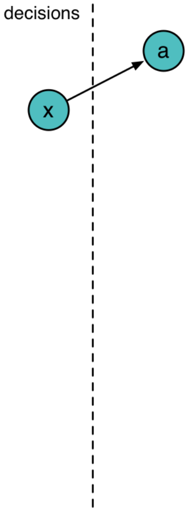
decisions

Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	$\{\neg x, a\}$
	b	$\top$	$\{\neg x, b, \neg a\}$
	c	$\top$	$\{\neg x, \neg a, \neg b, c\}$
	d	$\bot$	$\{\neg a, \neg d\}$
	e	$\bot$	$\{\neg b, d, \neg e\}$
2	y	$\bot$	decision
	f	$\top$	$\{y, f\}$
	g	$\top$	$\{\neg f, g\}$
	h	$\bot$	$\{\neg f, \neg g, \neg h\}$
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	$\{\neg u, w\}$
	m	$\bot$	$\{\neg u, \neg m, h, \neg w\}$
	m	$\top$	$\{\neg u, \neg f, m, \neg w\}$

Aufbau des Implikationsgraphen

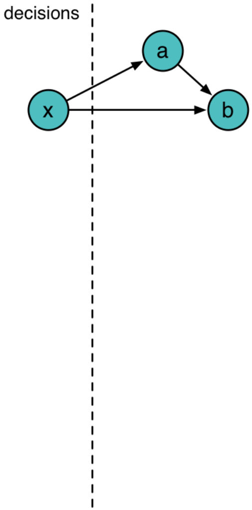
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	$\{\neg x, a\}$
	b	$\top$	$\{\neg x, b, \neg a\}$
	c	$\top$	$\{\neg x, \neg a, \neg b, c\}$
	d	$\bot$	$\{\neg a, \neg d\}$
	e	$\bot$	$\{\neg b, d, \neg e\}$
2	y	$\bot$	decision
	f	$\top$	$\{y, f\}$
	g	$\top$	$\{\neg f, g\}$
	h	$\bot$	$\{\neg f, \neg g, \neg h\}$
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	$\{\neg u, w\}$
	m	$\bot$	$\{\neg u, \neg m, h, \neg w\}$
	m	$\top$	$\{\neg u, \neg f, m, \neg w\}$

Aufbau des Implikationsgraphen



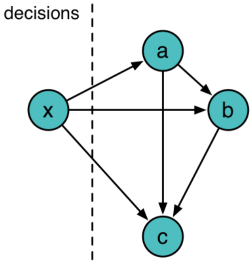
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	$\{\neg x, a\}$
	b	$\top$	$\{\neg x, b, \neg a\}$
	c	$\top$	$\{\neg x, \neg a, \neg b, c\}$
	d	$\bot$	$\{\neg a, \neg d\}$
	e	$\bot$	$\{\neg b, d, \neg e\}$
2	y	$\bot$	decision
	f	$\top$	$\{y, f\}$
	g	$\top$	$\{\neg f, g\}$
	h	$\bot$	$\{\neg f, \neg g, \neg h\}$
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	$\{\neg u, w\}$
	m	$\bot$	$\{\neg u, \neg m, h, \neg w\}$
	m	$\top$	$\{\neg u, \neg f, m, \neg w\}$

Aufbau des Implikationsgraphen



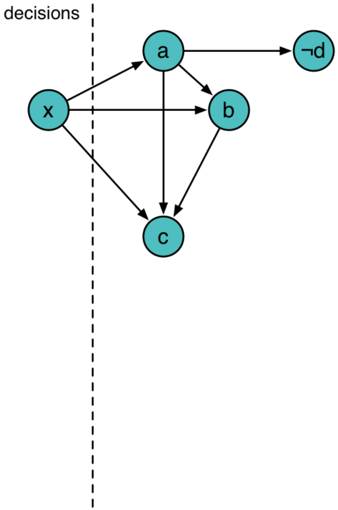
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



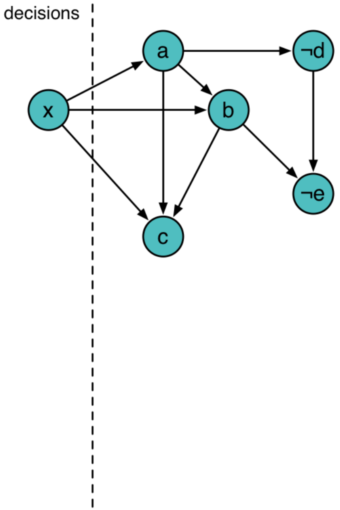
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



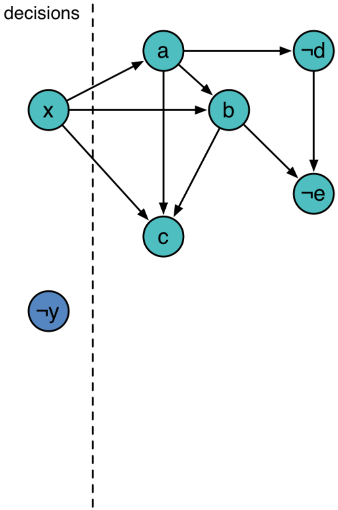
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



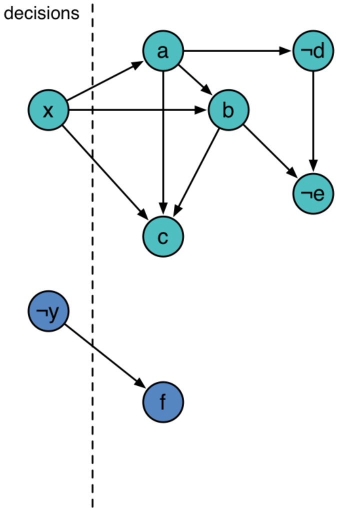
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



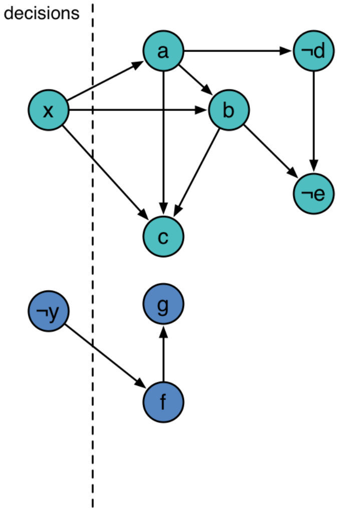
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\bot$	{¬u,¬m,h,¬w}
	m	$\top$	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



