SAT-Solving und Anwendungen

CDCL SAT-Solving

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

Universität Tübingen

30. April 2013

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Ausblick der letzten Vorlesung

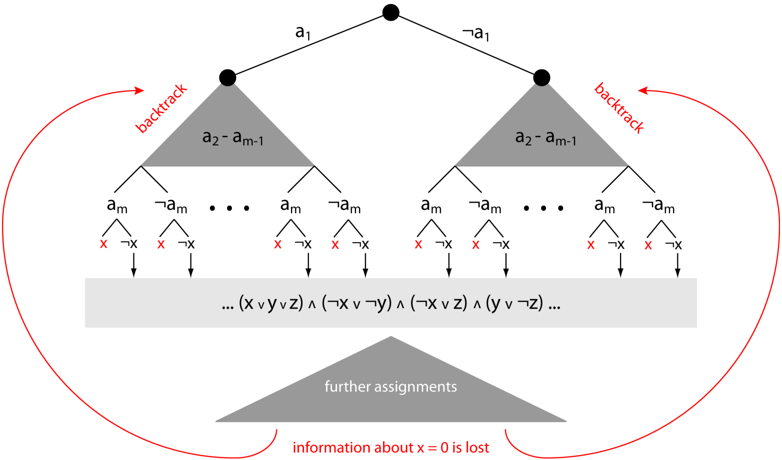
Zwei große Probleme bei DPLL:

- Vergessen von zusätzlichen Informationen beim Backtracking (Illustration auf der nächsten Folie)
  - Beobachtung 1: Springt man über eine gewisse Grenze beim Backtracking zurück, so "vergisst" man bereits erarbeitete Information (z.B. bestimmte UPs)
  - 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
  - Idee 2: Backtracking auch über mehrere Entscheidungen hinweg zu einer Variable weiter oben im Entscheidungsbaum

Resultat: SAT-Solver mit nicht-chronologischem Backtracking (Problem 1) und **Klausellernen** (Problem 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) wird der Konflikt analysiert:

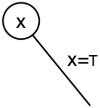
1. Speichere zu jeder Variable, ob sie durch Entscheidung oder UP belegt wurde
2. Bei UP speichere die Klausel, die die UP veranlasst hat *(Reason)*
3. Betreibe Resolution zwischen den einzelnen Reasons um eine neue Klausel zu erhalten
4. Füge diese neue Klausel der originalen Klauselmenge hinzu

Idee 2: Nicht-chronologisches Backtracking

1. Speichere zu jeder Variablenbelegung ein *Decision Level*
2. Berechne das neue Backtracking Level aus der neu gelernten Klausel

Aufbau der Datenstruktur

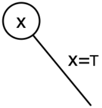
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Level	Variable	Wert	Grund
1	x	T	decision

Aufbau der Datenstruktur

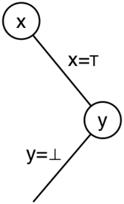
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Level	Variable	Wert	Grund
1	x	T	decision
	a	T	{¬x,a}
	b	T	{¬x,b,¬a}
	c	T	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}

Aufbau der Datenstruktur

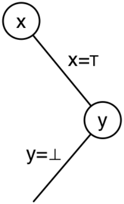
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Level	Variable	Wert	Grund
1	x	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

Aufbau der Datenstruktur

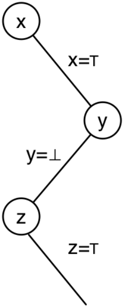
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Level	Variable	Wert	Grund
1	x	⊤	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}

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}
.
.
.

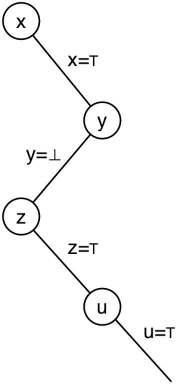


Level	Variable	Wert	Grund
1	x	T	decision
	a	T	{¬x,a}
	b	T	{¬x,b,¬a}
	c	T	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	T	{y,f}
	g	T	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	T	decision

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

Aufbau der Datenstruktur

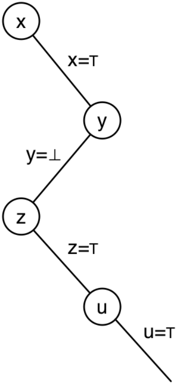
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



Level	Variable	Wert	Grund
1	x	⊤	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

