SAT-Solving und Anwendungen

Implementierung effizienter SAT Solver

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

Universität Tübingen

16. November 2017

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Fahrplan für heute

1. **Auswahlheuristiken für Entscheidungen**
   - Wie wählen wir die nächste Entscheidungsvariable?
2. **Effiziente Unit Propagation**
   - Wie bekommen wir Unit Propagation effizient?
3. **Verbesserungen beim Lernen**
   - Wie minimieren wir gelernte Klauseln?
4. **Clause Deletion**
   - Wann löschen wir welche gelernten Klauseln wieder?
5. **Restart Strategien**
   - Wann starten wir den Solver neu?

Auswahlheuristiken

Aktivitätsheuristiken

Wähle die Variable, die am aktivsten ist.

- Aktivität kann verschieden definiert sein
- Häufig: Variable, die kürzlich in vielen Konflikten (empty clauses) vorkam

Idee:

- Variablen, die oft in Konflikten vorkommen, spielen eine herausragende Rolle und sollten zuerst belegt werden
- Variablen haben verschieden großen Einfluss auf eine Formel (Bsp: Softwareverifikation, Variable die über einen großen `if-then-else`-Branch entscheidet beeinflusst Ergebnis mehr als andere Variablen)

Beispiel für Aktivitätsheuristik:

- **VSIDS** (variable state independent decaying sum)

VSIDS

Berechnung:

- Jede Variable bekommt eine *Aktivität act* zugewiesen
- Initial ist *act* die Anzahl der Vorkommen der Variable (positiv + negativ)
- Für jede gelernte Klausel $(x_1 \vee x_2 \vee ... \vee x_n)$: erhöhe *act* für alle x_i um konstanten Betrag c (1)
- Teile periodisch alle Aktivitäten durch einen Faktor f (2)
- Wähle Variable mit höchster Aktivität

Erklärung:

(1) Aktivität einer Variable wird höher, je häufiger sie in Konflikten auftaucht

(2) Aktivität aller Variablen wird von Zeit zu Zeit verringert, d.h. Konflikte die in der nahen Vergangenheit aufgetreten sind, werden bevorzugt

Damit:

- Wahl der Variablen, die **aktuell** (im aktuellen Suchbaum) am **häufigsten in Konflikten** vorkam

Fahrplan für heute: Wo stehen wir?

1. **Auswahlheuristiken für Entscheidungen** ✓
   - VSIDS
2. **Effiziente Unit Propagation**
   - Wie bekommen wir Unit Propagation effizient?
3. **Verbesserungen beim Lernen**
   - Wie minimieren wir gelernte Klauseln?
4. **Clause Deletion**
   - Wann löschen wir welche gelernten Klauseln wieder?
5. **Restart Strategien**
   - Wann starten wir den Solver neu?

Motivation für effiziente Unit Propagation

Beobachtung:

- Durch Profiling herausgefunden: SAT Solver befinden sich ca. 90% der Zeit in Unit Propagation
- Das ist auch logisch: Der exponentiell wachsende Lösungsraum wird durch die Zwänge eingeschränkt, die die UP ermöglichen. Nur UP macht effizientes SAT-Solving möglich (und UIP-Lernen trägt unmittelbar zu UP bei).
- UP ist der Schlüssel zur effizienten Implementierung von SAT Solvern

Naive Unit Propagation

- **Datenstruktur:** Jede Variable speichert eine *Adjazenzliste* von Klauseln, in denen sie als Literal vorkommt
- **Vorgehen:** Bei jeder Belegung einer Variable werden alle zugehörigen Klauseln der Adjazenzliste analysiert, ob sie nun unit sind
- **Problem:** Sehr viele überflüssige Analysen, da die meisten Klauseln nicht unit werden (d.h. jede Klausel mit n Literalen wird $n-1$ mal analysiert, bevor sie Unit ist)

Lazy Datenstrukturen

Globaler Verlauf des SAT Solving

Ein CDCL Solver belegt solange Variablen bis er entweder durch einen Konflikt auf Ebene Null gestoppt wird (Ergebnis: UNSAT) oder *alle* Variablen der Eingabeformel belegt sind (Ergebnis SAT).

Es lohnt sich zeitlich nicht, nach jeder Variablenbelegung alle Klauseln zu inspizieren, ob sie evtl. schon erfüllt sind, denn es gibt sehr viele Klauseln und sehr viele Variablen.

Status einer Klausel

Eine Klausel kann nach jeder Belegung einen von 4 Zuständen einnehmen: SAT (*true*), UNSAT (*false*), UNIT, UNKNOWN.
Im Verlauf des SAT Solving sind nur die Zustände UNIT und UNSAT interessant, da sie eine Unit Propagation bzw. eine Konfliktbehandlung auslösen. Diese Zustände können mit einer "faulen" Überwachung *(lazy watch)* erkannt werden.

Two Watched Literals

Idee: Um die Zustände UNIT und UNSAT zu erkennen, genügt es, 2 Literale pro Klausel zu „beobachten“ / referenzieren.

- Jede Variable zeigt auf die Literale, in denen sie vorkommt.
- Anfangs sind die beobachteten Literale unbelegt
- Solange 2 Literale unbelegt sind, kann die Klausel weder UNIT noch UNSAT sein, ihr Status ist also uninteressant und braucht nicht ermittelt zu werden.
- Sobald eine Variable so belegt wird, dass ein beobachtetes Literal zu 0 evaluiert, startet die Suche nach einem Ersatzliteral für die Beobachtung.
- Wird dabei ein erfülltes oder ein unbelegtes Literal gefunden, wird nun dieses beobachtet. (Der Zustand SAT interessiert dabei nicht weiter.)
- Endet die Suche erst beim anderen beobachteten Literal, so wird dieses nun doppelt beobachtet, und die Klausel ist UNIT, SAT, oder UNSAT je nach dem Wert dieses Literals. (Der Zustand SAT interessiert nicht weiter.)
- UNSAT ergibt sich nur im Spezialfall einer Konfliktklausel. Diese ist *gleichzeitig* mit der Reason Klausel UNIT geworden. Die Propagation der Reason Klausel wird zuerst vorgenommen und führt dazu, dass das (doppelt beobachtete) Unit Literal der Konfliktklausel *false* wird.

2 Watched Literals stateful Head/Tail Datenstruktur

- Erstmals 1997 im SAT Solver *Sato* benutzt

UP Erkennung mit Head/Tail watched Literals

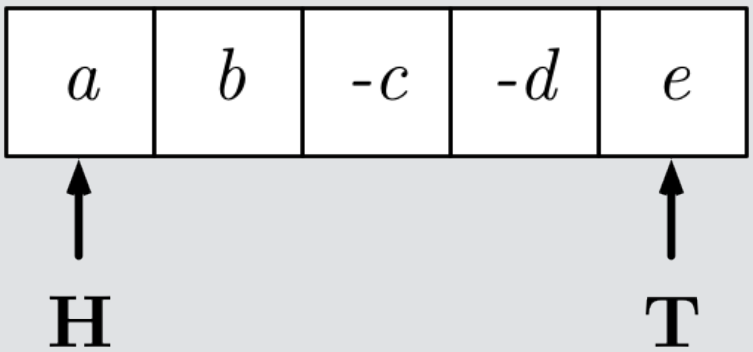
1. Mit jeder Klausel werden zwei Referenzen *("watches")* assoziiert:
   - Head H — initial auf dem ersten Literal der Klausel
   - Tail T — initial auf dem letzten Literal der Klausel

   Invariante: Links von H und rechts von T sind alle Literale *false*. Variablen, die *true* oder *unknown* sind befinden sich nur im Bereich von H bis T.
2. Wird ein referenziertes Literal belegt und evaluiert zu 1, so geschieht nichts.
3. Wird ein referenziertes Literal belegt und evaluiert zu 0, so wird nach innen hin nach einer neuen Referenz gesucht. Die bisherige Referenz bleibt für eventuelles Backtracking gespeichert.
   1. Wird ein unbelegtes oder erfülltes Literal gefunden, so wird **eine neue** Referenz auf dieses gesetzt.
   2. Wird stattdessen die zweite Referenz erreicht, so ist die Klausel je nach Evaluation dieser Referenz UNIT oder SAT (letzteres interessiert nicht weiter).
   3. Der Fall UNSAT kommt nur dann vor, wenn die Klausel schon UNIT ist und (bevor dies propagiert werden konnte) das verbleibende Literal durch eine andere Unit-Propagation mit *false* belegt wird.

Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer erfüllten Klausel



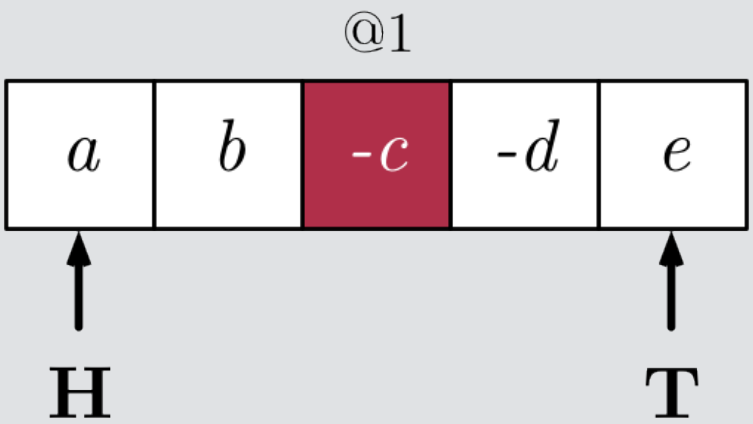
Level	Variable	Belegung

- Initial zeigen Head und Tail auf Anfang und Ende der Klausel

Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer erfüllten Klausel



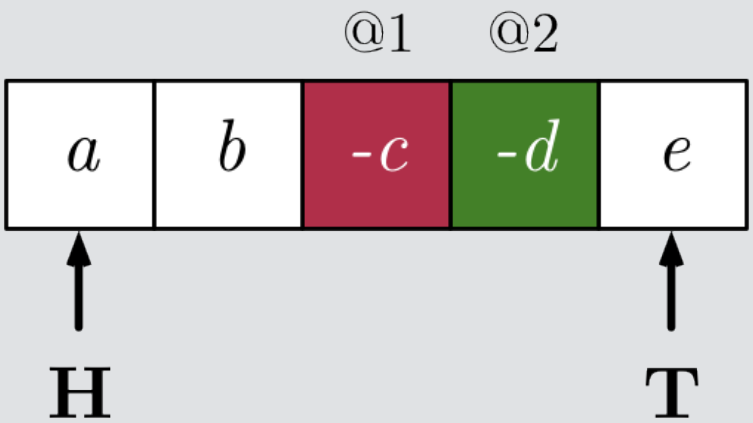
Level	Variable	Belegung
1	c	$\top$

- Variable c wird belegt
- Da c nicht referenziert ist, wird die Klausel nicht analysiert

Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer erfüllten Klausel



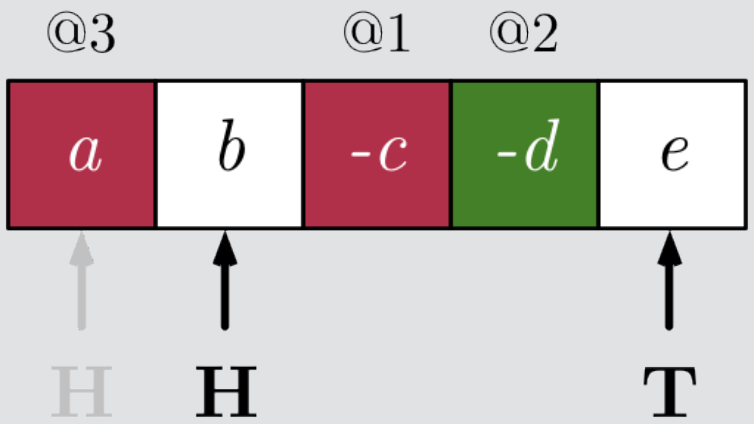
Level	Variable	Belegung
1	c	$\top$
2	d	$\bot$

- Variable d wird belegt
- Da d nicht referenziert ist, wird die Klausel nicht analysiert
- Klausel ist bereits SAT, was wir jedoch noch nicht wissen (lazy)

Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer erfüllten Klausel



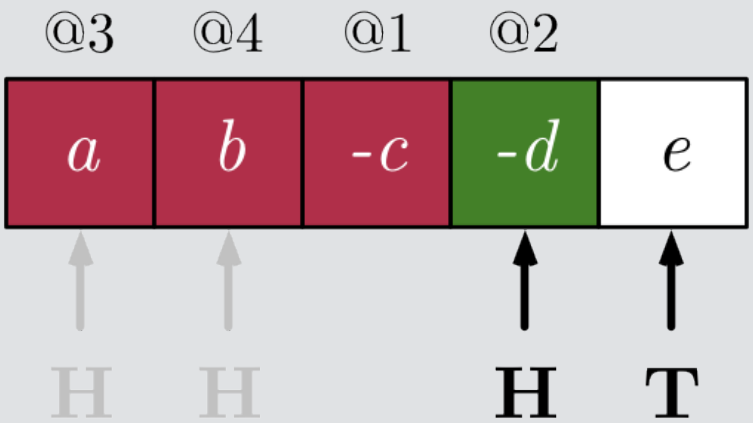
Level	Variable	Belegung
1	c	$\top$
2	d	$\bot$
3	a	$\bot$

