

# SAT-Solving und Anwendungen

## Implementierung effizienter SAT Solver

Prof. Dr. Wolfgang Küchlin  
Dipl. Inf. Christoph Zengler

Universität Tübingen

17. Mai 2011



# Fahrplan für heute

## ① Effiziente Unit Propagation

- Wie bekommen wir Unit Propagation effizient?

## ② Verbesserungen beim Lernen

- Wie minimieren wir gelernte Klauseln?

## ③ Clause Deletion

- Wann löschen wir welche gelernten Klauseln wieder?

## ④ 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 muss so sein: Der exponentiell wachsende Lösungsraum wird durch die Zwänge eingeschränkt, die die UP ermöglichen.
- UP ist der Schlüssel zu effizienten SAT Solvern

## Naive Unit Propagation (wie in der zweiten Übung)

- **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

**Idee:** Man muss eine Klausel nicht bei jeder Variablenbelegung neu analysieren  
Für jede Klausel sind nur zwei Zustände interessant: (1) Unit und (2) Leer

Dazu genügt es, 2 Literale pro Klausel zu „beobachten“ / referenzieren

- Es werden niemals Literale referenziert, die zu 0 evaluieren
- Sobald ein referenziertes Literal so belegt wird, dass es zu 0 evaluiert, wird die Referenz in der Klausel weiter bewegt

---

## Algorithm 1: clauseStatus( $\omega$ )

---

**if** REFA( $\omega$ ) == 1 *or* REFB( $\omega$ ) == 1 **then**

**return** SATISFIED

**else**

**if** REFA( $\omega$ ) == 0 *and* POSITION\_REFA( $\omega$ ) == POSITION\_REFB( $\omega$ ) **then**

**return** EMPTY

**else**

**if** REFA( $\omega$ ) ==  $u$  *and* POSITION\_REFA( $\omega$ ) == POSITION\_REFB( $\omega$ ) **then**

**return** UNIT

**else**

**return** UNRESOLVED

---

# Head/Tail Datenstruktur

- Erstmals 1997 im SAT Solver *Sato* benutzt

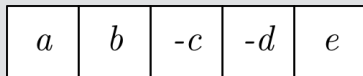
## Vorgehen

- 1 Mit jeder Klausel werden zwei Referenzen assoziiert:
  - Head  $H$  — initial auf dem ersten Literal der Klausel
  - Tail  $T$  — initial auf dem letzten Literal der Klausel
- 2 Wird eine referenzierte Variable belegt und evaluiert zu 0, so wird die entsprechende Referenz nach innen bewegt.
  - 1 Wird ein unbelegtes Literal gefunden, so wird **eine neue** Referenz auf dieses gesetzt (Die alte Referenz wird jedoch wegen Backtracking nicht gelöscht)
  - 2 Wird ein erfülltes Literal gefunden, so ist die Klausel SAT
  - 3 Wird die zweite Referenz erreicht ohne ein unbelegtes Literal gefunden zu haben, so ist die Klausel (je nach Evaluation der anderen Referenz) UNIT, SAT oder UNSAT

# Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

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

## Erkennen einer erfüllten Klausel



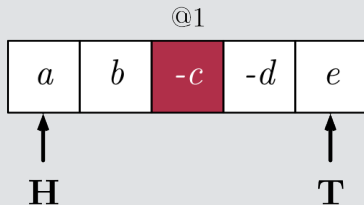
Level	Variable	Belegung
-------	----------	----------

- 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



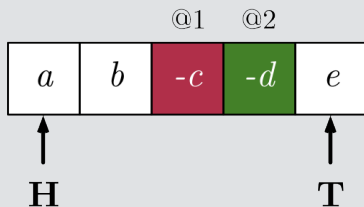
Level	Variable	Belegung
1	$c$	$\top$

- 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	$d$	$\perp$

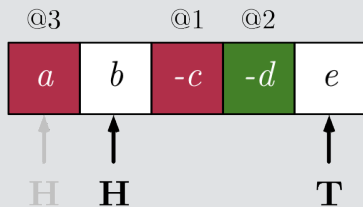
- Variable  $d$  wird belegt
- Da  $d$  nicht referenziert ist, wird die Klausel nicht analysiert
- Klausel ist bereits SAT, was wir jedoch noch nicht wissen (lazy)