Aufbau der Datenstruktur

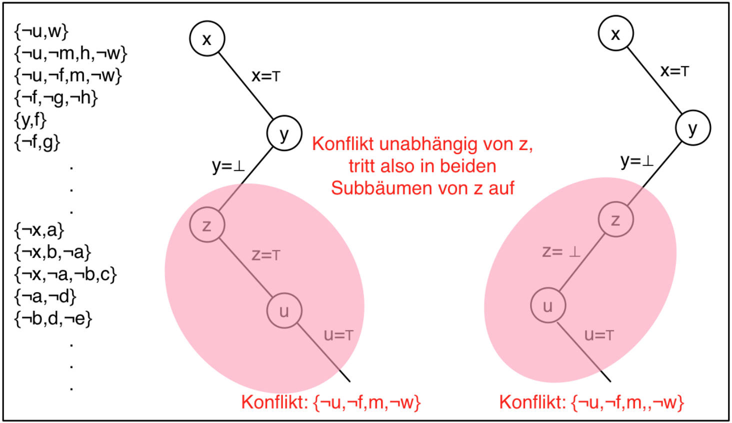
{¬u,w}
{¬u,¬m,h,¬w}
{¬u,¬f,m,¬w}
{¬f,¬g,¬h}
{y,f}
{¬f,g}
.
.
.
{¬x,a}
{¬x,b,¬a}
{¬x,¬a,¬b,c}
{¬a,¬d}
{¬b,d,¬e}
.
.
.



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"

Welche Klausel wird hinzugefügt?

Es können verschiedene Klauseln hinzugefügt werden. 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 Visualisierungen von Lernen:

- Resolution (schon bekannt)
- 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

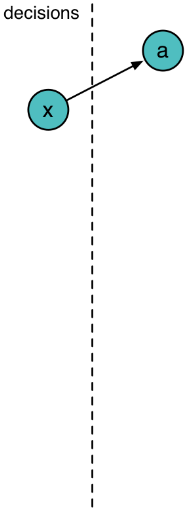
decisions

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

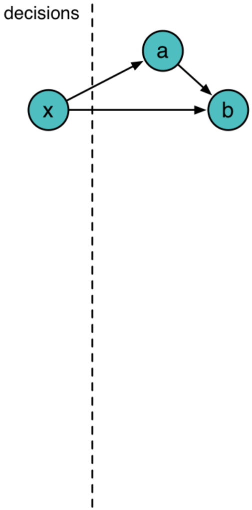
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\bot$	{¬u,¬m,h,¬w}
	m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

Aufbau des Implikationsgraphen



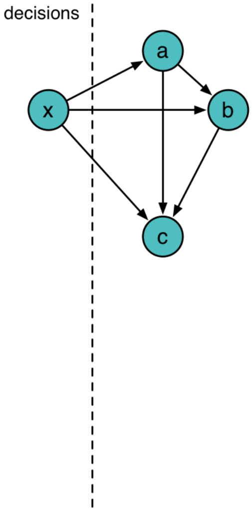
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



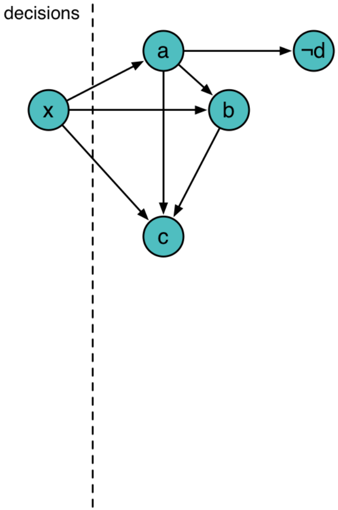
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\bot$	{¬u,¬m,h,¬w}
	m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

Aufbau des Implikationsgraphen



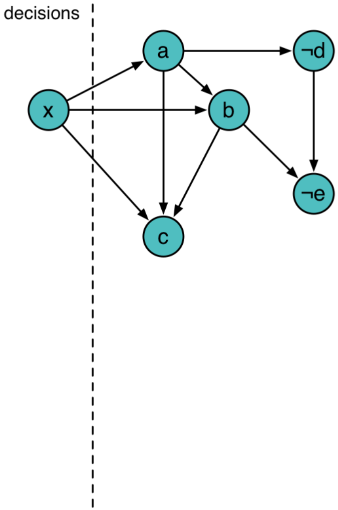
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