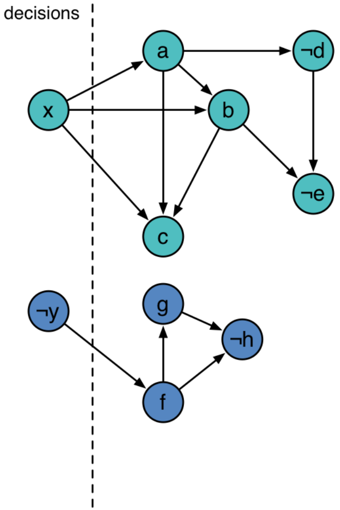
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



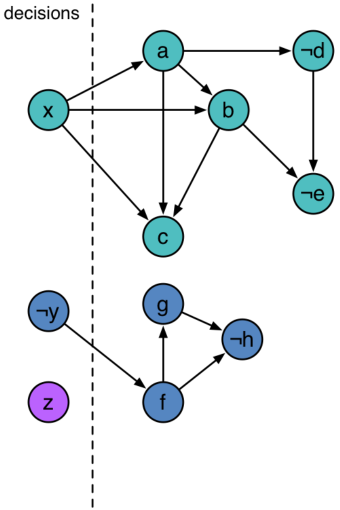
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



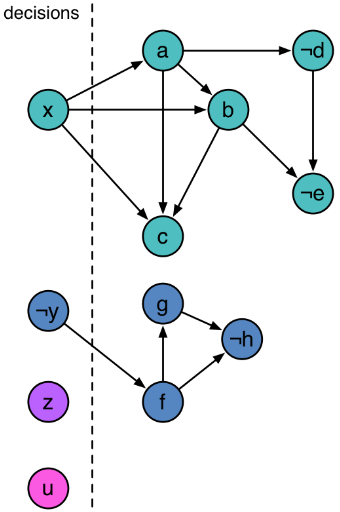
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\bot$	{¬u,¬m,h,¬w}
	m	$\top$	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



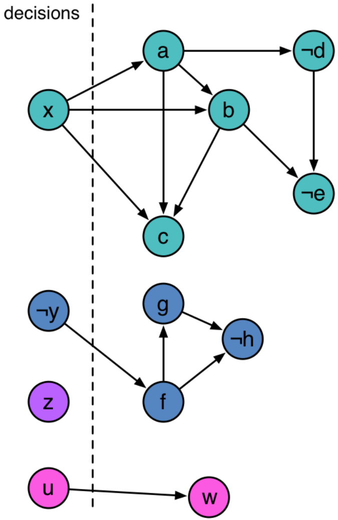
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



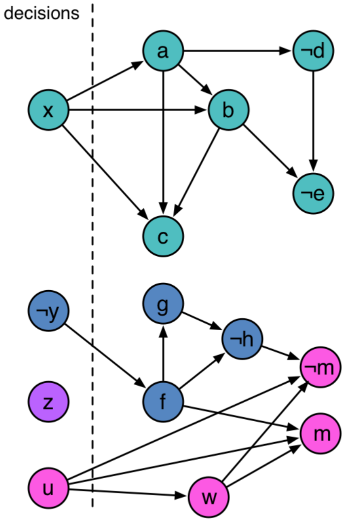
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\bot$	{¬u,¬m,h,¬w}
	m	$\top$	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



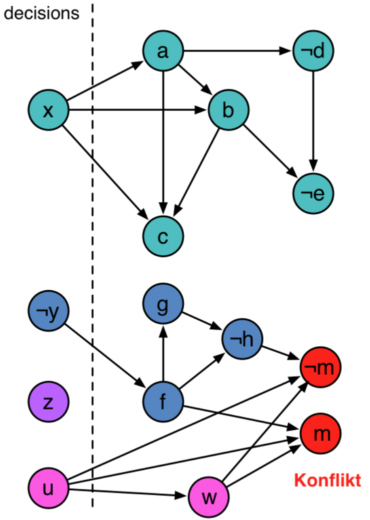
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



Level	Variable	Wert	Grund
1	x	T	decision
	a	T	{¬x,a}
	b	T	{¬x,b,¬a}
	c	T	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	T	{y,f}
	g	T	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	T	decision
4	u	T	decision
	w	T	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	T	{¬u,¬f,m,¬w}

Und welche Klausel lernen wir jetzt?

Erste Idee: Lernen von Entscheidungsvariablen

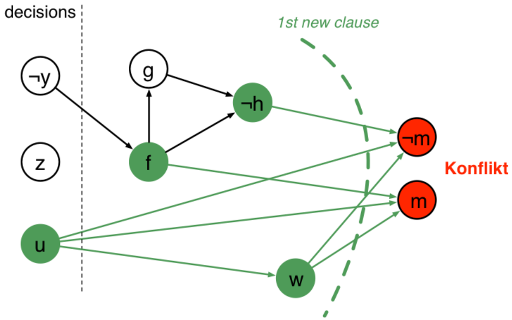
- Ursache für den Konflikt ist eine ungünstige Belegung der Entscheidungsvariablen. Diese darf so nicht vorkommen.
- Neue Klausel: $\neg(x \wedge \neg y \wedge z \wedge u) = (\neg x \vee y \vee \neg z \vee \neg u)$
- Aber: Bei genauerer Betrachtung spielt die Belegung von x und z für den Konflikt gar keine Rolle. Man kann also die obige Klausel mit $x \mapsto 0$ oder mit $z \mapsto 0$ erfüllen ohne damit den Konflikt zu beseitigen.
- Bei Hunderten von Variablen ist das häufig so.

Neue Idee: gezielte Konflikt-Analyse

- Ziel ist eine gelernte Klausel durch deren Erfüllung der Konflikt sicher vermieden wird.
- Ausgehend vom Konflikt, analysiere seine Ursachen genauer ...

Und welche Klausel lernen wir jetzt?

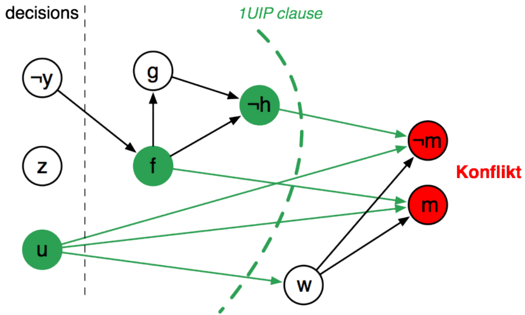
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Deren Belegungen sind für den Konflikt verantwortlich.