# Head/Tail Datenstruktur — Beispiel 1 (erfüllte Klausel)

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

## Erkennen einer erfüllten Klausel



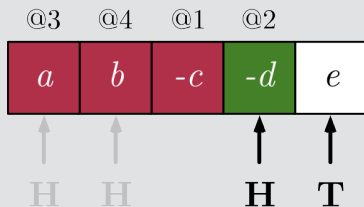
Level	Variable	Belegung
1	$c$	$\top$
2	$d$	$\perp$
3	$a$	$\perp$

- 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



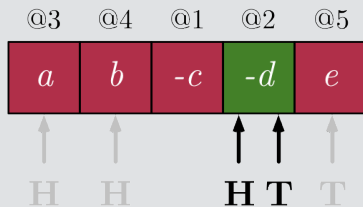
Level	Variable	Belegung
1	$c$	$\top$
2	$d$	$\perp$
3	$a$	$\perp$
4	$b$	$\perp$

- 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



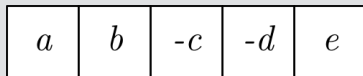
Level	Variable	Belegung
1	$c$	$\top$
2	$d$	$\perp$
3	$a$	$\perp$
4	$b$	$\perp$
5	$e$	$\perp$

- 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



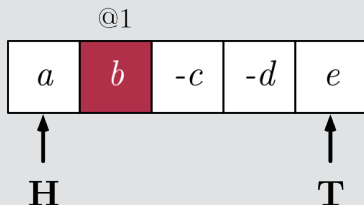
Level	Variable	Belegung
-------	----------	----------

- 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



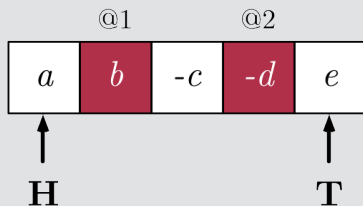
Level	Variable	Belegung
1	$b$	$\perp$

- 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



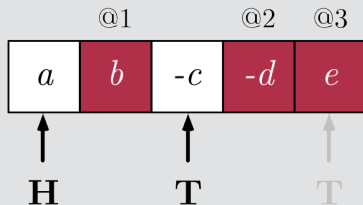
Level	Variable	Belegung
1	$b$	$\perp$
2	$d$	$\top$

- 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



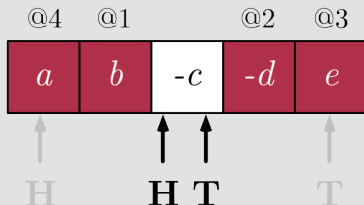
Level	Variable	Belegung
1	$b$	$\perp$
2	$d$	$\top$
3	$e$	$\perp$

- 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	$b$	$\perp$
2	$d$	$\top$
3	$e$	$\perp$
4	$a$	$\perp$

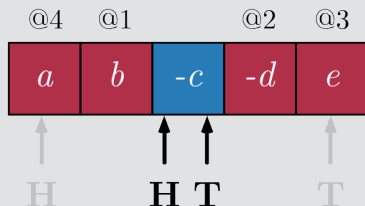
- Variable  $a$  wird belegt
- $a$  ist referenziert und evaluiert zu 0, daher wird die Referenz nach innen bewegt
- Alte Referenz wird gespeichert (für Backtracking)
- Tail Referenz wird gefunden



# Head/Tail Datenstruktur — Beispiel 2 (Unit Klausel)

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

## Erkennen einer Unit Klausel

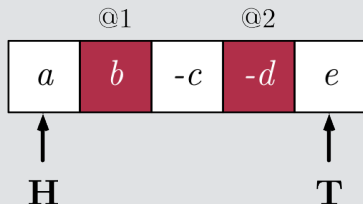


Level	Variable	Belegung
1	$b$	$\perp$
2	$d$	$\top$
3	$e$	$\perp$
4	$a$	$\perp$

- 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	$b$	$\perp$
2	$d$	$\top$

- Bei Backtracking zu Level 2 müssen auch die Head und Tail Referenzen wiederhergestellt werden

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

# Watched Literals

- **Idee:** Verzichte auf Ordnung ( $\text{Head} < \text{Tail}$ ) zwischen den beiden Referenzen und erlaube das Bewegen der Referenzen in beide Richtungen
- Dadurch „zustandslose“ Referenzen
- Erstmals 2001 im SAT Solver *Chaff* benutzt

## Vorgehen

- ① Mit jeder Klausel werden zwei Referenzen (*watched literals*) assoziiert
- ② Wird eine referenzierte Variable belegt und evaluiert zu 0, so wird die entsprechende Referenz bewegt (in beliebige Richtung, auch über Klauselgrenze hinaus).