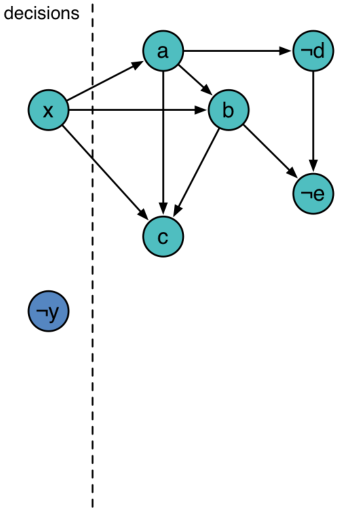
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\bot$	{¬u,¬m,h,¬w}
	m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

Aufbau des Implikationsgraphen



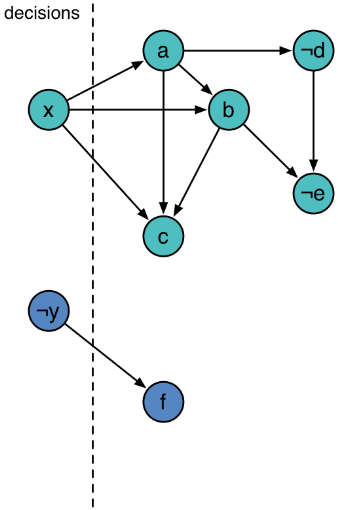
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	$\{\neg x, a\}$
	b	$\top$	$\{\neg x, b, \neg a\}$
	c	$\top$	$\{\neg x, \neg a, \neg b, c\}$
	d	$\bot$	$\{\neg a, \neg d\}$
	e	$\bot$	$\{\neg b, d, \neg e\}$
2	y	$\bot$	decision
	f	$\top$	$\{y, f\}$
	g	$\top$	$\{\neg f, g\}$
	h	$\bot$	$\{\neg f,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



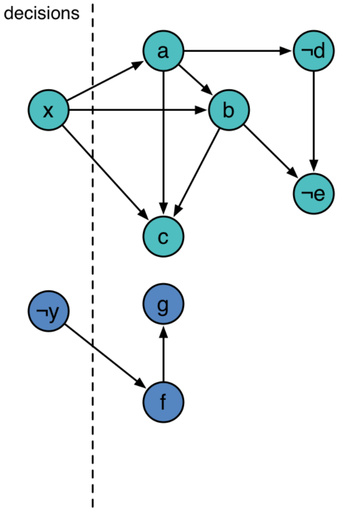
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



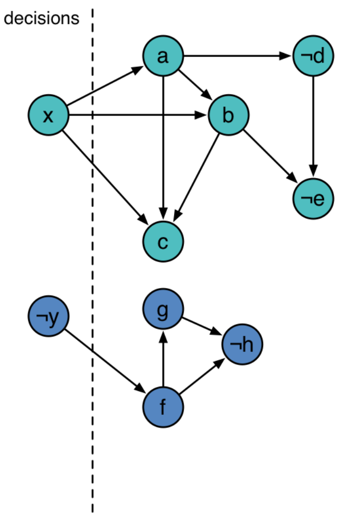
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



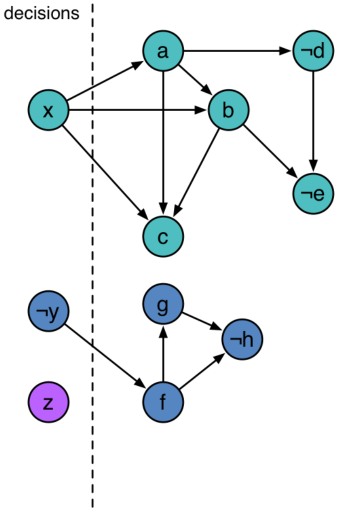
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



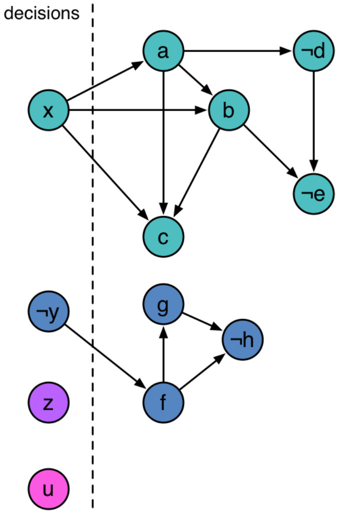
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\bot$	{¬a,¬d}
	e	$\bot$	{¬b,d,¬e}
2	y	$\bot$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\bot$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\bot$	{¬u,¬m,h,¬w}
	m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