Neue Klausel: $\neg(u \wedge f \wedge \neg h \wedge w) = (\neg u \vee \neg f \vee h \vee \neg w)$

Und welche Klausel lernen wir jetzt?

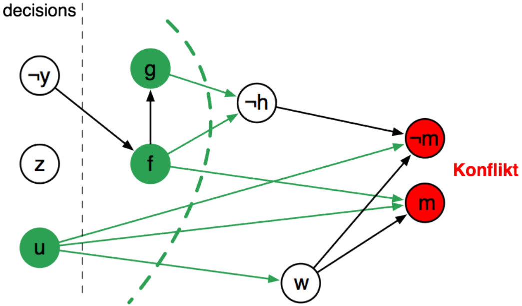
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Deren Belegungen sind für den Konflikt verantwortlich.



Neue Klausel: $\neg(u \wedge f \wedge \neg h) = (\neg u \vee \neg f \vee h)$

Und welche Klausel lernen wir jetzt?

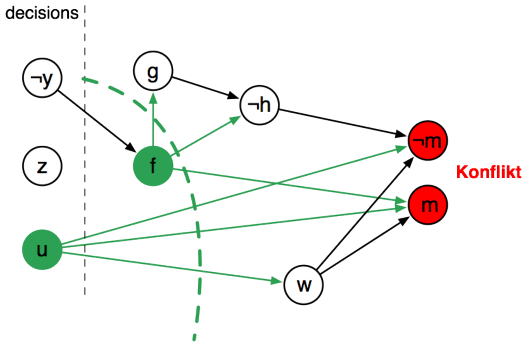
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Deren Belegungen sind für den Konflikt verantwortlich.



Neue Klausel: $\neg(u \wedge f \wedge g) = (\neg u \vee \neg f \vee \neg g)$

Und welche Klausel lernen wir jetzt?

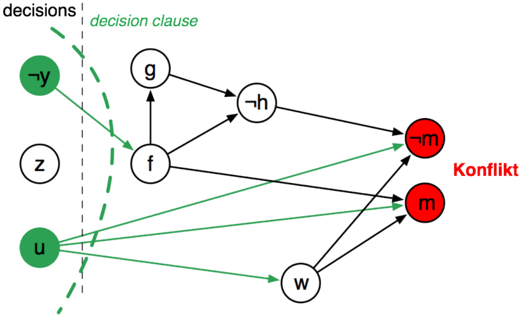
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Deren Belegungen sind für den Konflikt verantwortlich.



Neue Klausel: $\neg(u \wedge f) = (\neg u \vee \neg f)$

Und welche Klausel lernen wir jetzt?

- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Deren Belegungen sind für den Konflikt verantwortlich.



Neue Klausel: $\neg(u \wedge \neg y) = (\neg u \vee y)$

Wiederholung: Resolution — 1

Das Mitführen des Implikationsgraphen ist in der Praxis zu teuer. Die zu den Schnitten gehörigen Klauseln werden wir stattdessen ausgehend von der Konfliktklausel durch Resolutionsschritte berechnen.

Definition (Resolution)

Kommt ein Literal λ positiv in einer Klausel c_1 und negativ in einer Klausel c_2 vor, so kann man die Resolvente

$$r = c_1 \setminus \{\lambda\} \cup c_2 \setminus \{\neg\lambda\}$$

berechnen.

Beispiel (Resolution)

- $c_1 = \{a, \neg b, c, \neg d\}$
- $c_2 = \{\neg a, e, \neg f\}$

Resolvente: $r = \{\neg b, c, \neg d, e, \neg f\}$

Bemerkung: Es darf immer nur über **genau ein Literal** *resolviert werden.*

Wiederholung: Resolution — 2

Bedeutung der Resolvente

Für eine Klauselmenge C und eine Resolvente r zweier Klauseln $c_1, c_2 \in C$ gilt

$$C \equiv C \cup \{r\}.$$

Beweis (kurz): Da r eine Resolvente ist, gilt $C \models \{r\}$ und daher $C \models C \cup \{r\}$. Die Gegenrichtung ist trivial.
Beweis für $C \models \{r\}$[1] Sei α eine beliebige Belegung für C. Dann ist α auch eine Belegung für $C \cup \{r\}$ (da $var(C) = var(C \cup \{r\})$). Angenommen $\alpha \models C$, dann muss auch für alle Klauseln $c \in C$ gelten, dass $\alpha \models c$.

Die Resolvente r der beiden Klauseln $c_1, c_2 \in C$ hat die Form

$$c_1 \setminus \{\lambda\} \cup c_2 \setminus \{\neg\lambda\}.$$

Wir betrachten zwei Fälle:

1. $\alpha \models \lambda$: Aus $\alpha \models c_2$ und $\alpha \not\models \neg\lambda$ folgt $\alpha \models (c_2 \setminus \{\neg\lambda\})$ und daher $\alpha \models r$
2. $\alpha \not\models \lambda$: Aus $\alpha \models c_1$ folgt $\alpha \models (c_1 \setminus \{\lambda\})$ und daher $\alpha \models r$.

[1] vgl. U. Furbach, *Logic for computer scientists*, `http://bit.ly/cY6dwS` (Th. 7)

Berechnung von Schnittklauseln mit Resolution

Beobachtung

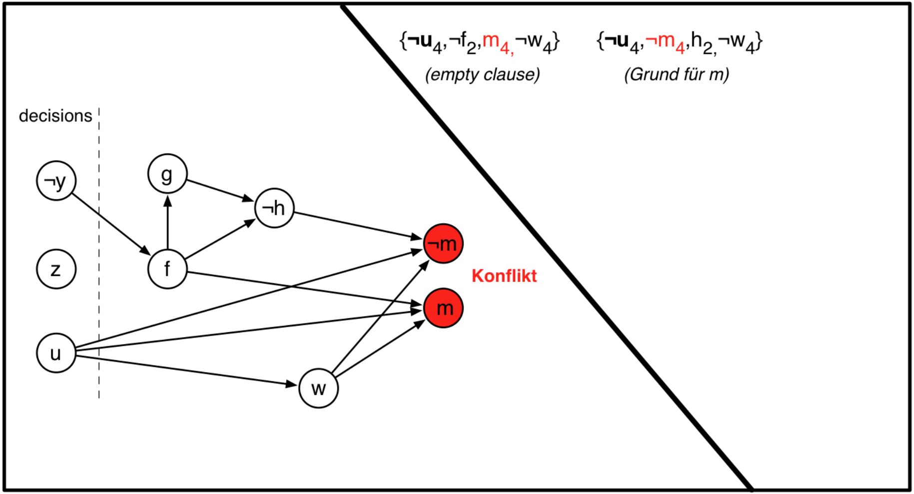
Die Schnittklauseln können ausgehend von der Konfliktklausel durch Resolutionsschritte berechnet werden.