**Vorteil:** Die Anzahl von Referenzen pro Klausel ist konstant 2 (nach Backtracking können die aktuellen Referenzen beibehalten werden)

**Nachteil:** Um festzustellen, dass eine Klausel UNIT oder UNSAT ist, müssen alle Literale der Klausel durchgegangen werden

# Watched Literals — Beispiel (Unit Klausel)

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

## Erkennen einer Unit Klausel

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



$W_1$   $W_2$

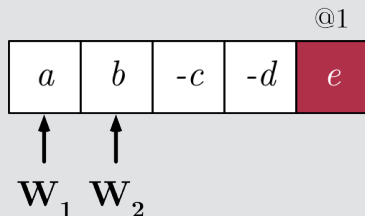
Level	Variable	Belegung
-------	----------	----------

- 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



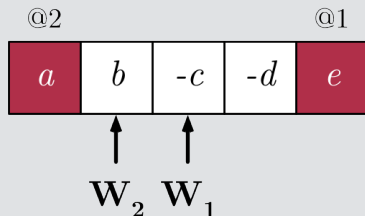
Level	Variable	Belegung
1	$e$	$\perp$

- 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



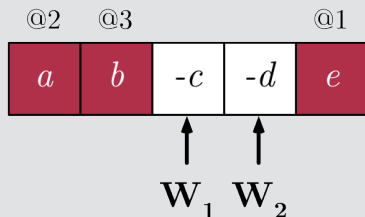
Level	Variable	Belegung
1	$e$	$\perp$
2	$a$	$\perp$

- 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



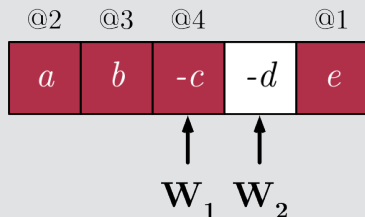
Level	Variable	Belegung
1	$e$	$\perp$
2	$a$	$\perp$
3	$b$	$\perp$

- 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	$e$	$\perp$
2	$a$	$\perp$
3	$b$	$\perp$
4	$c$	$\top$

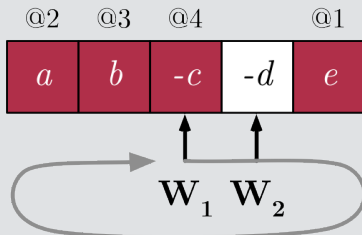
- Variable  $c$  wird belegt
- $c$  ist referenziert und evaluiert zu 0, daher wird die Referenz bewegt



# Watched Literals — Beispiel (Unit Klausel)

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

## Erkennen einer Unit Klausel



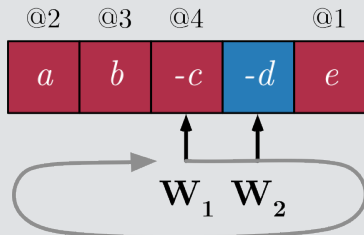
Level	Variable	Belegung
1	$e$	$\perp$
2	$a$	$\perp$
3	$b$	$\perp$
4	$c$	$\top$

- 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



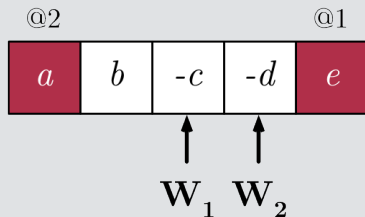
Level	Variable	Belegung
1	$e$	$\perp$
2	$a$	$\perp$
3	$b$	$\perp$
4	$c$	$\top$

- 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	$e$	$\perp$
2	$a$	$\perp$

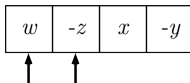
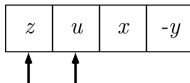
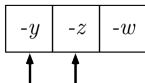
- Bei Backtracking (z.B. zu Level 2) können die Watched Literals beibehalten werden
- unbelegte Watched Literals bleiben unbelegt
- Keine Annahmen über Belegungsstatus links und rechts der Referenzen

# CDCL Durchlauf mit Watched Literals

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

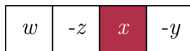
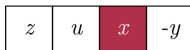
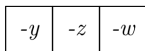
Level	Variable	Belegung	Reason
-------	----------	----------	--------