Aufbau des Implikationsgraphen



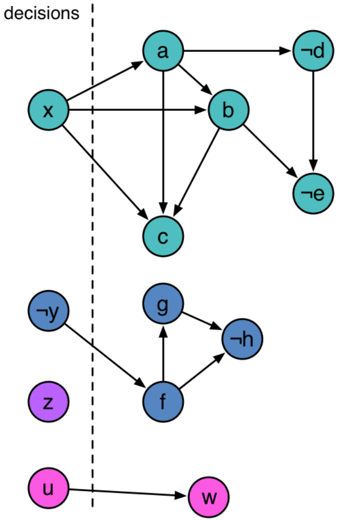
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



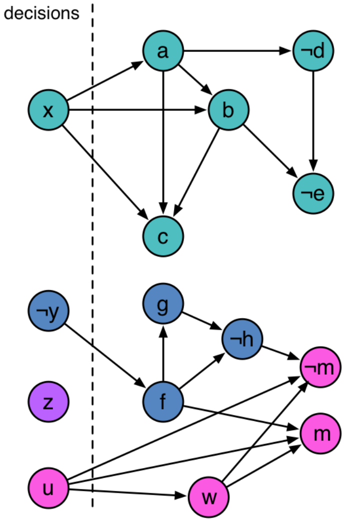
Level	Variable	Wert	Grund
1	x	⊤	decision
	a	⊤	{¬x,a}
	b	⊤	{¬x,b,¬a}
	c	⊤	{¬x,¬a,¬b,c}
	d	⊥	{¬a,¬d}
	e	⊥	{¬b,d,¬e}
2	y	⊥	decision
	f	⊤	{y,f}
	g	⊤	{¬f,g}
	h	⊥	{¬f,¬g,¬h}
3	z	⊤	decision
4	u	⊤	decision
	w	⊤	{¬u,w}
	m	⊥	{¬u,¬m,h,¬w}
	m	⊤	{¬u,¬f,m,¬w}

Aufbau des Implikationsgraphen



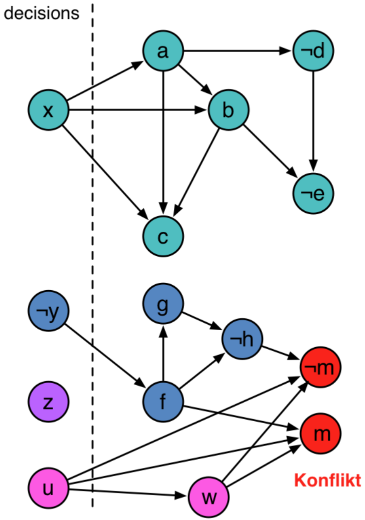
Level	Variable	Wert	Grund
1	x	$\top$	decision
	a	$\top$	{¬x,a}
	b	$\top$	{¬x,b,¬a}
	c	$\top$	{¬x,¬a,¬b,c}
	d	$\perp$	{¬a,¬d}
	e	$\perp$	{¬b,d,¬e}
2	y	$\perp$	decision
	f	$\top$	{y,f}
	g	$\top$	{¬f,g}
	h	$\perp$	{¬f,¬g,¬h}
3	z	$\top$	decision
4	u	$\top$	decision
	w	$\top$	{¬u,w}
	m	$\perp$	{¬u,¬m,h,¬w}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

Und welche Klausel lernen wir jetzt?

Erste Idee: Lernen von Entscheidungsvariablen

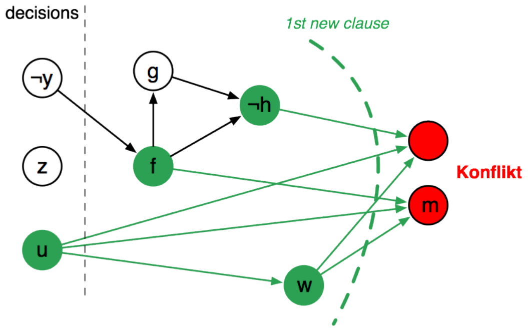
- Ursache für den Konflikt ist die spezielle Belegung der Entscheidungsvariablen. Diese darf so nicht vorkommen.
- Neue Klausel: $\neg(x \wedge \neg y \wedge z \wedge u) = (\neg x \vee y \vee \neg z \vee \neg u)$
- Aber: Die Belegung von x und z spielt für den Konflikt gar keine Rolle.
- Bei Hunderten von Variablen ist das häufig so, außerdem wäre die gelernte Klausel viel zu lang, also viel zu speziell.