- Erster Schritt: Berechnung der First New Clause.
  - An einem Konflikt sind immer zwei Unit-Klauseln beteiligt:
    - Eine **Reason Clause** $R = R_r \cup \{\lambda\}$. Die Klausel ist unit mit Unit Literal λ. Diese Klausel ist der Grund für $\lambda = 1$.
    - Eine **Conflict Clause** $K = K_r \cup \{\neg\lambda\}$. Die Klausel ist unit mit Unit Literal $\neg\lambda$. Durch die Unit-Propagation $\lambda = 1$ der Reason Clause wird die Konfliktklausel nun zu einer Empty Clause, d.h. die Funktion wird an der Stelle dieser Variablenbelegung insgesamt $= 0$.
  - K und R enthalten also zwingend die komplementären Literale λ und $\neg\lambda$ und haben deshalb eine Resolvente $N = K_r \cup R_r$ mit $N = 0$ unter der gegenwärtigen Belegung. (N ist nicht tautologisch. K_r und R_r können keine komplementären Literale enthalten, da sonst eines der beiden $= 1$ wäre.)

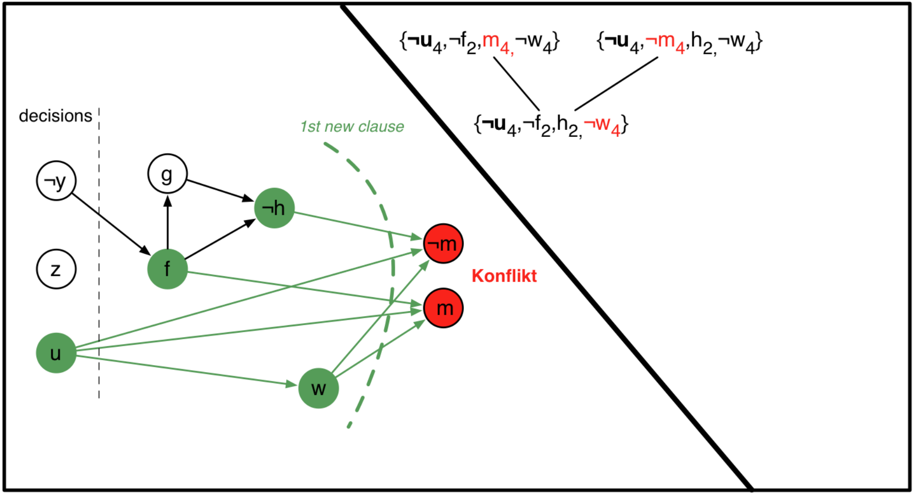
Berechnung von Schnittklauseln mit Resolution

- Folgende Schritte: Berechnung höherer Schnittklauseln.
    - Sei μ irgend ein (früher) durch Unit Propagation mit 0 belegtes Literal in einer Schnittklausel N. Dann gibt es für diese Belegung eine Reason Clause $G = G_r \cup \{\neg\mu\}$. Folglich gibt es eine Resolvente R_1 zwischen N und G. In R_1 ist das μ aus N ersetzt durch die Literale von G_r, deren Belegung einmal der Grund war für die Belegung $\mu = 0$ durch Unit Propagation.
    - Mit jeder solchen Resolution schreitet man im Implikationsgraph rückwärts von rechts nach links und erzeugt sukzessive die Klauseln zu jedem Schnitt. Das Verfahren endet spätestens in einer Decision Clause, in der keine UP Variablen mehr enthalten sind.
    - Jede Schnittklausel ist als Resolvente eine logische Folgerung der ursprünglichen Klauselmenge C und kann deshalb zu C hinzugefügt werden. Sie verhindert, dass die mit dem gegenwärtigen "Konflikt" getroffene Nullstelle jemals wieder getroffen wird, denn sie würde zuvor Unit werden und eine andere Belegung der letzten Variable erzwingen.

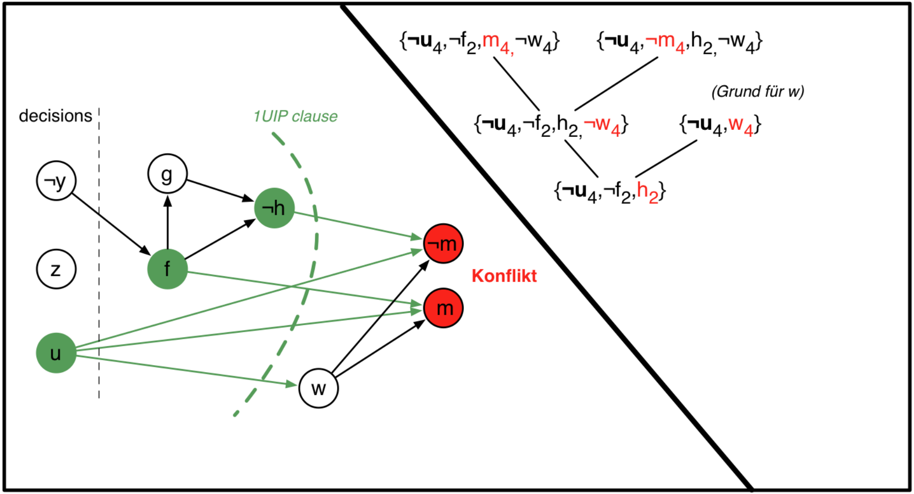
Berechnung von Schnittklauseln mit Resolution