- Variable a wird belegt
- a ist referenziert und evaluiert zu 0, daher wird die Referenz nach innen bewegt
- Alte Referenz wird gespeichert (für Backtracking)
- Nächstes unbelegtes Literal b wird referenziert

Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer erfüllten Klausel



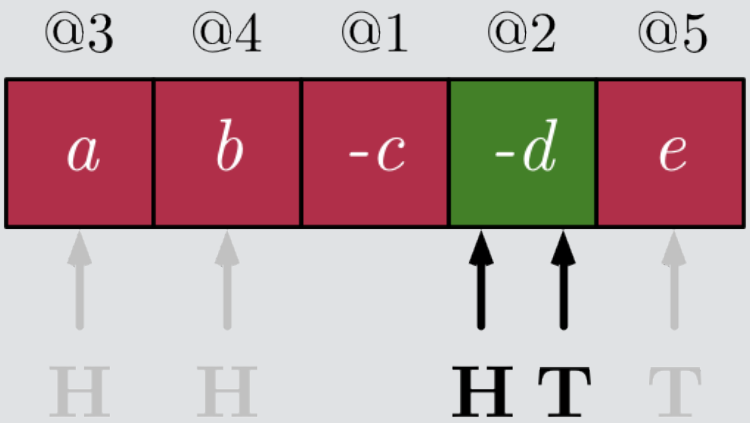
Level	Variable	Belegung
1	c	$\top$
2	d	$\bot$
3	a	$\bot$
4	b	$\bot$

- Variable b wird belegt
- b ist referenziert und evaluiert zu 0, daher wird die Referenz nach innen bewegt
- Alte Referenz wird gespeichert (für Backtracking)
- Erfülltes Literal $\neg d$ wird gefunden, d.h. wir wissen nun, dass die Klausel SAT ist

Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer erfüllten Klausel



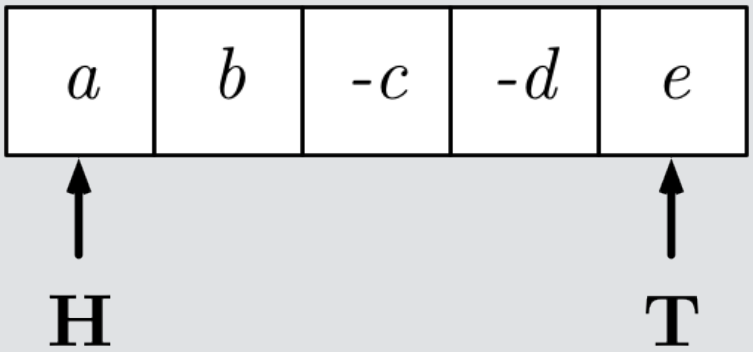
Level	Variable	Belegung
1	c	$\top$
2	d	$\bot$
3	a	$\bot$
4	b	$\bot$
5	e	$\bot$

- Variable e wird belegt
- e ist referenziert und evaluiert zu 0, daher wird die Referenz nach innen bewegt
- Alte Referenz wird gespeichert (für Backtracking)
- Head Referenz wird gefunden, wir wissen dass die Klausel SAT ist

Head/Tail Datenstruktur — Beispiel 2 (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



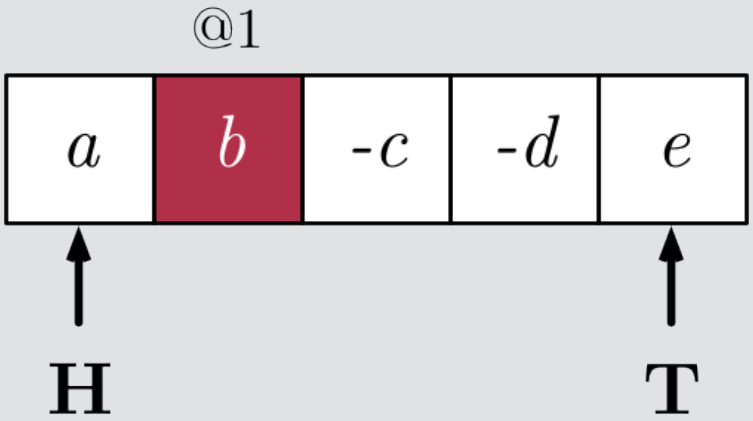
Level	Variable	Belegung

- Initial zeigen Head und Tail auf Anfang und Ende der Klausel

Head/Tail Datenstruktur — Beispiel 2 (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



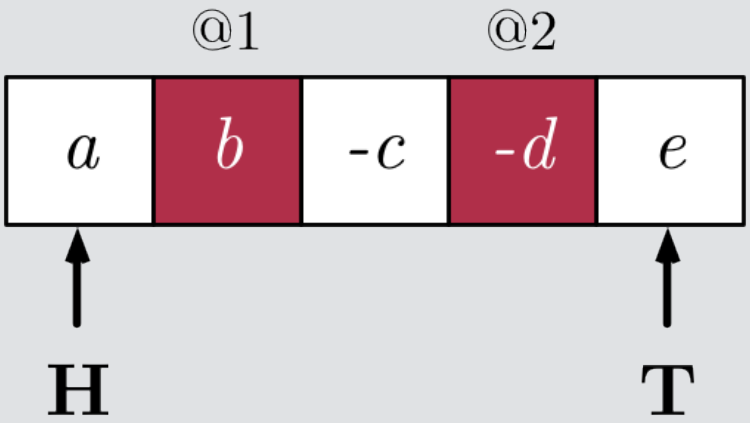
Level	Variable	Belegung
1	b	$\bot$

- Variable b wird belegt
- Da b nicht referenziert ist, wird die Klausel nicht analysiert

Head/Tail Datenstruktur — Beispiel 2 (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



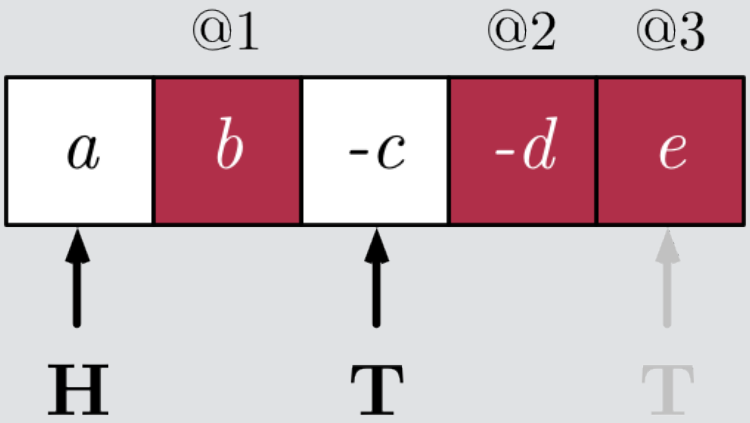
Level	Variable	Belegung
1	b	$\bot$
2	d	$\top$

- Variable d wird belegt
- Da d nicht referenziert ist, wird die Klausel nicht analysiert

Head/Tail Datenstruktur — Beispiel 2 (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



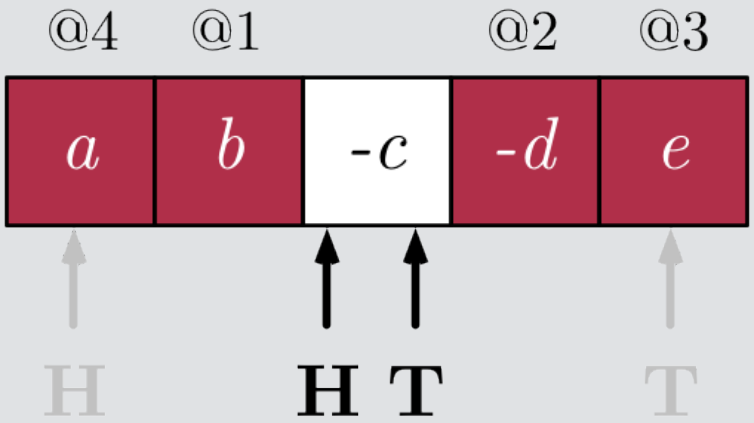
Level	Variable	Belegung
1	b	$\bot$
2	d	$\top$
3	e	$\bot$

- Variable e wird belegt
- e ist referenziert und evaluiert zu 0, daher wird die Referenz nach innen bewegt
- Alte Referenz wird gespeichert (für Backtracking)
- Nächstes unbelegtes Literal $\neg c$ wird referenziert

Head/Tail Datenstruktur — Beispiel 2 (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



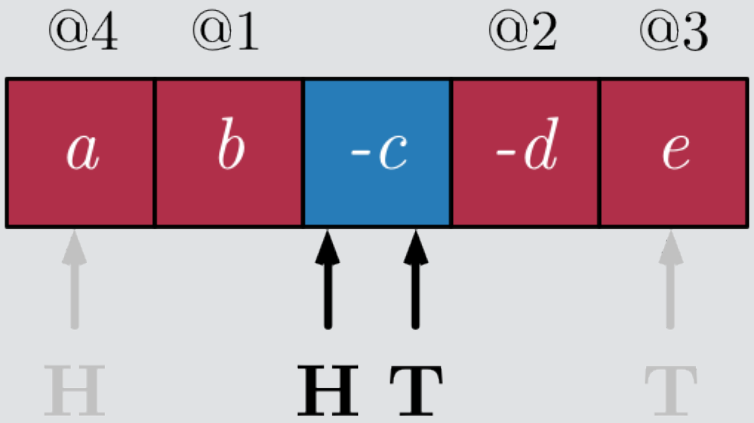
Level	Variable	Belegung
1	b	$\bot$
2	d	$\top$
3	e	$\bot$
4	a	$\bot$

- Variable a wird belegt
- a ist referenziert und evaluiert zu 0, daher wird die Referenz nach innen bewegt
- Alte Referenz wird gespeichert (für Backtracking)
- Tail Referenz wird gefunden

Head/Tail Datenstruktur — Beispiel 2 (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



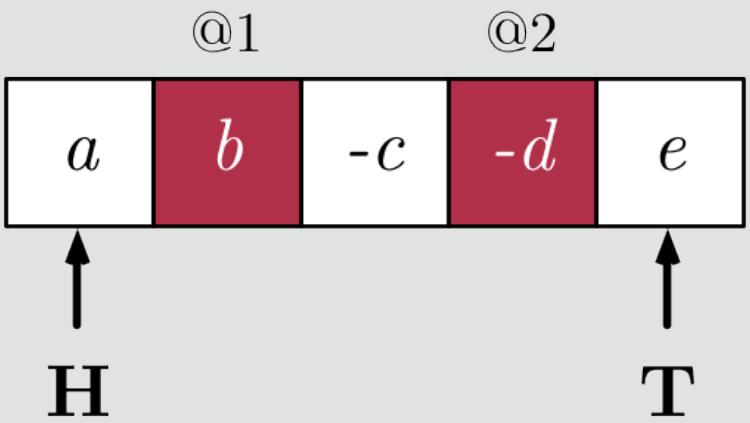
Level	Variable	Belegung
1	b	$\bot$
2	d	$\top$
3	e	$\bot$
4	a	$\bot$

- Head und Tail zeigen auf das selbe Literal, welches unbelegt ist

$\Rightarrow$ Klausel ist nun UNIT

Head/Tail Datenstruktur — Beispiel 3 (Backtracking)

Was passiert beim Backtracking?



Level	Variable	Belegung
1	b	$\bot$
2	d	$\top$

- Bei Backtracking zu Level 2 müssen auch die Head und Tail Referenzen wieder hergestellt werden, damit die Invariante weiterhin gilt.

Großer Nachteil: Im worst-case benötigt man für eine Klausel mit n Literalen n Referenzen (aktuellen Head + Tail und alte Referenzen für Backtracking)

Watched Literals stateless Datenstruktur

- **Idee:** Verzichte auf die Ordnung (Head $<$ Tail) zwischen den beiden Referenzen und damit auf die Invariante.
- Die Suche nach einem Ersatzliteral muss sich daher nötigenfalls über die gesamte Klausel erstrecken.
- Dadurch „zustandslose“ Referenzen, die nach Backtracking weiter benutzt werden können.
- Erstmals 2001 im SAT Solver *Chaff* benutzt

Vorgehen

1. Mit jeder Klausel werden zwei Referenzen (*watched literals*) assoziiert
2. Wird die Variable eines referenzierten Literals belegt sodass dieses zu 0 evaluiert, so wird die entsprechende Referenz bewegt (in beliebige Richtung, zirkulär über die Klausel).