Neue Idee: gezielte Konflikt-Analyse

- Ausgehend vom Konflikt, analysiere seine Ursachen genauer …

Und welche Klausel lernen wir jetzt?

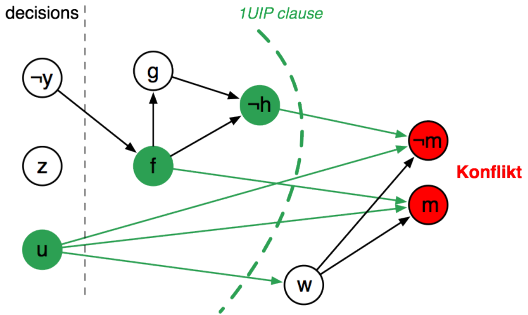
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Diese sind für den Konflikt verantwortlich.



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?

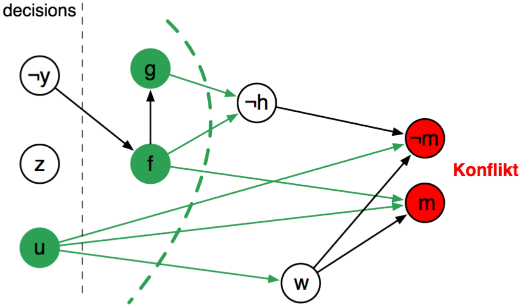
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Diese sind für den Konflikt verantwortlich.



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

Und welche Klausel lernen wir jetzt?

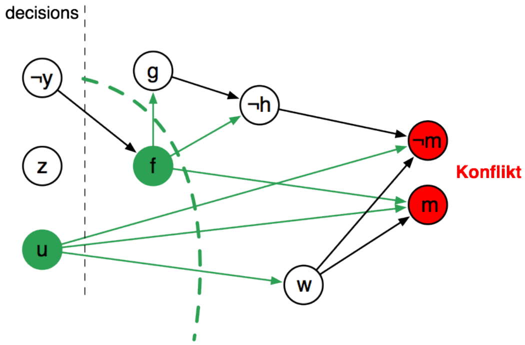
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Diese sind für den Konflikt verantwortlich.



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

Und welche Klausel lernen wir jetzt?

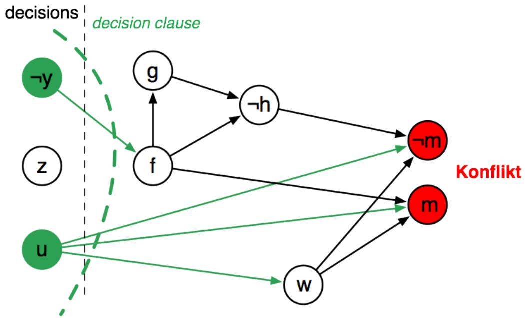
- Jeder Schnitt im Implikationsgraphen, der Entscheidungsvariablen von Konfliktvariablen trennt, ist ein gültiger *Cut*.
- Bilde eine neue Klausel aus allen Variablen, die eine ausgehende Kante durch den Cut besitzen. Diese sind für den Konflikt verantwortlich.



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. Diese 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 **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

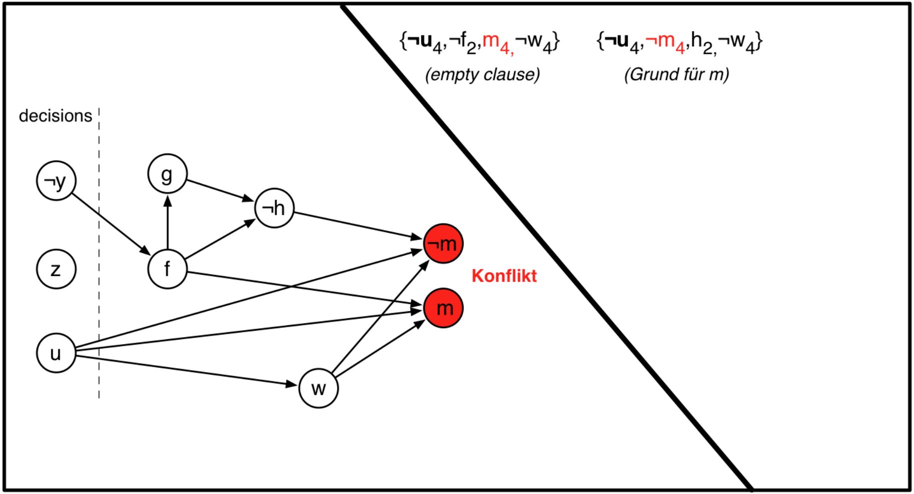
Die Schnittklauseln können ausgehend von der Konfliktklausel durch Resolutionsschritte berechnet werden.