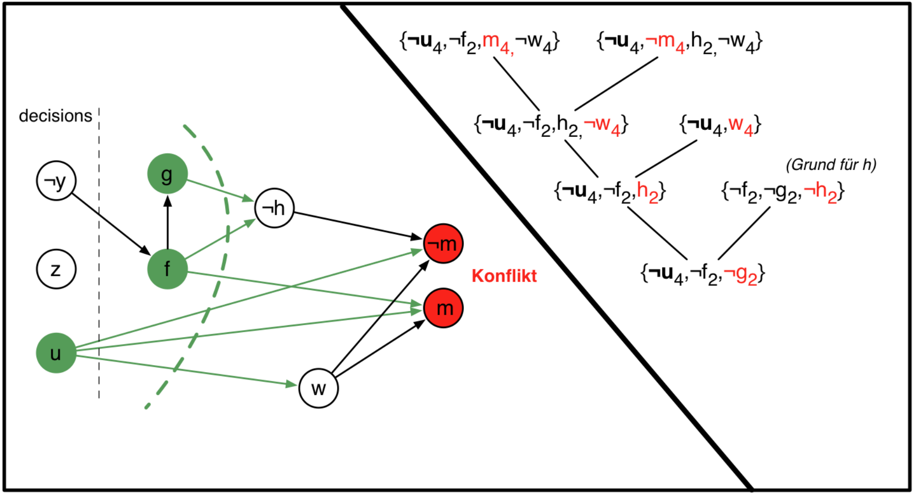
Berechnung von Schnittklauseln mit Resolution



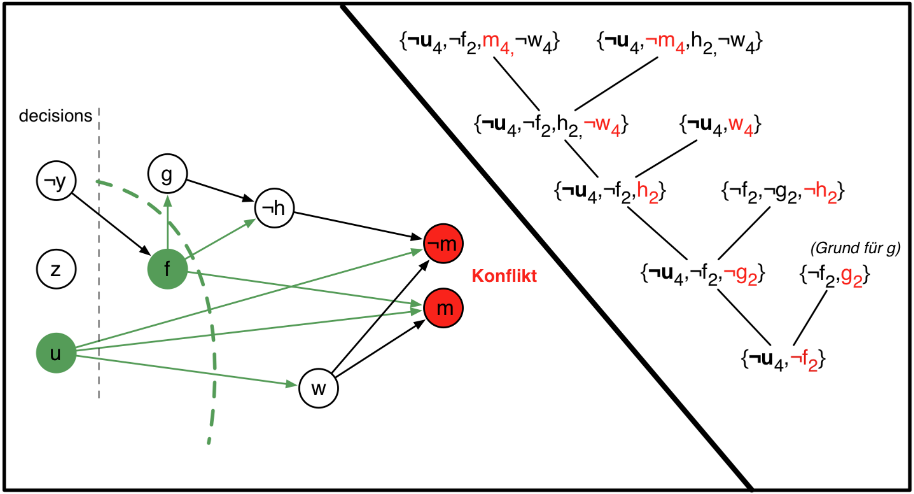
Berechnung von Schnittklauseln mit Resolution



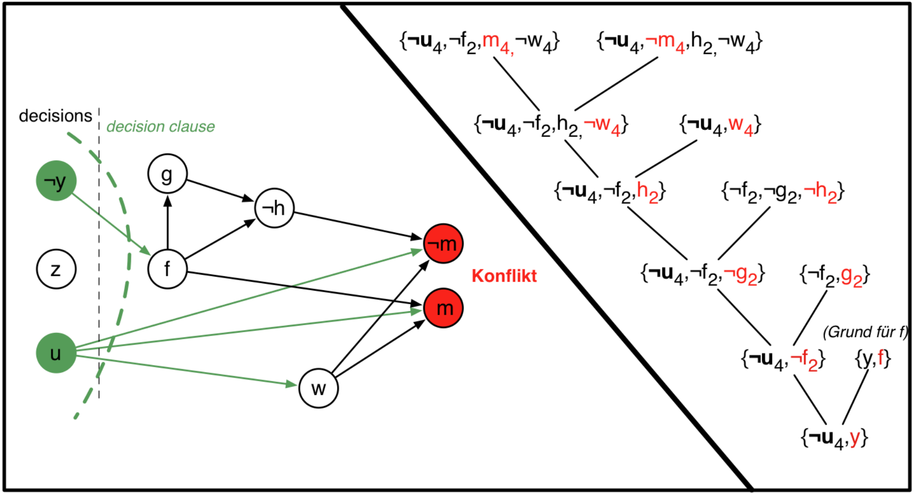
Berechnung von Schnittklauseln mit Resolution



Berechnung von Schnittklauseln mit Resolution



Berechnung von Schnittklauseln mit Resolution



Verschiedene Strategien zum Lernen neuer Klauseln

Abhängig davon, wie lange man Resolution betreibt bzw. wo man den Implikationsgraphen schneidet, ergeben sich verschiedene neue Klauseln:

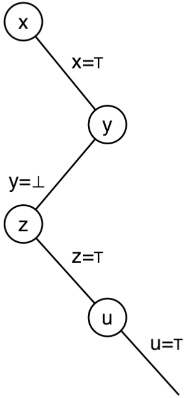
- First New Cut Clause: Die erste neue Klausel, die man durch Resolution erhält. Entspricht dem Cut, der auf der einen Seite nur die beiden Konfliktliterale enthält.
- 1UIP Clause (First Unique Implication Point): In der Klausel ist genau eine Variable (UIP) auf höchstem (größtem) Decision Level ℓ_u. Löscht man die Variablenbelegungen rückwärts bis inkl. Level ℓ_u, wird die Klausel Unit. Man kann mit dem Löschen solange fortfahren wie die Klausel Unit bleibt, also bis zum zweithöchsten Level (Level ℓ_z), der in der Klausel vorkommt. 1UIP ist die erste UIP Klausel, die man ausgehend vom Konflikt erreicht.
- Decision Clause: enthält nur noch Entscheidungsvariablen

Einsetzen der 1UIP Klausel

- Hinzufügen der neuen Klausel zur Klauselmenge
- Backtracking zu einem Level ℓ_b, das (numerisch) kleiner ist als das Level ℓ_u der UIP Variable und mindestens so groß wie das Level ℓ_z, also $\ell_z \leq \ell_b < \ell_u$.
- Unit Propagation. (Die 1UIP Klausel ist nach dem Backtracking immer Unit.)