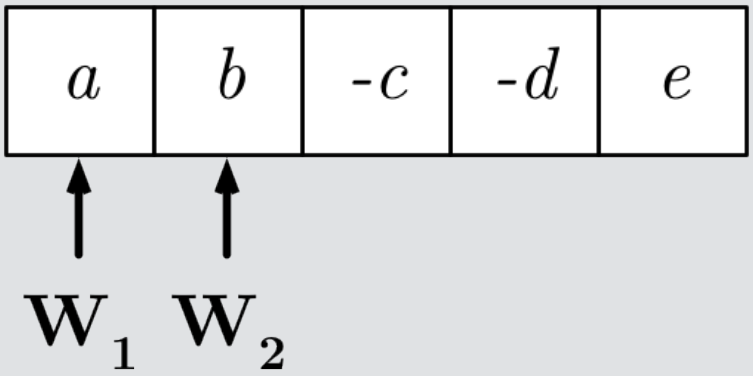
Vorteil: Die Anzahl von Referenzen pro Klausel ist konstant 2 (nach Backtracking können die aktuellen Referenzen beibehalten werden, da die Invariante weg ist.)

Nachteil: Um festzustellen, dass eine Klausel UNIT oder UNSAT ist, müssen alle Literale der Klausel durchgegangen werden, da die Invariante weg ist.

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



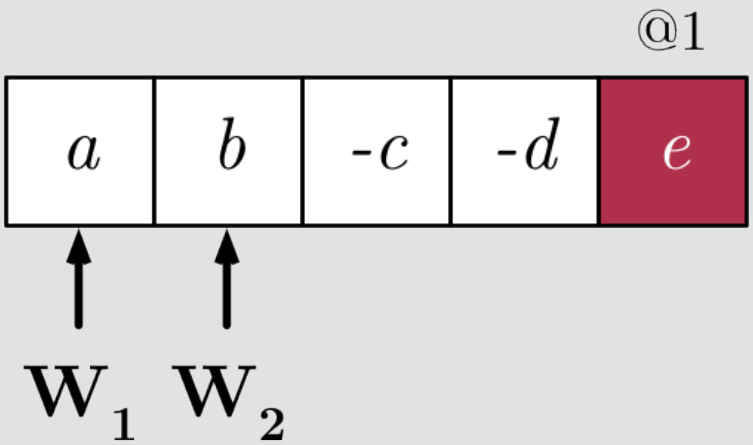
Level	Variable	Belegung

- Initial zeigen die beiden Watched Literal Referenzen auf beliebige Literale der Klausel

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



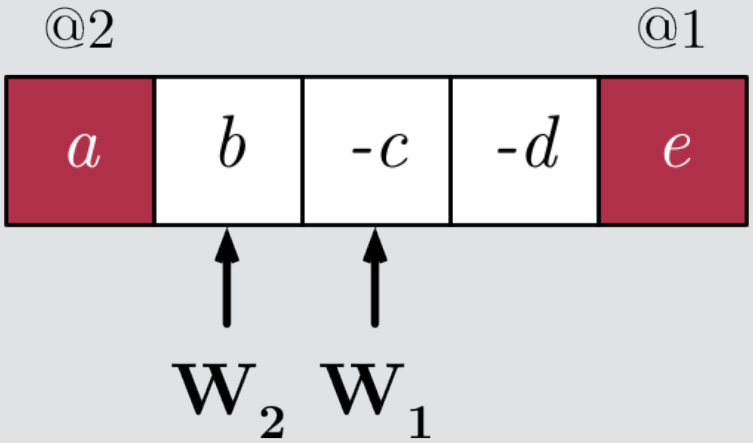
Level	Variable	Belegung
1	e	$\bot$

- Variable e wird belegt
- Da e nicht referenziert ist, wird die Klausel nicht analysiert

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



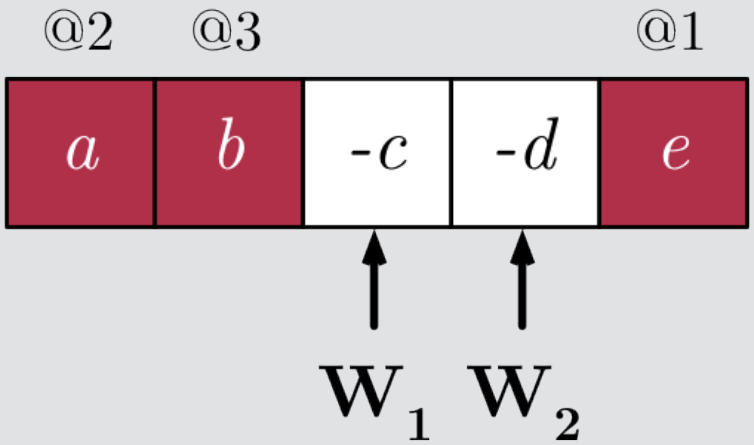
Level	Variable	Belegung
1	e	$\bot$
2	a	$\bot$

- Variable a wird belegt
- a ist referenziert und evaluiert zu 0, daher wird die Referenz bewegt (in diesem Fall nach rechts)
- Neues unbelegtes Literal $\neg c$ wird gefunden und referenziert

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



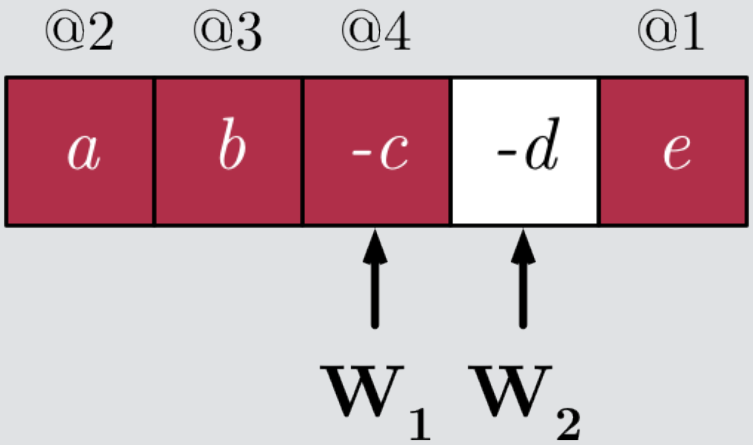
Level	Variable	Belegung
1	e	$\bot$
2	a	$\bot$
3	b	$\bot$

- Variable b wird belegt
- b ist referenziert und evaluiert zu 0, daher wird die Referenz bewegt (in diesem Fall nach rechts)
- Neues unbelegtes Literal $\neg d$ wird gefunden und referenziert

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



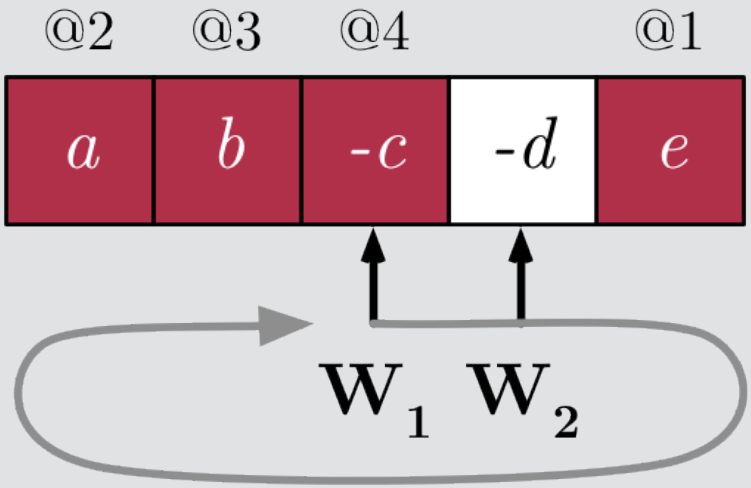
Level	Variable	Belegung
1	e	$\bot$
2	a	$\bot$
3	b	$\bot$
4	c	$\top$

- Variable c wird belegt
- c ist referenziert und evaluiert zu 0, daher wird die Referenz bewegt

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



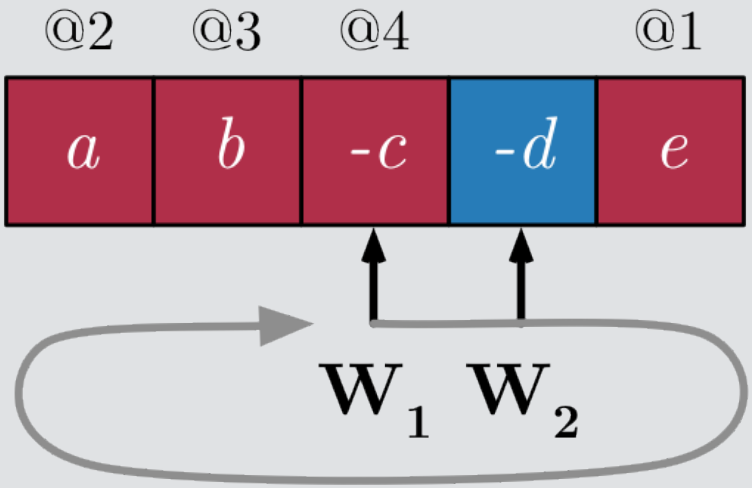
Level	Variable	Belegung
1	e	$\bot$
2	a	$\bot$
3	b	$\bot$
4	c	$\top$

- Variable c wird belegt
- c ist referenziert und evaluiert zu 0, daher wird die Referenz bewegt
- Es wird bis zum Ende der Klausel nach rechts gesucht, jedoch kein unbelegtes und unreferenziertes Literal gefunden, d.h. Beginn von vorne, bis man wieder bei der original Referenz W_1 landet

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



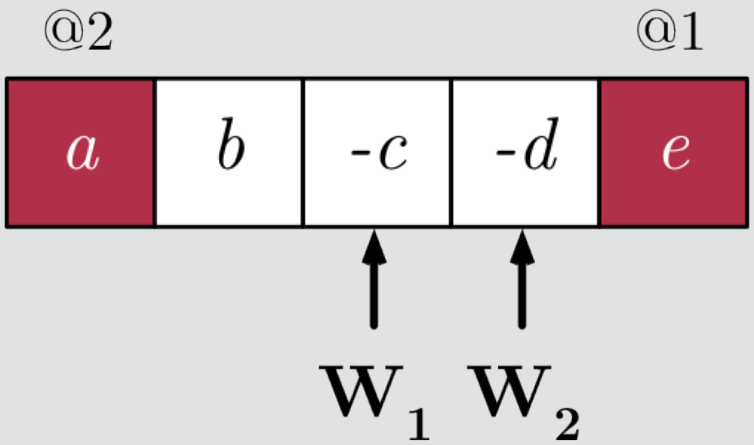
Level	Variable	Belegung
1	e	$\bot$
2	a	$\bot$
3	b	$\bot$
4	c	$\top$

- Variable c wird belegt
- c ist referenziert und evaluiert zu 0, daher wird die Referenz bewegt
- Es wird bis zum Ende der Klausel nach rechts gesucht, jedoch kein unbelegtes und unreferenziertes Literal gefunden, d.h. Beginn von vorne, bis man wieder bei der original Referenz W_1 landet
- Wert von W_2 entscheidet Status der Klausel: da W_2 unbelegt $\Rightarrow$ UNIT

Watched Literals — Beispiel (Unit Klausel)

- Klausel $a \vee b \vee \neg c \vee \neg d \vee e$

Erkennen einer Unit Klausel



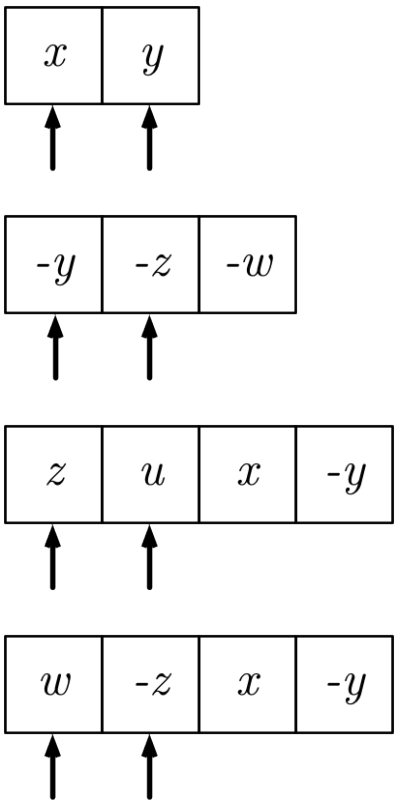
Level	Variable	Belegung
1	e	$\bot$
2	a	$\bot$

- Bei Backtracking (z.B. zu Level 2) können die Watched Literals beibehalten werden
- unbelegte Watched Literals bleiben unbelegt
- Keine Annahmen über Belegungsstatus links und rechts der Referenzen