- Zum Zeitpunkt des Konflikts gibt es (als Folge einer Entscheidung) zwei Unit-Klauseln:
  - 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 Uni-Propagation $\lambda = 1$ wird die Konfliktklausel nun zu einer Empty Clause, d.h. die Funktion wird an der Stelle 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. (K_r und R_r können keine komplementären Literale enthalten, da sonst eines der beiden $= 1$ gewesen wäre.)

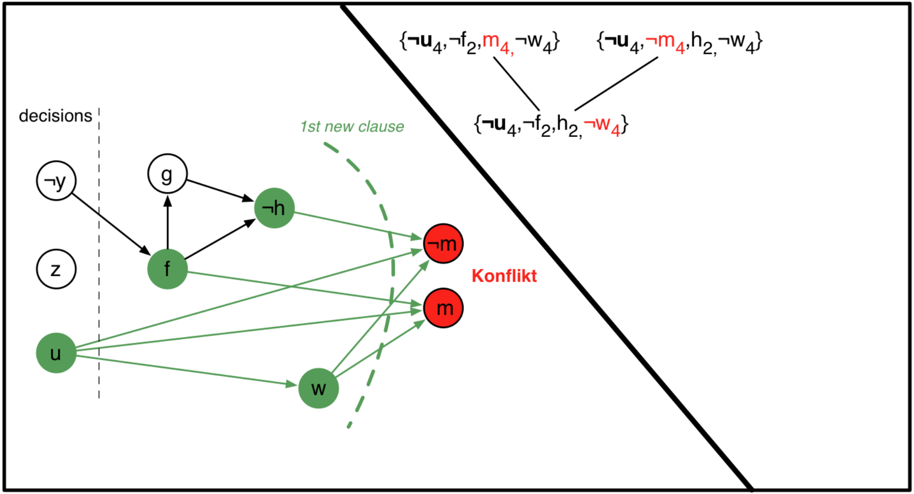
Berechnung von Schnittklauseln mit Resolution

- Sei μ irgend ein (früher) durch Unit Propagation mit 0 belegtes Literal in 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.

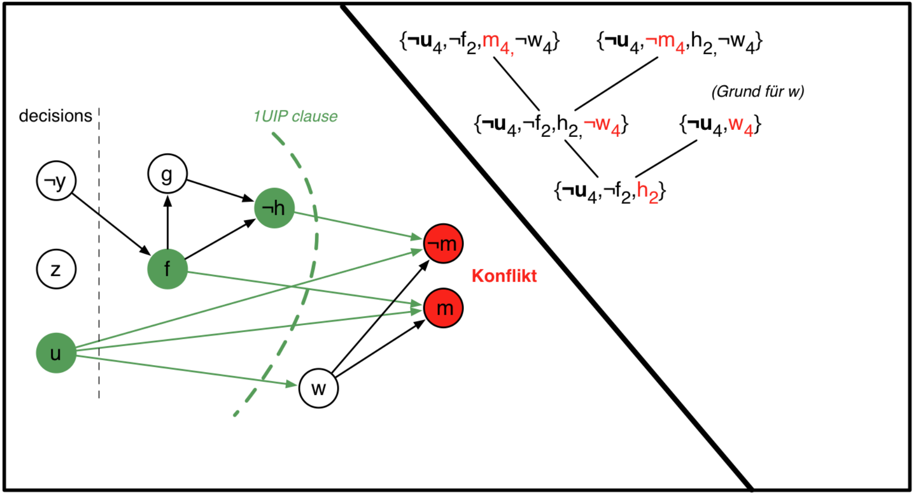
Berechnung von Schnittklauseln mit Resolution