Auflösung des Beispiels mit der 1UIP Clause

{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.
{¬u,¬f,h} *(Neue Klausel)*

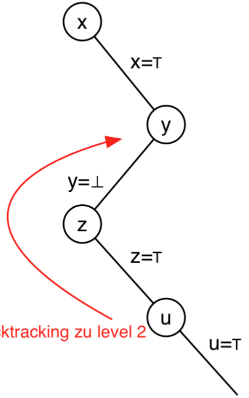


Konflikt: {¬u,¬f,m,¬w}

Level	Variable	Wert	Grund
1	x	T	decision
	a	T	{¬x,a}
	b	T	{¬x,b,¬a}
	c	T	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	T	{y,f}
	g	T	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	T	decision
4	u	T	decision
	w	T	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}

Auflösung des Beispiels mit der 1UIP Clause

{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.
{¬u,¬f,h} *(Neue Klausel)*

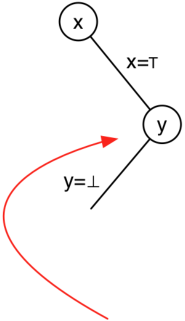


Konflikt: {¬u,¬f,m,¬w}

Level	Variable	Wert	Grund
1	x	T	decision
	a	T	{¬x,a}
	b	T	{¬x,b,¬a}
	c	T	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	T	{y,f}
	g	T	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	T	decision
4	u	T	decision
	w	T	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}

Auflösung des Beispiels mit der 1UIP Clause

$\{\neg u,w\}$
$\{\neg u,\neg m,h,\neg w\}$
$\{\neg u,\neg f,m,\neg w\}$
$\{\neg f,\neg g,\neg h\}$
$\{y,f\}$
$\{\neg f,g\}$
.
.
.
$\{\neg x,a\}$
$\{\neg x,b,\neg a\}$
$\{\neg x,\neg a,\neg b,c\}$
$\{\neg a,\neg d\}$
$\{\neg b,d,\neg e\}$



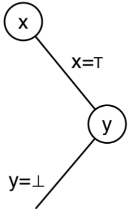
Backtracking zu level 2

.
.
.
$\{\neg u,\neg f,h\}$ *(Neue Klausel)*

Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	$\{\neg x,a\}$
	b	$\top$	$\{\neg x,b,\neg a\}$
	c	$\top$	$\{\neg x,\neg a,\neg b,c\}$
	d	$\bot$	$\{\neg a,\neg d\}$
	e	$\bot$	$\{\neg b,d,\neg e\}$
2	y	$\bot$	decision
	f	$\top$	$\{y,f\}$
	g	$\top$	$\{\neg f,g\}$
	h	$\bot$	$\{\neg f,\neg g,\neg h\}$

Auflösung des Beispiels mit der 1UIP Clause

{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.
{¬u,¬f,h} *(Neue Klausel)*



Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
	u	⊥	**{¬u,¬f,¬h}**

Nach UP von u sind alle gegebenen Klauseln erfüllt

Bemerkenswertes zu CDCL mit 1UIP Lernen

Non-chronological backtracking (back-jumping) Im Beispiel ging das Backtracking von Entscheidungsebene 4 über die Variable z hinweg zur Ebene 2

Änderung der Belegungsreihenfolge Nach dem Backtracking wird an der Stelle, wo vorher z (durch decision) belegt wurde, nun u (durch Unit Propagation) belegt; z käme erst danach an die Reihe.

Konfliktbeseitigung mit 1UIP Klauseln Nach Backtracking erzwingt die 1UIP Clause die geänderte Belegung einer Variablen. Dies kann eine Entscheidungsvariable sein (die nun zu einer UP Variablen wird), aber auch eine UP Variable. Im letzteren Fall wird mittelbar die ehemals zugehörige Entscheidungsvariable zu einer UP Variablen, da der Zwang jetzt umgekehrt wirkt.

Finaler Konflikt – UNSAT Ein Konflikt, der ohne jede Entscheidungen rein durch Unit Propagationen zustande kommt, findet auf Ebene 0 statt. Er kann nicht mehr aufgelöst werden und führt zum Ergebnis UNSAT.

Globaler Ablauf des 1UIP CDCL Algorithmus Als Folge eines jeden Konflikts wird (unmittelbar oder mittelbar) eine Entscheidungsvariable durch eine UP-Variable ersetzt. Dies setzt sich fort, bis entweder ein Finaler Konflikt auftritt oder eine erfüllende Belegung gefunden wurde.

Konfliktbeseitigung mit 1UIP Klauseln

Im Beispiel oben ersehen wir aus dem Implikationsgraphen, dass derselbe Konflikt auch aufgetreten wäre unter der Belegungsreihenfolge $x \to u \to z \to y$. Ausgehend vom Konflikt hätten wir eine ähnliche Resolutionskette erhalten, allerdings wären die Entscheidungsebenen 2 und 4 in den Variablen vertauscht und w_2 würde erst später eliminiert werden.

Wir erhalten nun die folgenden 5 Klauseln (vgl. auch nachfolgende Folie):

1. First New Clause: $\{\neg u_2, \neg f_4, h_4, \neg w_2\}$
2. $\{\neg u_2, \neg f_4, \neg g_4, \neg w_2\}$
3. 1UIP: $\{\neg u_2, \neg f_4, \neg w_2\}$
4. 2UIP: $\{\neg u_2, y_4, \neg w_2\}$
5. 3UIP (decision clause): $\{\neg u_2, y_4\}$

Durch die andere Reihenfolge ist die 1UIP-Clause nun nicht schon wie bisher als zweite Klausel mit $\{\neg u_4, \neg f_2, h_2\}$ erreicht sondern "erst" mit $\{\neg u_2, \neg f_4, \neg w_2\}$.

Nach Backtracking zu Ebene 2 wird nun durch die 1UIP Clause zunächst die alte UP Variable f_4 durch Unit Propagation auf Ebene 2 neu belegt ($f_2 = 0$). Erst in der Folge davon wird durch eine weitere Unit Propagation auf Ebene 2 die frühere Entscheidungsvariable y_4 neu belegt ($y_2 = 1$) und wird so zu einer UP-Variable.