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

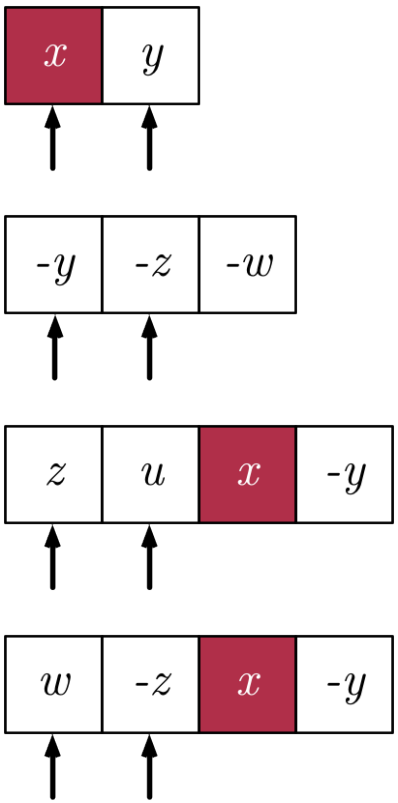
Level	Variable	Belegung	Reason



- Initial sind alle Watched Literals auf die ersten beiden Literale gesetzt

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

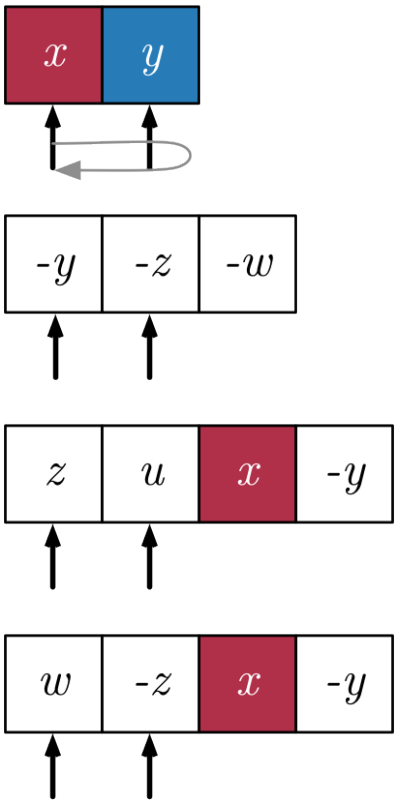


Level	Variable	Belegung	Reason
1	x	$\bot$	decision

- Decision: $x \mapsto \bot$
- Alle Klauseln in denen x beobachtet wird, müssen analysiert werden
- Nur in **Klausel 1** ist x referenziert und evaluiert zu 0

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

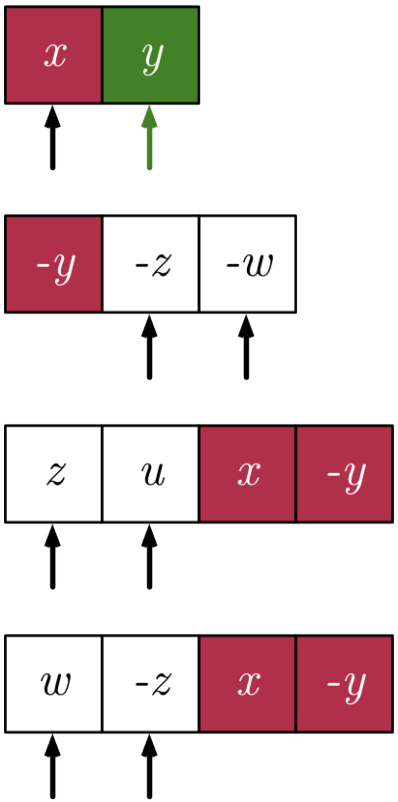


Level	Variable	Belegung	Reason
1	x	$\bot$	decision

- Decision: $x \mapsto \bot$
- **Klausel 1:** Neue Referenz wird nicht gefunden
- Da zweite Referenz auf unbelegtes Literal zeigt $\Rightarrow$ UNIT

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

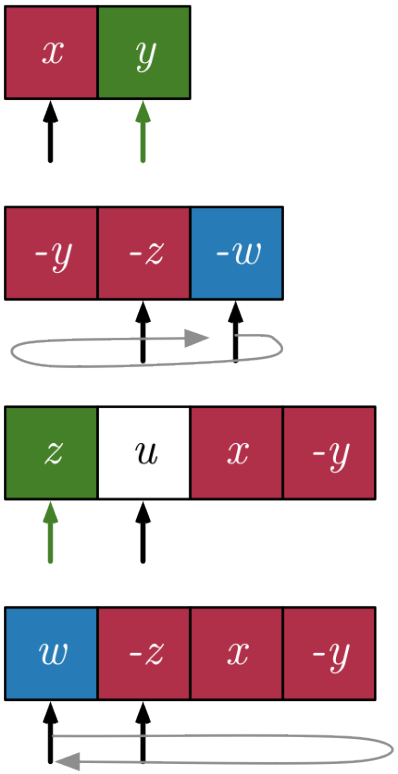


Level	Variable	Belegung	Reason
1	x	$\bot$	decision
	y	$\top$	$\{x, y\}$

- Unit Propagation: $y \mapsto \top$
- Alle Klauseln, in denen y referenziert ist, müssen analysiert werden
- **Klausel 1:** y ist referenziert, evaluiert jedoch zu 1, d.h. kein Handlungsbedarf, Klausel ist SAT
- **Klausel 2:** y ist referenziert und evaluiert zu 0, d.h. Referenz muss verschoben werden (auf $\neg w$)
- In allen *unresolved* Klauseln zeigen nun die Watched Literals auf unbelegte Literale, d.h. keine weiteren Unit Propagations möglich

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

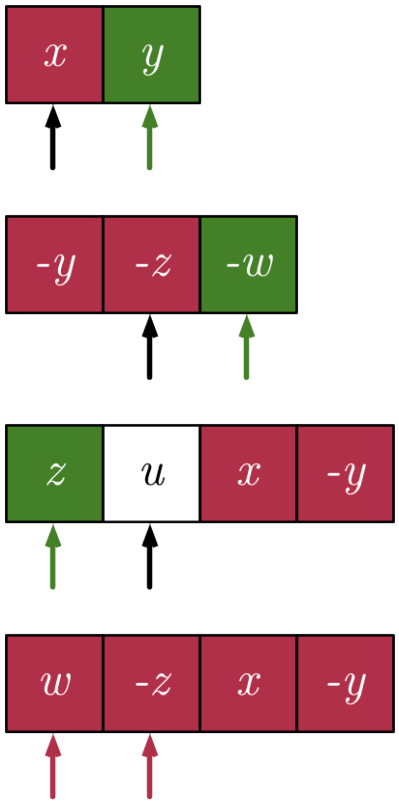


Level	Variable	Belegung	Reason
1	x	$\bot$	decision
	y	$\top$	$\{x, y\}$
2	z	$\top$	decision

- Decision: $z \mapsto \top$
- Alle Klauseln, in denen z referenziert ist, müssen analysiert werden
- **Klausel 2 und 4:** z ist referenziert, und evaluiert zu 0, d.h. Referenz muss verschoben werden $\Rightarrow$ Klausel ist UNIT
- **Klausel 3:** z ist referenziert, evaluiert jedoch zu 1, d.h. kein Handlungsbedarf, Klausel ist SAT

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

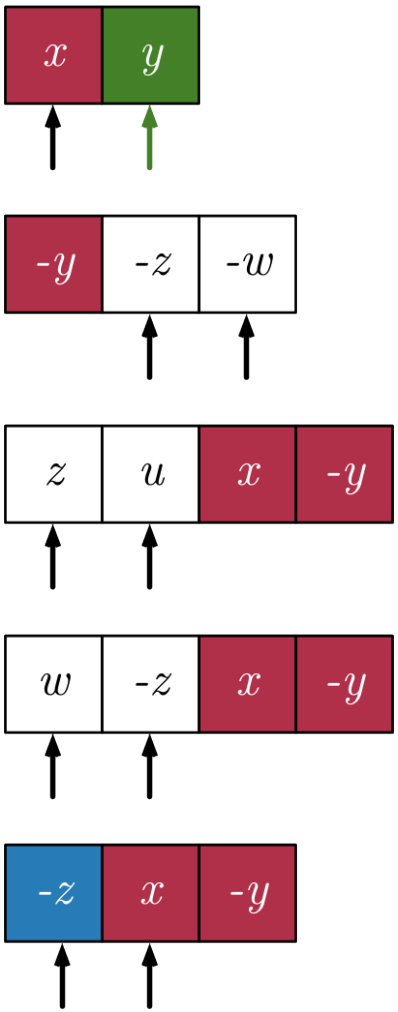


Level	Variable	Belegung	Reason
1	x	$\bot$	decision
	y	$\top$	$\{x, y\}$
2	z	$\top$	decision
	w	$\bot$	$\{\neg y, \neg z, \neg w\}$
	w	$\top$	$\{w, \neg z, x, \neg y\}$

- **Konflikt** mit Variable w
- **Klausel 4** ist leer

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

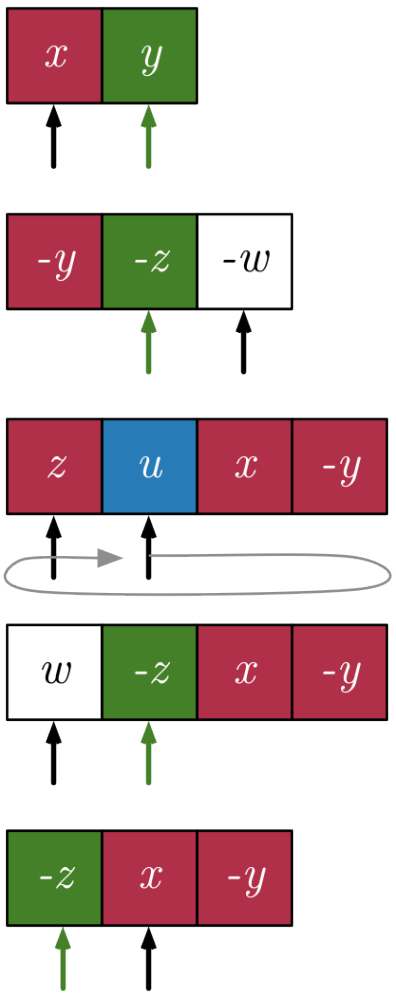


Level	Variable	Belegung	Reason
1	x	$\bot$	decision
	y	$\top$	$\{x, y\}$

- **Konflikt** mit Variable w
- **Klausel 4** ist leer
- Lernen der 1-UIP Klausel $\{\neg z, x, \neg y\}$
- Backtracking zu Level 1
- Alle Watched Literal Referenzen bleiben erhalten
- Neue **Klausel 5** ist nun UNIT

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

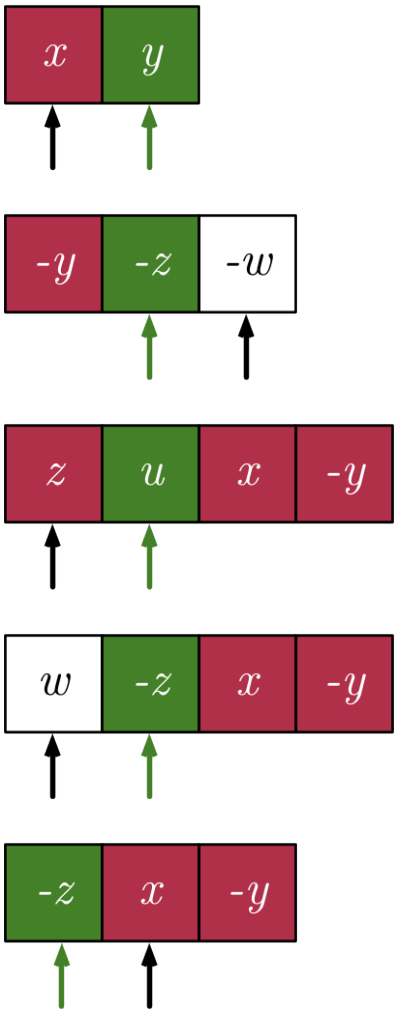


Level	Variable	Belegung	Reason
1	x	$\bot$	decision
	y	$\top$	$\{x, y\}$
	z	$\bot$	$\{\neg z, x, \neg y\}$

- Unit Propagation: $z \mapsto \bot$
- **Klausel 2, 4 und 5:** z ist referenziert, evaluiert jedoch zu 1, d.h. kein Handlungsbedarf, Klauseln sind SAT
- **Klausel 3:** z ist referenziert, und evaluiert zu 0, d.h. Referenz muss verschoben werden $\Rightarrow$ Klausel ist UNIT

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

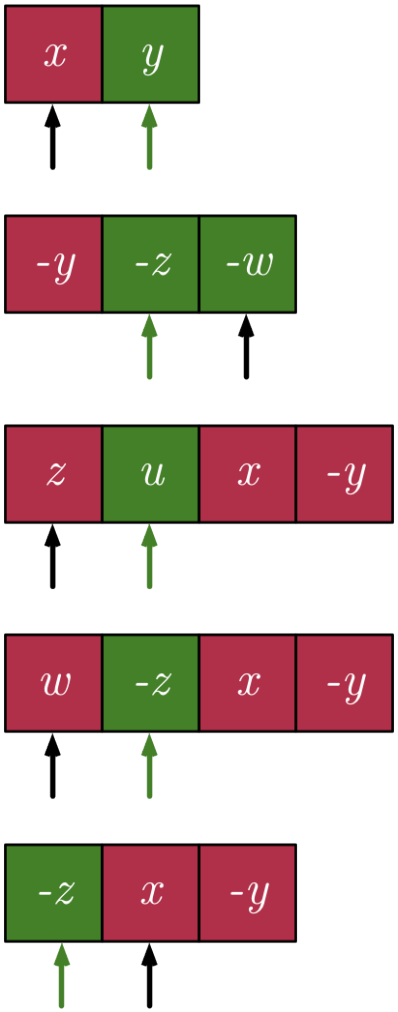


Level	Variable	Belegung	Reason
1	x	$\bot$	decision
	y	$\top$	$\{x, y\}$
	z	$\bot$	$\{\neg z, x, \neg y\}$
	u	$\top$	$\{z, u, x, \neg y\}$

- Unit Propagation: $u \mapsto \top$
- **Klausel 3:** u ist referenziert, evaluiert jedoch zu 1, d.h. kein Handlungsbedarf, Klausel ist SAT

CDCL Durchlauf mit Watched Literals

Klauselmenge: $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$



Level	Variable	Belegung	Reason
1	x	$\bot$	decision
	y	$\top$	$\{x, y\}$
	z	$\bot$	$\{\neg z, x, \neg y\}$
	u	$\top$	$\{z, u, x, \neg y\}$
2	w	$\bot$	decision

- Decision: $w \mapsto \bot$
- **Klausel 2 und 4:** w ist referenziert, ist jedoch wegen zweiter Referenz bereits SAT, kein Handlungsbedarf

Alle Variablen sind belegt und es gibt keine leere Klausel
$\Rightarrow$ **Instanz ist erfüllbar**

Fahrplan für heute: Wo stehen wir?