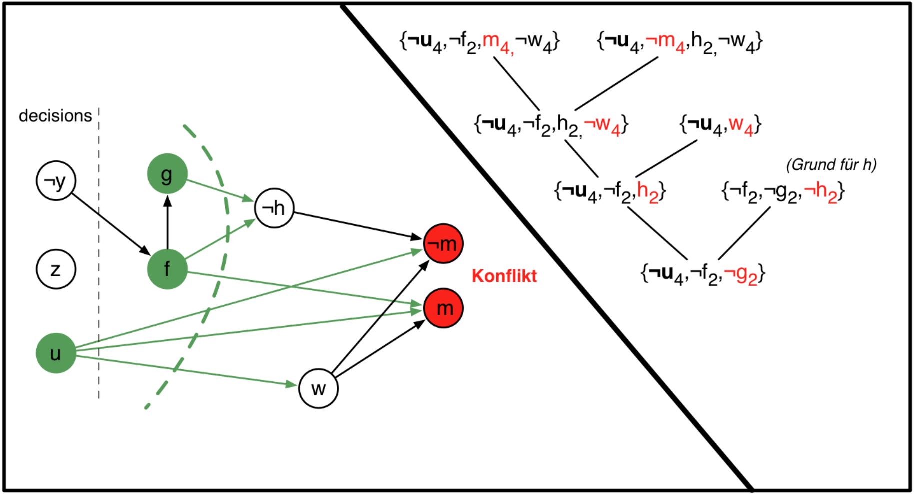
Berechnung von Schnittklauseln mit Resolution



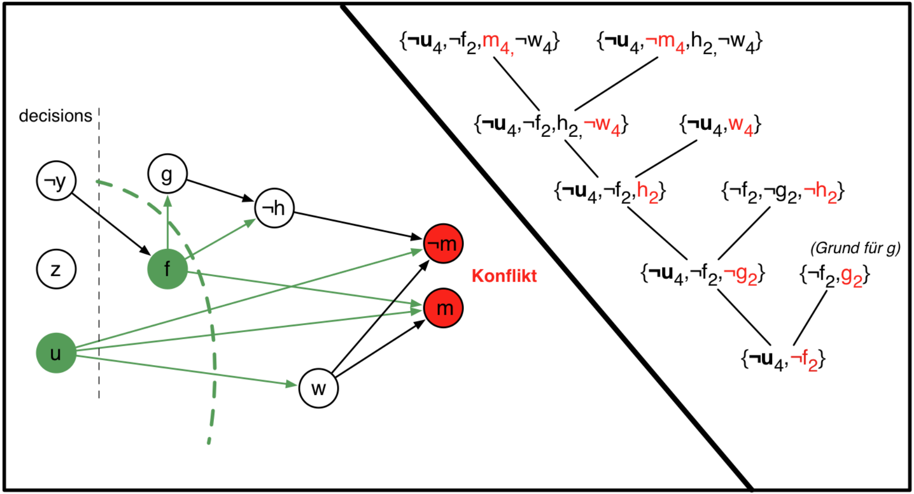
Berechnung von Schnittklauseln mit Resolution



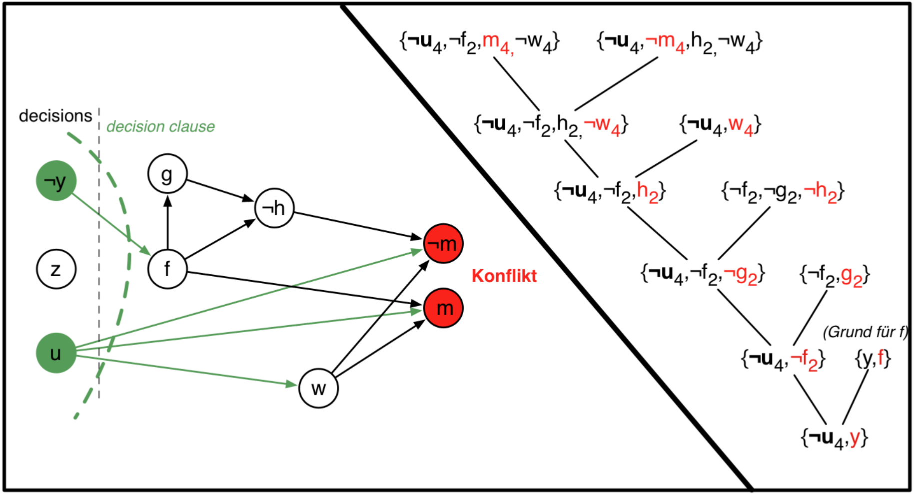
Berechnung von Schnittklauseln mit Resolution



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:

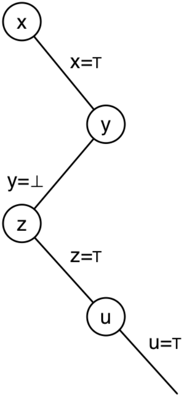
- First New Cut Clause: Die erste neue Klausel, die man durch Resolution erhält. Entspricht dem Cut, der auf der einen Seite nur die beiden Konfliktliterale enthält.
- 1UIP Clause (First Unique Implication Point): In der Klausel ist genau eine Variable (UIP) auf höchstem (größtem) Decision Level. Löscht man die Variablenbelegungen rückwärts bis inkl. diesem Level, wird die Klausel Unit. Man kann mit dem Löschen solange fortfahren wie die Klausel Unit bleibt, also bis zum zweithöchsten Level, 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 höchstem (größtem) Level der neuen Klausel, das $<$ als das aktuelle Decision Level ist.
- 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)*

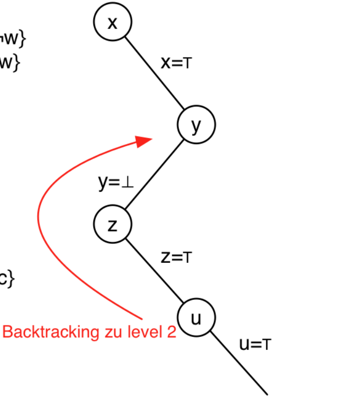


Konflikt: {¬u,¬f,m,¬w}

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)*

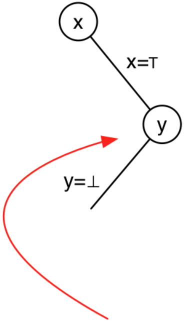


Konflikt: {¬u,¬f,m,¬w}

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

{¬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)*

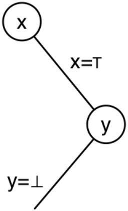


Backtracking zu level 2

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

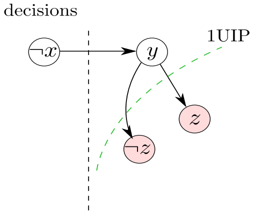
Nach UP von u sind alle gegebenen Klauseln erfüllt

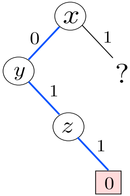
CDCL: Beispiel — Corner Case 1

Non-chronological backtracking mit Änderung der Belegungsreihenfolge.

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

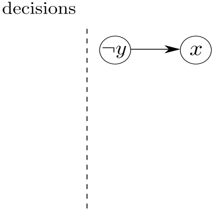
Level	Variable	Wert	Grund
1	X	0	decision
	y	1	$\{x, y\}$
	Z	1	$\{\neg y, z\}$
	Z	0	$\{\neg y, \neg z\}$

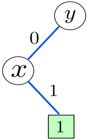




$$\{\{x, y\}, \{\neg y, z\}, \{\neg y, \neg z\}, \{\neg y\}\}$$

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



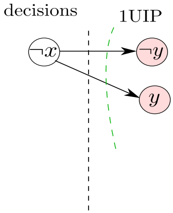


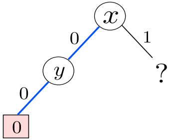
CDCL: Beispiel — Corner Case 2

Backtracking auf Ebene 0: Resultat UNSAT.

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

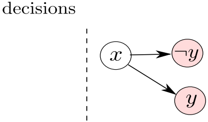
Level	Variable	Wert	Grund
1	x	0	decision
	y	0	$\{x,\neg y\}$
	y	1	$\{x,y\}$

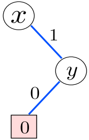




$$\{\{x,y\},\{x,\neg y\},\{\neg x,y\},\{\neg x,\neg y\},\{x\}\}$$

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





Konflikt auf Level 0: UNSAT

Der CDCL Algorithmus

```
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
```

Algorithm 1: sat(C)

Der Konfliktanalyse Algorithmus

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`
$C = C \cup \{\text{newclause}\}$; `// Füge neue Klausel hinzu`
level $=$ computeBacktrackLevel(newclause) ; `// Berechne Backtrack Level`
return *level*;

Algorithm 2: analyzeConf(ec, C)

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

Korrektheit des CDCL Algorithmus — 1[2]

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] Die folgenden Beweise finden sich in: L. Zhang, and S. Malik. **Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications.** In Proceedings of DATE '03, 2003

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. $\square$

Partielle Korrektheit + Termination $\Rightarrow$ Totale Korrektheit