Konfliktbeseitigung mit 1UIP Klauseln

Variablenreihenfolge: $x \to u \to z \to y$. Resolutionsbaum:

$\{\neg u_2, \neg f_4, m_4, \neg w_2\}$ $\{\neg u_2, \neg m_4, h_4, \neg w_2\}$

1st new clause: $\{\neg u_2, \neg f_4, h_4, \neg w_2\}$ $\{\neg f_4, \neg g_4, \neg h_4\}$

$\{\neg u_2, \neg f_4, \neg g_4, \neg w_2\}$ $\{\neg f_4, g_4\}$

1st UIP: $\{\neg u_2, \neg f_4, \neg w_2\}$ $\{y_4, f_4\}$

$\{\neg u_2, y_4, \neg w_2\}$ $\{\neg u_2, w_2\}$

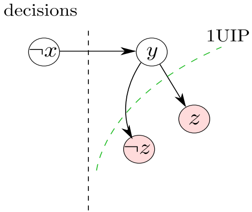
decision clause: $\{\neg u_2, y_4\}$

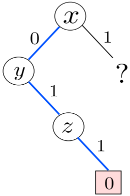
1UIP CDCL Beispiel — Corner Case 1

Änderung der Belegungsreihenfolge.

$$\{\{x, y\}, \{\neg y, z\}, \{\neg y, \neg z\}\}$$

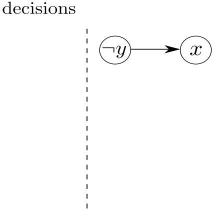
Level	Variable	Wert	Grund
1	X	0	decision
	y	1	$\{x, y\}$
	Z	1	$\{\neg y, z\}$
	Z	0	$\{\neg y, \neg z\}$

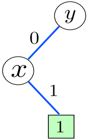




$$\{\{x, y\}, \{\neg y, z\}, \{\neg y, \neg z\}, \{\neg y\}\}$$

Level	Variable	Wert	Grund
0	y	0	$\{\neg y\}$
	X	1	$\{x, y\}$



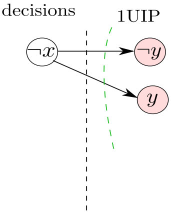


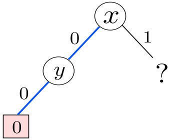
CDCL Beispiel — Corner Case 2

Backtracking auf Ebene 0: Resultat UNSAT.

$$\{\{x, y\}, \{x, \neg y\}, \{\neg x, y\}, \{\neg x, \neg y\}\}$$

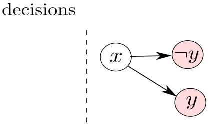
Level	Variable	Wert	Grund
1	x	0	decision
	y	0	$\{x, \neg y\}$
	y	1	$\{x, y\}$

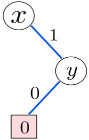




$$\{\{x, y\}, \{x, \neg y\}, \{\neg x, y\}, \{\neg x, \neg y\}, \{x\}\}$$

Level	Variable	Wert	Grund
0	x	1	$\{x\}$
	y	0	$\{\neg x, \neg y\}$
	y	1	$\{\neg x, y\}$





Konflikt auf Level 0: UNSAT

Erklärung von UNSAT durch Resolution

Erklärungen

In der Praxis muss das Ergebnis UNSAT erklärt werden, bzw. es muss ein Beweis dafür präsentiert werden, den der Benutzer unabhängig vom SAT-Solver verstehen kann. Hierfür eignet sich (in erster Näherung) ein Resolutionsbeweis.

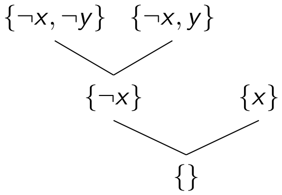
Resolutionsbeweis von UNSAT

- Es existiert ein Konflikt auf Ebene 0.
- Beginnend mit der Konfliktklausel erzeugen wir einen Resolutionsbeweis. Dieser endet mit der leeren Klausel, da es jetzt für alle Belegungen einen Grund gibt, sodass man sukzessive alle Literale aus der Konfliktklausel entfernen kann.
- In einer Erklärung dürfen aber keine gelernte Klauseln auftreten, da dem Nutzer nur Klauseln aus der Ursprungsmenge bekannt sind. Gelernte Klauseln müssen entweder aus dem Beweis entfernt oder ihrerseits erklärt werden.
- Für jede gelernte Klausel gibt es aber wiederum einen Resolutionsbeweis. Diesen muss man sich bei ihrer Herleitung merken und an Stelle der Klausel in dem Beweis von UNSAT einsetzen.

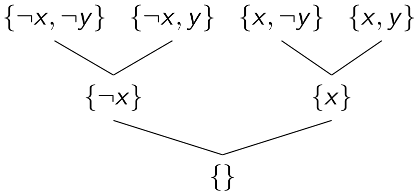
CDCL Beispiel — Erklärung von UNSAT

Klauselmenge: $\{\{x, y\}, \{x, \neg y\}, \{\neg x, y\}, \{\neg x, \neg y\}\}$.

Resolutionsbaum für leere Klausel (Konflikt auf Level 0):



Vollständiger Resolutionsbaum (inklusive Auflösung gelernter Klausel $\{x\}$):



Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

0

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

01

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

011

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

0111

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

01110

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

0111011

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

01110110

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

011101101111

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an

0111011011110**1** Konflikt

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

0111011011110111 Konflikt

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

011101101111011 Konflikt

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

0111011110111110111111 Konflikt

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

0111011110111110111111 Konflikt

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

01110111101111111111111111 Konflikt

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

0111011110111111111111111

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

11011110111011110111 Konflikt

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

... und so weiter bis entweder...

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

1111011110111110111011 sat

Von Entscheidungen zu Erzwingungen durch UP

Zur Veranschaulichung des Fortschritts durch 1UIP Lernen konstruieren wir einen Binärvektor wie folgt:

- Starte mit einem leeren Vektor
- Bei einer Entscheidung hänge hinten eine 0 an
- Bei einer Propagation hänge eine 1 an
- Wenn ein Konflikt auftritt, gehe zurück zu einer 0 und ändere sie zu einer 1

oder

11111111111111111111111 Konflikt: unsat

Der CDCL Algorithmus

Algorithm 1: sat(C)

```
Input: Clauseset C
Output: SAT or UNSAT
level = 0 ;                                   // Setze initiales Level auf 0
α = ∅ ;                                       // Alle Variablen sind unbelegt
while true do
    UP(C, α) ;                                // Unit Propagation
    if C contains an empty clause then
        ec = empty clause;
        level = analyzeConf(ec, C) ;          // Lerne neue Klausel
        if level == −1 then
            return UNSAT
        backtrack(level) ;                    // Backtracking
    else
        if α ⊨ C then
            return SAT
        level = level + 1 ;                   // Erhöhe Level, ...
        choose x ∉ α ;                        // ... wähle neue Variable,
        α = α ∪ [x ↦ 0] ;                     // ... belege Variable
```