1. **Auswahlheuristiken für Entscheidungen** ✓
   - VSIDS
2. **Effiziente Unit Propagation** ✓
   - Lazy Datenstrukturen
   - Head/Tail Listen
   - Watched Literals
3. **Verbesserungen beim Lernen**
   - Wie minimieren wir gelernte Klauseln?
4. **Clause Deletion**
   - Wann löschen wir welche gelernten Klauseln wieder?
5. **Restart Strategien**
   - Wann starten wir den Solver neu?

Clause Minimization — Motivation

Problem: Lernen beschleunigt zwar den Solving Prozess, führt jedoch auch zu

- erhöhtem Speicherverbrauch (für gelernte Klauseln)
- unnützen Klauseln (die selten Unit werden)

Verbesserungsidee: Versuche, die gelernten Klauseln zu minimieren

- um Speicherbedarf zu verringern
- die Wahrscheinlichkeit zu erhöhen, dass Klausel Unit wird

Grundidee (Lokale Minimierung)

1. Berechne die erste entstehende 1-UIP Klausel
2. Führe dann nur noch Resolutionen aus, die die Klausel verkürzen (Self-Subsuming Resolution)

Self-Subsuming Resolution

Definition (Self-Subsuming Resolution)

Resolution zwischen zwei Klauseln $A \cup \lambda$ und $B \cup \{\neg\lambda\}$ wobei $A \subset B$. Die Resolvente ist dann B und subsumiert die Klausel $B \cup \{\neg\lambda\}$.

Beispiel

Klauseln

- $\lambda_1 = \{a, b, \neg c, \neg d, e\}$
- $\lambda_2 = \{a, b, \neg d, \neg e\}$

Resolution

- $\text{res}(\lambda_1, \lambda_2) = \{a, b, \neg c, \neg d\}$

$\text{res}(\lambda_1, \lambda_2)$ subsumiert λ_1 und damit kann λ_1 durch $\text{res}(\lambda_1, \lambda_2)$ ersetzt werden.

- In jedem Self-Subsuming Resolution Schritt wird genau das Literal, über das resolviert wird, aus der größeren Klausel entfernt.

Lokale Minimierung

Vorgehen

1. Berechne die 1-UIP Klausel
2. Führe Self-Subsuming Resolution in umgekehrter Belegungsreihenfolge der Literale aus (Resolution jeweils mit den Reasons der Belegung)

Implementierung

1. Markiere alle Literale der 1-UIP Klausel u in allen Reason Clauses
2. Lösche all jene Literale aus u, in deren Reason alle Literale markiert sind

Beispiel

- 1-UIP Klausel: $\{a, c, \neg d, \neg e\}$
- Reason für e: $\{a, c, e\}$ (alle Literale markiert, e kann gelöscht werden)
- Reason für d: $\{a, c, d\}$ (alle Literale markiert, d kann gelöscht werden)
- Reason für c: $\{a, f, \neg g, \neg c\}$ (nicht alle Literale markiert, keine Aktion)

Minimierte Klausel: $\{a, c\}$

Rekursive Minimierung — Motivation

Idee: Die gelernte Klausel darf im Laufe der Minimierung zwischenzeitlich größer werden, wenn sie nur am Ende kleiner ist

Beispiel

- 1-UIP Klausel: $\{a, b, c, d\} = \lambda_1$
- Grund für a: $\{\neg a, e, f\} = \lambda_2$
- Grund für b: $\{\neg b, e, f\} = \lambda_3$
- Grund für e: $\{c, \neg e\} = \lambda_4$
- Grund für f: $\{d, \neg f\} = \lambda_5$

Keine Self-Subsuming Resolution möglich, jedoch:

- $\text{res}(\lambda_1, \lambda_2) = \{b, c, d, e, f\}$
- Nun Self-Subsuming Resolution mit λ_3, λ_4 und λ_5 möglich
- **Minimierte Klausel:** $\{c, d\}$

Idee: Erkenne diese Situation durch Analyse des Implikationsgraphen.

Implikationsgraph — Dominanz

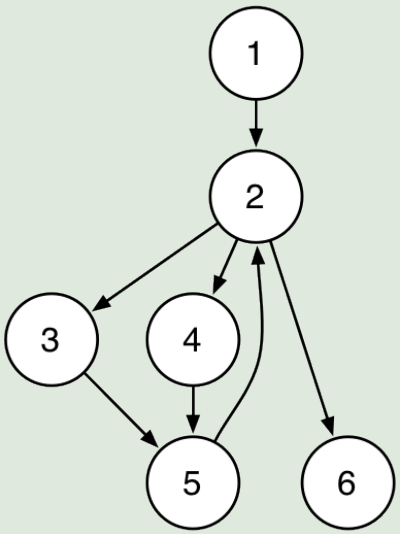
Analyse des Implikationsgraphen:

- Ein Literal kann aus der 1-UIP Klausel u entfernt werden, wenn alle Literale seiner Reason im Implikationsgraph von Literalen aus u dominiert werden.

Definition (Dominanz)

Sei $G = \langle V, E, S \rangle$ ein Graph mit ausgezeichnetem Anfangsknoten S. Für zwei Knoten $u, v \in V$ sagt man, dass Knoten u den Knoten v *dominiert*, wenn jeder Pfad, der in einem Knoten in S beginnt und in v endet den Knoten u beinhaltet.

Beispiel



- Jeder Knoten dominiert sich selber (reflexiv)
- 2 dominiert 3, 4, 5 und 6
- Kein anderer knoten dominiert einen anderen
- *Dominanz ist auch transitiv*

Rekursive Minimierung — Implementierung

Analyse des Implikationsgraphs:

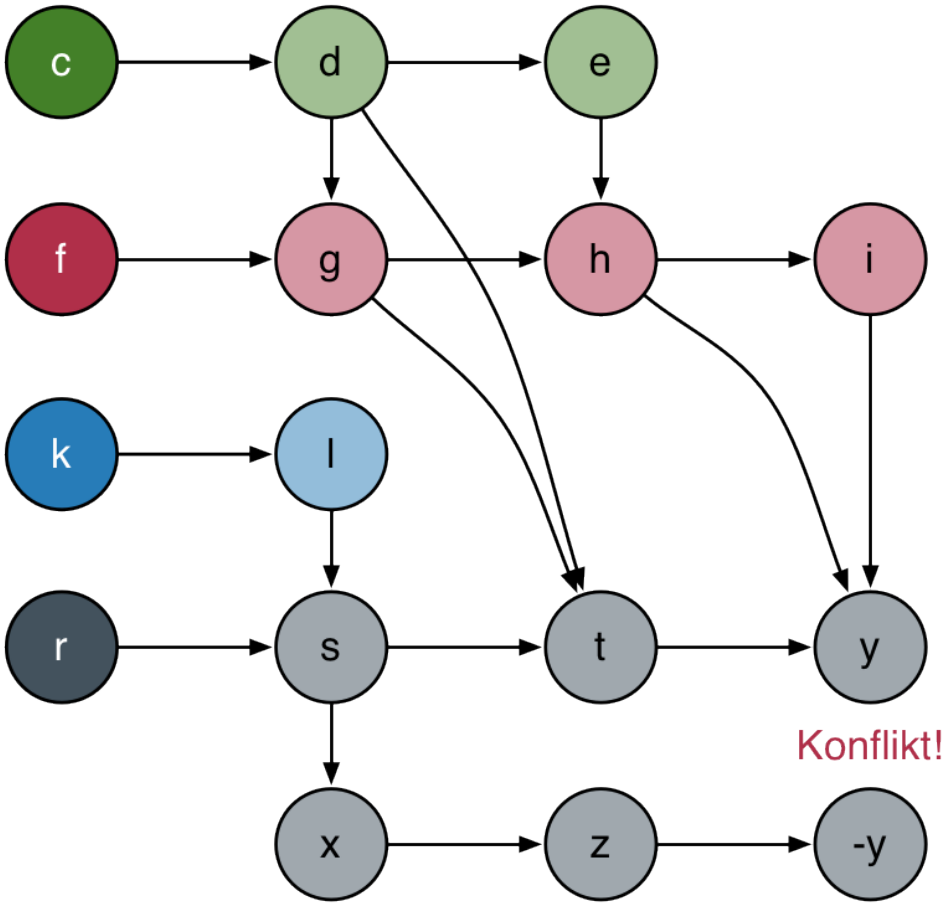
- Ein Literal kann aus der 1-UIP Klausel u entfernt werden, wenn alle Literale seiner Reason im Implikationsgraph von Literalen aus u dominiert werden.

Implementierung

1. Berechne die 1-UIP Klausel u, markiere alle ihre Literale
2. Alle implizierten Variablen λ in u sind Kandidaten zum Löschen
3. *Kann $\lambda \in u$ gelöscht werden?*
   Suche im Implikationsgraph:
   1. Starte bei den Literalen der Reason von λ
   2. Durchsuche den Implikationsgraph rückwärts
   3. Stoppe bei markierten Literalen oder Decisions
   4. Wenn jede Suche bei markierten Literalen endet, kann λ gelöscht werden

- So implementiert in z.B. MiniSAT oder Pico/PrecoSAT
- Ca. 10% mehr Instanzen aus dem SAT Race 2008 konnten mit diesen Techniken gelöst werden

Clause Minimization - Beispiel — Lokale Minimierung



Implikationsgraph für einen Konflikt

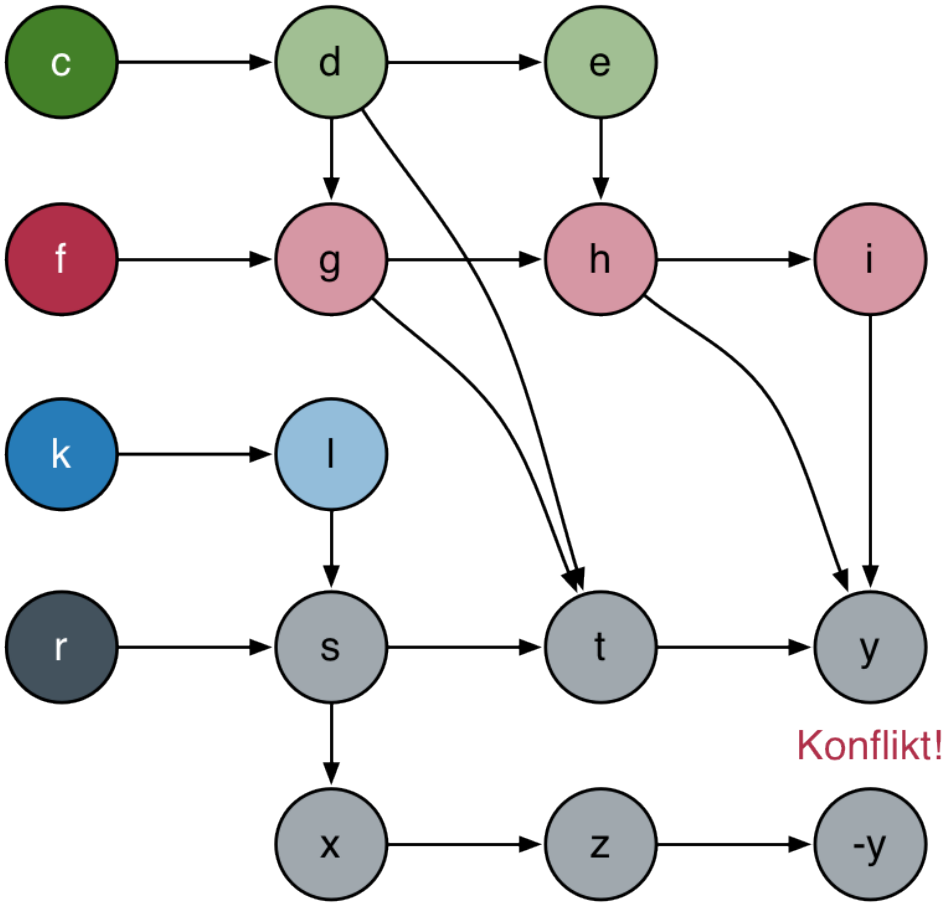
- 1-UIP Klausel:

$$\{\neg s, \neg g, \neg d, \neg h, \neg i\}$$

- Reasons:

Var	Reason
s	$\{s, \neg r, \neg l\}$
g	$\{g, \neg f, \neg d\}$
d	$\{d, \neg c\}$
h	$\{h, \neg e, \neg g\}$
i	$\{i, \neg h\}$

Clause Minimization - Beispiel — Lokale Minimierung



Implikationsgraph für einen Konflikt

- 1-UIP Klausel:

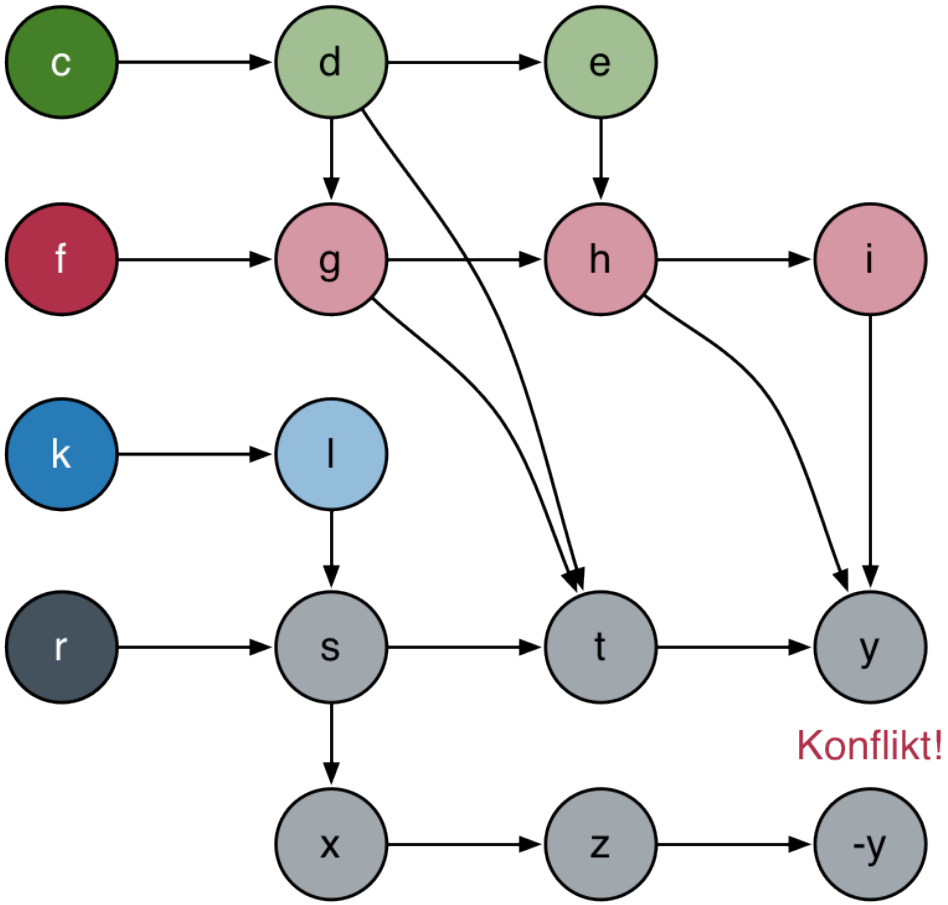
$$\{\neg s, \neg g, \neg d, \neg h, \neg i\}$$

- Reasons:

Var	Reason
s	$\{s, \neg r, \neg l\}$
g	$\{g, \neg f, \neg d\}$
d	$\{d, \neg c\}$
h	$\{h, \neg e, \neg g\}$
i	$\{i, \neg h\}$

- **Lokale Minimierung**
  - Markiere alle Variablen der 1-UIP in allen ihren reason clauses.

Clause Minimization - Beispiel — Lokale Minimierung



Implikationsgraph für einen Konflikt

- 1-UIP Klausel:

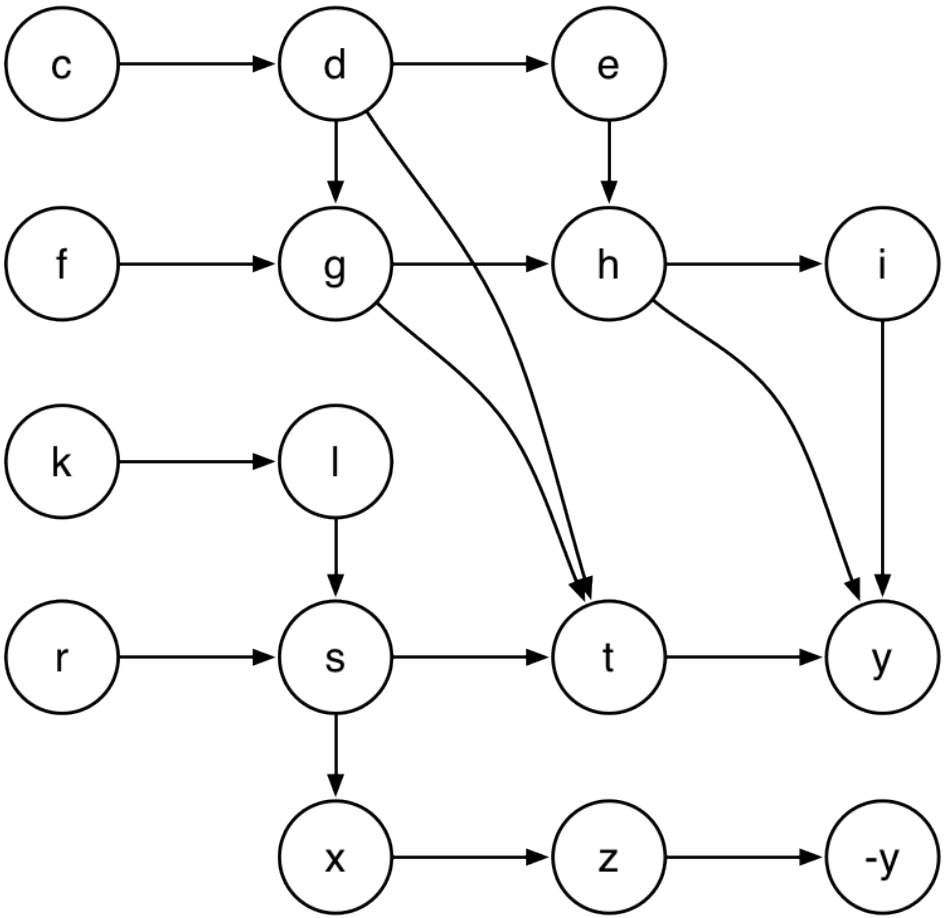
$$\{\neg s, \neg g, \neg d, \neg h\}$$

- Reasons:

Var	Reason
s	$\{s, \neg r, \neg l\}$
g	$\{g, \neg f, \neg d\}$
d	$\{d, \neg c\}$
h	$\{h, \neg e, \neg g\}$
i	$\{i, \neg h\}$

- **Lokale Minimierung**
  - Markiere alle Variablen der 1-UIP in allen ihren reason clauses.
  - Lösche jedes Literal, dessen Reason komplett markiert ist (hier: i)

Clause Minimization — Beispiel — Rekursive Minimierung

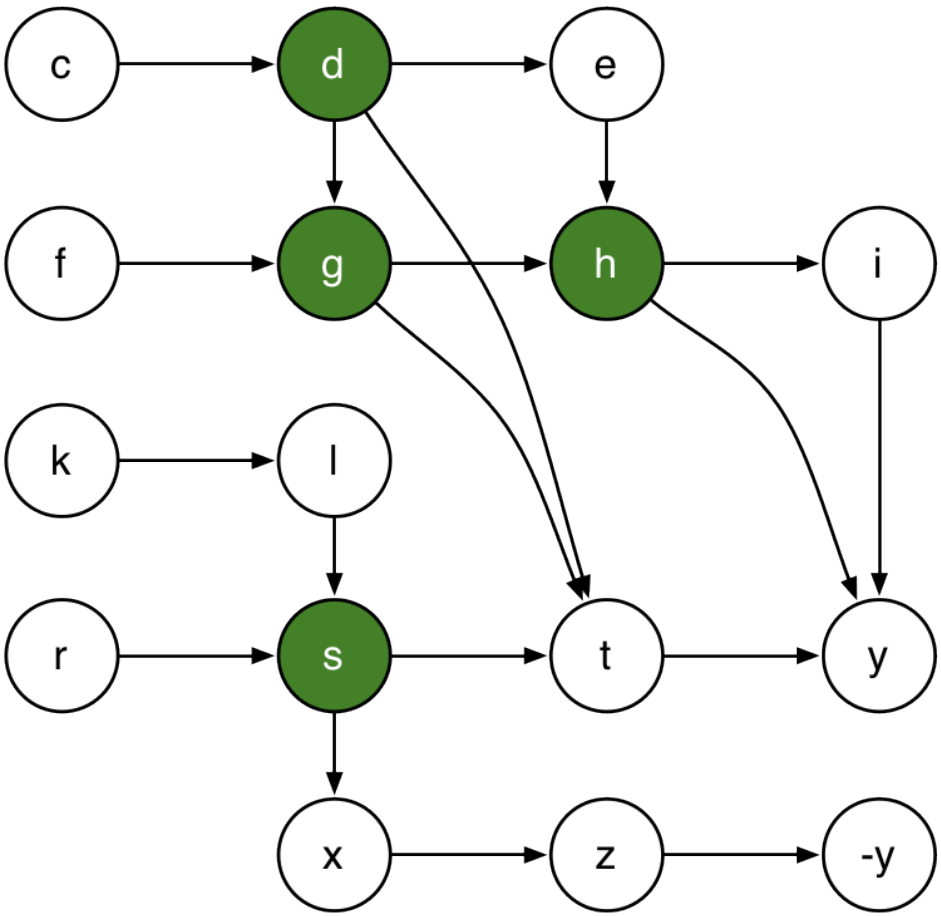


Implikationsgraph für einen Konflikt

- 1-UIP Klausel (lokal minimiert):

$$\{\neg s, \neg g, \neg d, \neg h\}$$

Clause Minimization — Beispiel — Rekursive Minimierung



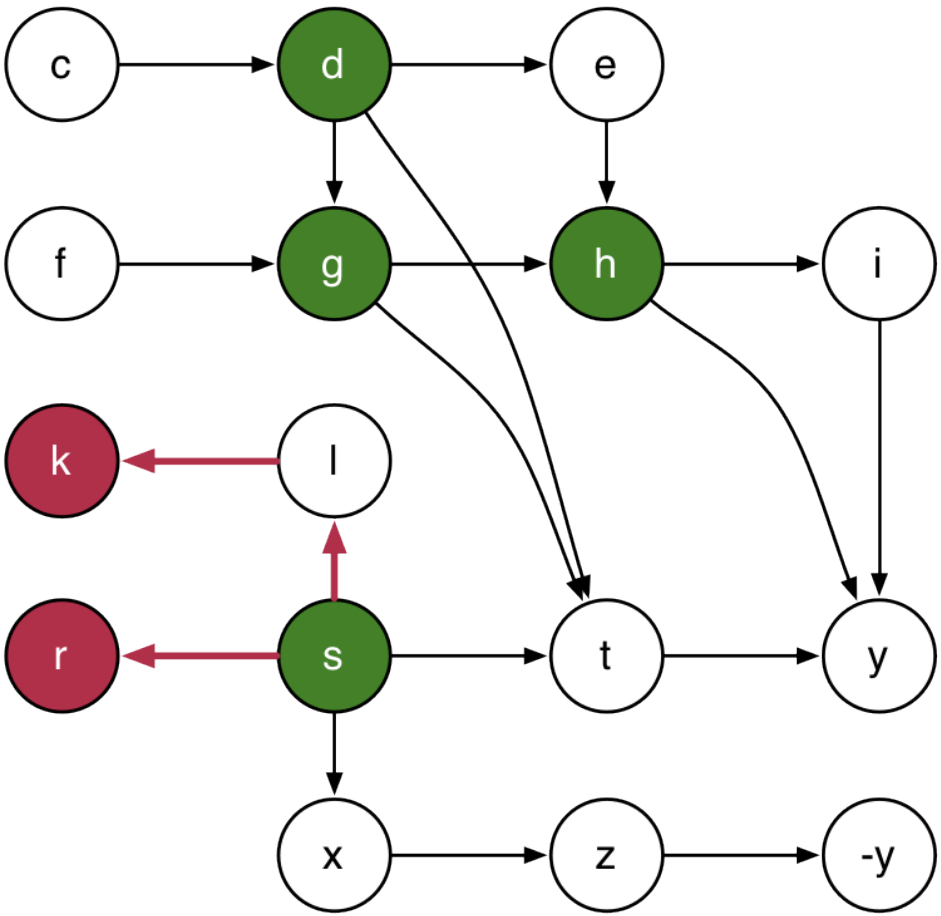
Implikationsgraph für einen Konflikt

- 1-UIP Klausel (lokal minimiert):

$$\{\neg s, \neg g, \neg d, \neg h\}$$

- **Rekursive Minimierung**
  - Markiere alle Literale der 1-UIP im Implikationsgraph.

Clause Minimization — Beispiel — Rekursive Minimierung



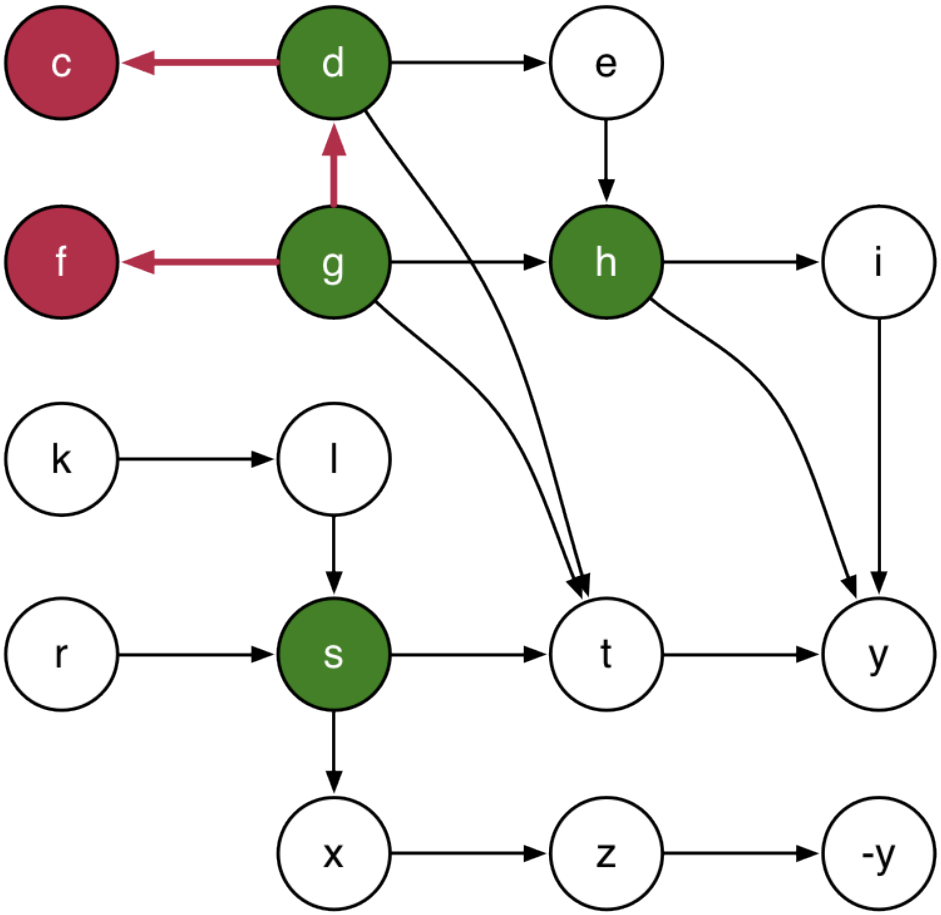
Implikationsgraph für einen Konflikt

- 1-UIP Klausel (lokal minimiert):

$$\{\neg s, \neg g, \neg d, \neg h\}$$

- **Rekursive Minimierung**
  - Markiere alle Literale der 1-UIP im Implikationsgraph.
  - *Kann s gelöscht werden?* Rückwärtssuche von s aus
  - Pfade stoppen jeweils bei Decision Variablen k und r (nicht markiert), d.h. s kann nicht gelöscht werden

Clause Minimization — Beispiel — Rekursive Minimierung



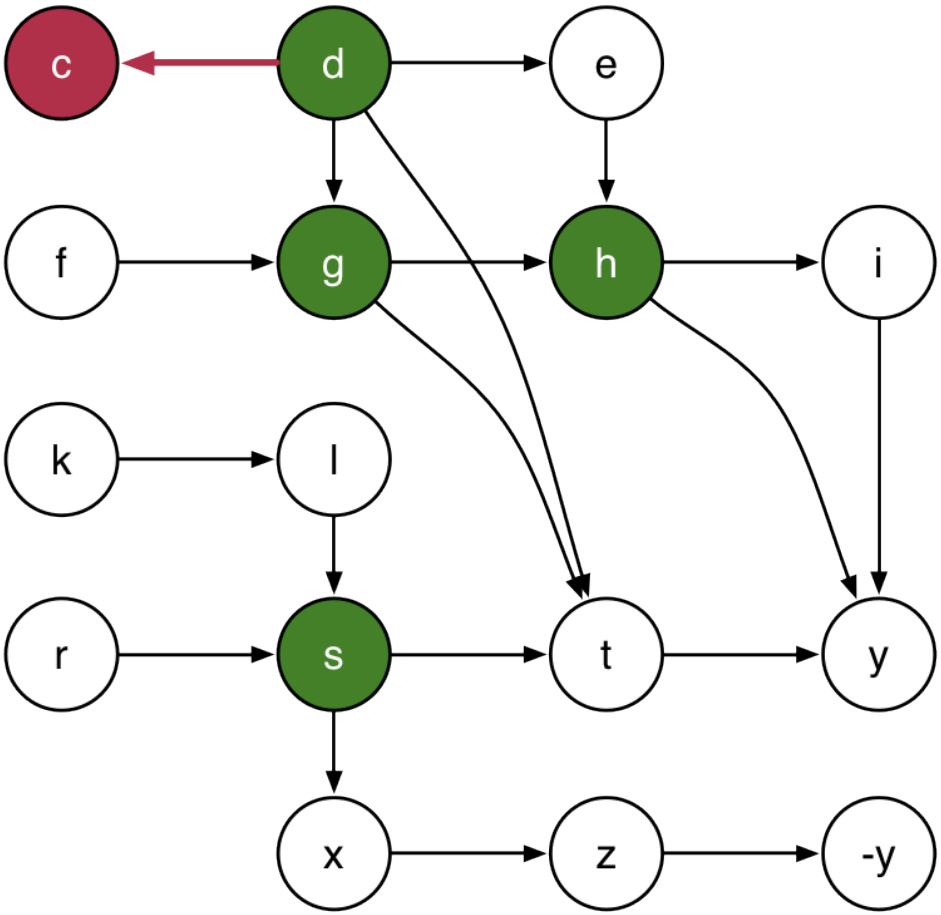
Implikationsgraph für einen Konflikt

- 1-UIP Klausel (lokal minimiert):

$$\{\neg s, \neg g, \neg d, \neg h\}$$

- **Rekursive Minimierung**
  - Markiere alle Literale der 1-UIP im Implikationsgraph
  - *Kann g gelöscht werden?* Rückwärtssuche von g aus
  - Pfade stoppen bei Decision Variable f (nicht markiert) und Variable d (markiert), d.h. g kann nicht gelöscht werden

Clause Minimization — Beispiel — Rekursive Minimierung



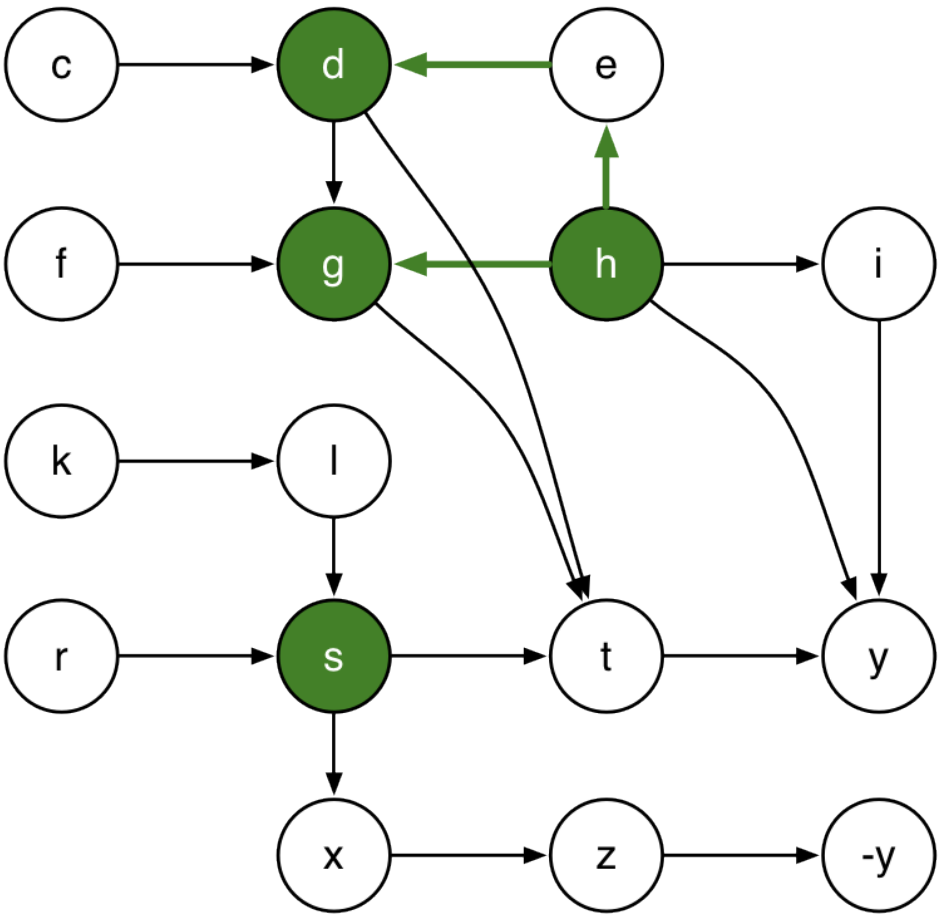
Implikationsgraph für einen Konflikt

- 1-UIP Klausel (lokal minimiert):

$$\{\neg s, \neg g, \neg d, \neg h\}$$

- **Rekursive Minimierung**
  - Markiere alle Literale der 1-UIP im Implikationsgraph
  - *Kann d gelöscht werden?* Rückwärtssuche von d aus
  - Einziger Pfad stoppt bei Decision Variablen c (nicht markiert), d.h. d kann nicht gelöscht werden

Clause Minimization — Beispiel — Rekursive Minimierung



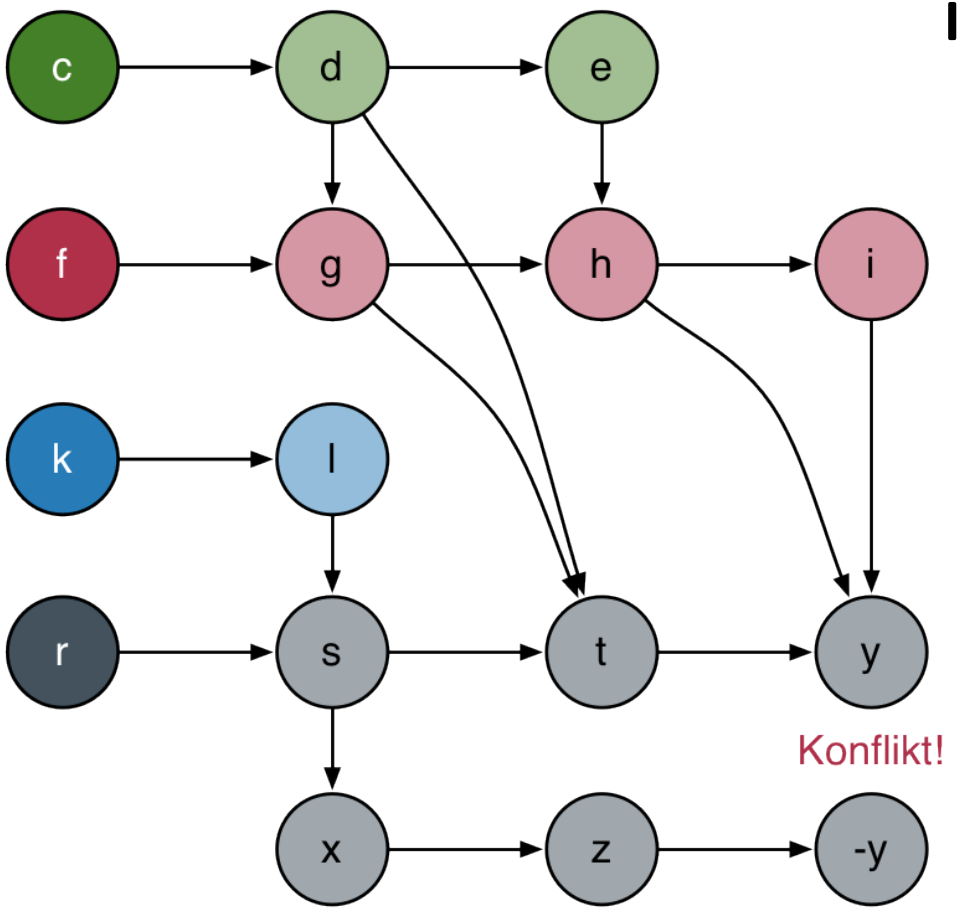
Implikationsgraph für einen Konflikt

- 1-UIP Klausel (lokal minimiert):

$$\{\neg s, \neg g, \neg d, \neg h\}$$

- **Rekursive Minimierung**
  - Markiere alle Literale der 1-UIP im Implikationsgraph
  - *Kann h gelöscht werden?* Rückwärtssuche von h aus
  - Beide möglichen Pfade stoppen bei den markierten Variablen g und d, d.h. h kann gelöscht werden

Clause Minimization — Beispiel — Rekursive Minimierung



Implikationsgraph für einen Konflikt

- Original 1-UIP:

$$\{\neg s, \neg g, \neg d, \neg h, \neg i\}$$

- + lokal minimiert:

$$\{\neg s, \neg g, \neg d, \neg h\}$$

- + rekursiv minimiert:

$$\{\neg s, \neg g, \neg d\}$$

Fahrplan für heute: Wo stehen wir?