Der Konfliktanalyse Algorithmus

Algorithm 2: analyzeConf(ec, C)

```
Input: an empty clause ec, the set of clauses C
Output: the backtracking level
if current decision level == 0 then
    return −1
lv = the last assigned variable before the conflict clause;
reason = the reason of lv;
newclause = resolve(ec,reason) ;                      // Erste Resolution
while the stop criterion for newclause is not met do
    cv = chooseLiteral(newclause);
    reason = the reason of cv;
    newclause = resolve(newclause,reason) ;           // Weitere Resolution
C = C ∪ {newclause} ;                                 // Füge neue Klausel hinzu
level = computeBacktrackLevel(newclause) ;            // Berechne Backtrack Level
return level;
```

- **Stopkriterium bei 1UIP:** Nur noch eine Variable auf höchstem Decision Level l_h
- **Backtracklevel:** $\max\{\text{level} \mid \text{level} < l_h\}$

Korrektheit[2] des CDCL Algorithmus — 1

Theorem (Erfüllbarkeit einer Formel)

Ist die Ausgabe von $\mathrm{sat}(C) = \mathtt{SAT}$*, so ist die Klauselmenge* C *erfüllbar.*

Beweis.

Der Algorithmus gibt nur dann `SAT` zurück, wenn eine erfüllende Belegung α gefunden wurde. α erfüllt sowohl die gelernten Klauseln als auch C. □

Theorem (Unerfüllbarkeit einer Formel)

Ist die Ausgabe von $\mathrm{sat}(C) = \mathtt{UNSAT}$*, so ist* C *eine Kontradiktion.*

Intuition

Auf einer Kontradiktion ersetzt der Algorithmus sukzessive die möglichen Entscheidungen zur Konstruktion einer erfüllenden Belegung durch Unit-Propagationen aufgrund von gelernten Klauseln. Wird UNSAT zurückgegeben, so sind keine Entscheidungen mehr möglich.

[2] Beweise nach: L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In: *Proc. DATE'03*.

Korrektheit des CDCL Algorithmus — 2

Theorem (Unerfüllbarkeit einer Formel)

Ist die Ausgabe von $\mathrm{sat}(C) = \mathtt{UNSAT}$*, so ist* C *eine Kontradiktion.*

Beweis.

Der Algorithmus gibt nur dann `UNSAT` zurück, wenn ein Konflikt auf Decision Level 0 gefunden wurde. Von diesem Konflikt ausgehend kann man nun wie in der `while`-Schleife im Algorithmus *analyzeConf* Resolutionen ausführen. Auf Level 0 gibt es keine Entscheidungsvariable (alle Variablen wurden durch UP belegt). D.h. im Speziellen muss es zu jeder Variable eine Reason geben. Da in jedem Durchlauf der `while`-Schleife eine Variable aus `newclause` durch Resolution entfernt wird und es zu jeder Variable eine Reason gibt, endet man nach spätestens n Schritten (Anzahl der Variablen in der Konfliktklausel) bei der leeren Klausel. D.h. die leere Klausel ist aus der Formel durch Resolution erzeugbar und daher muss die Formel unerfüllbar sein. □

Korrektheit der Ausgabe des Algorithmus: **Partielle Korrektheit!**

Termination des CDCL Algorithmus — 1

Theorem (Termination des Algorithmus)

Der Algorithmus sat(C) *terminiert für jede beliebige Eingabeklauselmenge* C.

Intuition

Der Algorithmus versucht im Grunde alle möglichen Variablenbelegungen bis die erste erfüllende Belegung gefunden wurde oder keine der endlich vielen Alternativen (Entscheidungen) mehr möglich sind. Andererseits trifft er keine Nullstelle ein zweites Mal, da die gelernten Klauseln dies verhindern.

Vorbereitung des Beweises — 1

Sei $k(l)$ die Anzahl der Variablen, die auf Level l belegt wurden. Sei n die Anzahl der Variablen der Originalformel. Auf jedem Decision Level (außer Level 0) muss mindestens eine Variable belegt werden. Folgende Aussagen gelten offensichtlich:

1. $\forall l \; [(0 \leq l \leq n) \rightarrow k(l) \leq n]$
2. $\forall l \; [(l > n) \rightarrow k(l) = 0]$
3. $\sum_{l=0}^{n} k(l) = n$

Termination des CDCL Algorithmus — 2

Vorbereitung des Beweises — 2

Wir betrachten die Funktion

$$f = \sum_{l=0}^{n} \frac{k(l)}{(n+1)^l}.$$

Diese Funktion bildet „eine Art lexikographische Ordnung": Für zwei Variablenbelegungen α und β der selben Formel gilt $f_\alpha > f_\beta$ gdw.

1. ein Decision Level d mit $0 \leq d < n$ existiert, in dem $k_\alpha(d) > k_\beta(d)$ gilt, und
2. für alle Decision Levels l mit $0 \leq l < d$ gilt, dass $k_\alpha(l) = k_\beta(l)$.

Intuitiv gewichtet diese Funktion Variablenbelegungen auf kleinen Decision Levels höher. Die folgende Ungleichung hält:

$$\frac{1}{(n+1)^j} > \frac{n}{(n+1)^{j+1}} \tag{1}$$

D.h. von zwei verschiedene Variablenbelegungen hat diejenige den größeren Wert von f, bei der die Belegungen auf die kleineren Decision Levels konzentriert sind.

Termination des CDCL Algorithmus — 3

Beweis.

Der Wert von f steigt während des Solving Prozesses monoton an. Solange kein Konflikt auftritt ist dies offensichtlich, da in jedem Schritt neue Variablen auf dem höchsten Decision Level belegt werden, ohne dass ältere Variablenbelegungen verändert werden.

Tritt ein Konflikt auf, erfolgt ein Backtracking zu einem kleineren Decision Level und auf diesem Level wird mindestens eine neue Variable belegt. Aus der Ungleichung (1) folgt, dass auch in diesem Fall der Wert von f ansteigt — ungeachtet der Variablenbelegungen, die während des Backtracking Prozesses rückgängig gemacht wurden.

Da f nur eine endliche Anzahl von verschiedenen Werten annehmen kann, muss der Algorithmus also terminieren. □

Partielle Korrektheit + Termination ⇒ Totale Korrektheit