1. **Auswahlheuristiken für Entscheidungen ✓**
   - VSIDS
2. **Effiziente Unit Propagation ✓**
   - Lazy Datenstrukturen
   - Head/Tail Listen
   - Watched Literals
3. **Verbesserungen beim Lernen ✓**
   - Lokale Minimierung
   - Rekursive Minimierung
4. **Clause Deletion**
   - Wann löschen wir welche gelernten Klauseln wieder?
5. **Restart Strategien**
   - Wann starten wir den Solver neu?

Clause Deletion

Problem: Unbegrenztes Klauselllernen kann teilweise unpraktikabel sein:

- Gelernte Klauseln verbrauchen Speicherplatz und führen möglicherweise zu vollem Speicher
- Für jeden Konflikt wird eine Klausel gelernt, Anzahl der Konflikte kann im worst-case exponentiell in der Anzahl der Variablen sein
- Große gelernte Klauseln sind meistens nicht sonderlich nützlich
- Große Klauseln führen die Suche oft in falsche Richtungen und haben damit oft höhere Kosten als Nutzen

Was man möchte: Garantieren, dass die Anzahl gelernter Klauseln polynomial in der Anzahl von Variablen ist

1. Statische Verfahren: *n-order learning*, *m-size relevance-based learning* oder *k-bounded learning*
2. Dynamische Verfahren: Aktivitätsheuristik für Klauseln

Clause Deletion — Statische Verfahren

n-order learning

Nur Klauseln mit n oder weniger Literalen werden gelernt.

m-size relevance-based learning

Klauseln werden nur temporär gelernt. Bleiben im Speicher, solange sie entweder Belegungen implizieren oder unit sind. Sobald die Anzahl unbelegter Variablen der Klausel $> m$ ist, werden sie gelöscht.

k-bounded learning

Klauseln mit weniger als k Variablen werden gespeichert. Größere Klauseln werden gelöscht, sobald die Anzahl unbelegter Variablen > 1 ist, d.h. sie nicht mehr unit sind.

Kombination von k-bounded learning und m-size relevance-based learning:

- Klauseln kleiner als k werden immer gespeichert
- Klauseln größer als k werden gelöscht, wenn mehr als m Variablen unbelegt sind

Clause Deletion — Dynamische Verfahren

Beobachtung:

- Viele der gelernten Klauseln werden im folgenden Solving Prozess nicht mehr benutzt

$\rightarrow$ Klauselmenge bläht sich unnötig auf

Ansatz:

- Bewerte neu gelernte Klauseln (wie Variablen) ebenfalls mit Aktivität
- Initiale Aktivität einer neu gelernten Klausel ist eine Konstante c
- Jedesmal, wenn die Klausel in der Berechnung einer neuen Klausel vorkommt (d.h. in der Resolution benutzt wird), wird die Aktivität erhöht
- Periodisch werden alle Aktivitäten verringert
- Periodisch werden alle Klauseln mit geringer Aktivität (unter einem vorher bestimmten Schwellenwert) gelöscht

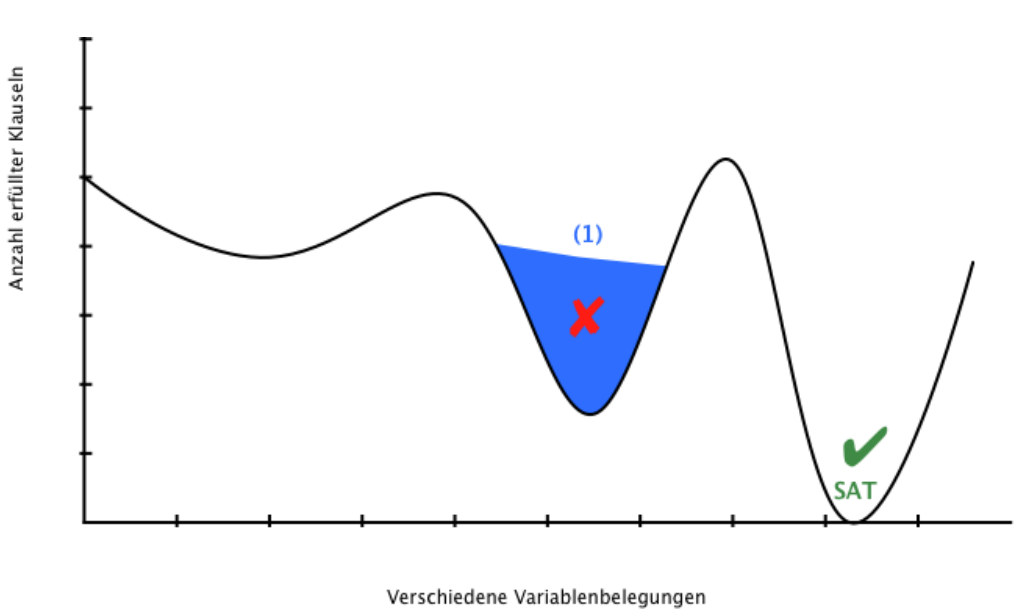
Effekt:

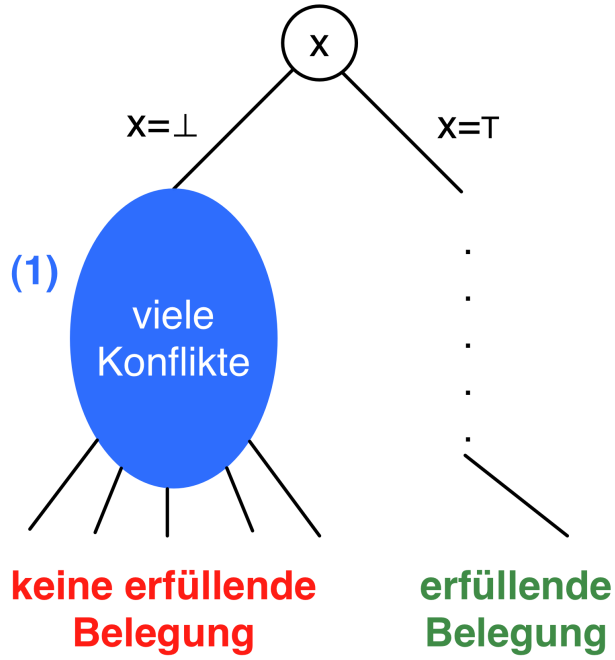
- Wird eine neu gelernte Klausel nie oder selten benutzt, wird sie nach einer gewissen Zeit wieder gelöscht

Fahrplan für heute: Wo stehen wir?

1. **Auswahlheuristiken für Entscheidungen** ✓
   - VSIDS
2. **Effiziente Unit Propagation** ✓
   - Lazy Datenstrukturen
   - Head/Tail Listen
   - Watched Literals
3. **Verbesserungen beim Lernen** ✓
   - Lokale Minimierung
   - Rekursive Minimierung
4. **Clause Deletion** ✓
   - Statisch: n-order, m-size relevance-based, k-bounded
   - Dynamisch: Aktivitätsheuristik
5. **Restart Strategien**
   - Wann starten wir den Solver neu?

Gefahr von lokalen Minima





Lokale Minima

Mit VSIDS besteht die Gefahr des „Festfressens“ in einem lokalen Minimum (1):

- Heuristik wählt immer aktuelle Konfliktvariablen aus, jedoch besteht ein größerer Konflikt ganz am Anfang des Suchbaums
- Bei einer großen Anzahl von Variablen kann ein Verlassen des lokalen Minimums sehr lange dauern

Lösung für lokale Minima: Restarts

Restart

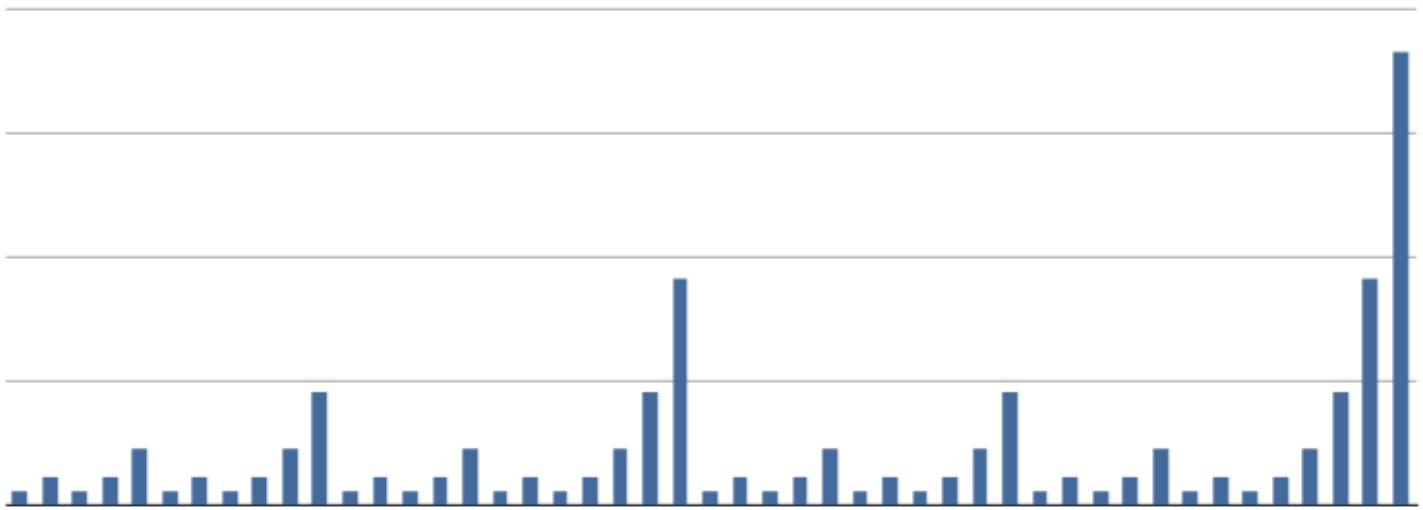
Der SAT Solver wird nach einer bestimmten Anzahl an Konflikten zurückgesetzt und neu gestartet, d.h.

- Alle Variablenbelegungen werden rückgängig gemacht
- Alle Aktivitäten von Variablen und Klauseln werden auf initialen Wert gesetzt
- **Aber:** Gelernte Klauseln bleiben erhalten (möglicherweise finden jedoch Clause Deletions statt)

- **Problem:** Durch Restarts kann CDCL unvollständig werden.
- **Ansatz:** Anzahl der Konflikte nach denen neu gestartet wird, wird mit jedem Restart erhöht
  1. konstanter Faktor: 50,100,200,400,800,...
  2. Luby Sequenz: 20, 40, 20, 40, 80, 20, 40, 20, 40, 80, 160, ...

Luby Sequenzen

- Sequenz wächst nicht konstant an, sondern wird immer wieder zurückgesetzt
- Nach jedem Zurücksetzten läuft die Sequenz einen Schritt mehr



Auf realen Instanzen zeigen Restarts mit Luby Sequenz aktuell die besten Ergebnisse

Fahrplan für heute: Wo stehen wir?

1. **Auswahlheuristiken für Entscheidungen ✓**
   - VSIDS
2. **Effiziente Unit Propagation ✓**
   - Lazy Datenstrukturen
   - Head/Tail Listen
   - Watched Literals
3. **Verbesserungen beim Lernen ✓**
   - Lokale Minimierung
   - Rekursive Minimierung
4. **Clause Deletion ✓**
   - Statisch: n-order, m-size relevance-based, k-bounded
   - Dynamisch: Aktivitätsheuristik
5. **Restart Strategien ✓**
   - Luby Sequenz

Zusammenfassung

Alles, was man für einen effizienten state-of-the-art SAT Solver braucht (so zu finden in z.B. MiniSAT):

1. Klauselllernen
2. Nicht-chronologisches Backtracking
3. Gute Auswahlheuristiken (z.B. VSIDS)
4. Effiziente Unit Propagation (watched literals)
5. Restarts mit Clause Deletions

Fazit:

- Im Gegensatz zu anderen Problemen (z.B. lineare Optimierung mit Simplex) kann SAT Solving auf wenige Kernpunkte reduziert werden
- Einer der besten aktuellen SAT Solver – MiniSAT – kommt mit < 1000 Zeilen Code aus