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



# CDCL Durchlauf mit Watched Literals

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

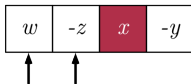
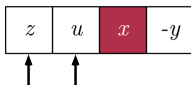
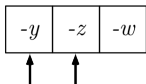


Level	Variable	Belegung	Reason
1	$x$	$\perp$	decision

- Decision:  $x \mapsto \perp$
- 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:  $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

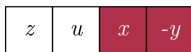


Level	Variable	Belegung	Reason
1	$x$	$\perp$	decision

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

# CDCL Durchlauf mit Watched Literals

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

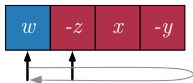
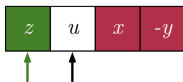
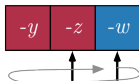


Level	Variable	Belegung	Reason
1	x	$\perp$	decision
	y	$\top$	$\{x, y\}$

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



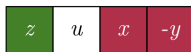
Level	Variable	Belegung	Reason
1	$x$	$\perp$	decision
	$y$	$\top$	$\{x, y\}$
2	$z$	$\top$	decision

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

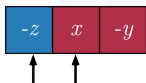
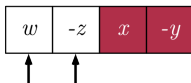
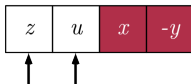
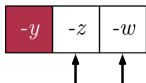


Level	Variable	Belegung	Reason
1	x	$\perp$	decision
	y	$\top$	$\{x, y\}$
2	z	$\top$	decision
	w	$\perp$	$\{\neg y, \neg z, \neg w\}$
	w	$\top$	$\{w, \neg z, x, \neg y\}$

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

# CDCL Durchlauf mit Watched Literals

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

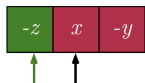
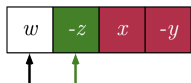
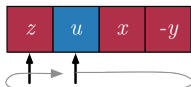
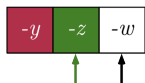


Level	Variable	Belegung	Reason
1	x	$\perp$	decision
	y	$\top$	$\{x, y\}$

- **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	$x$	$\perp$	decision
	$y$	$\top$	$\{x, y\}$
	$z$	$\perp$	$\{\neg z, x, \neg y\}$

- Unit Propagation:  $z \mapsto \perp$
- **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:  $\{\{x, y\}, \{\neg y, \neg z, \neg w\}, \{z, u, x, \neg y\}, \{w, \neg z, x, \neg y\}\}$

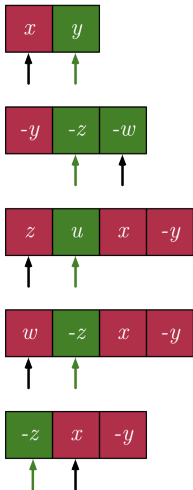


Level	Variable	Belegung	Reason
1	$x$	$\perp$	decision
	$y$	$\top$	$\{x, y\}$
	$z$	$\perp$	$\{\neg z, x, \neg y\}$
	$u$	$\top$	$\{z, u, x, \neg y\}$
			$\{w, \neg z, x, \neg y\}$

- 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	$x$	$\perp$	decision
	$y$	$\top$	$\{x, y\}$
	$z$	$\perp$	$\{\neg z, x, \neg y\}$
	$u$	$\top$	$\{z, u, x, \neg y\}$
2	$w$	$\perp$	decision

- Decision:  $w \mapsto \perp$
- **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 Effiziente Unit Propagation ✓

- Lazy Datenstrukturen
- Head/Tail Listen
- Watched Literals

## ② Verbesserungen beim Lernen

- Wie minimieren wir gelernte Klauseln?

## ③ Clause Deletion

- Wann löschen wir welche gelernten Klauseln wieder?

## ④ 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 subsumiert dann 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  $\lambda_1$  und damit kann  $\lambda_1$  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  $u$
- 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  $e$ :  $\{a, \neg c, e\}$  (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  $\lambda_3$ ,  $\lambda_4$  und  $\lambda_5$  möglich
- **Minimierte Klausel:**  $\{c, d\}$

**Idee:** Erkenne diese Situation durch Analyse des Implikationsgraphen.

# Implikationsgraph — Dominanz

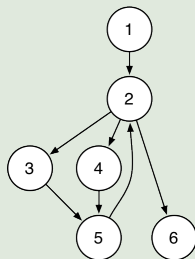
## Analyse des Implikationsgraphen:

- 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.

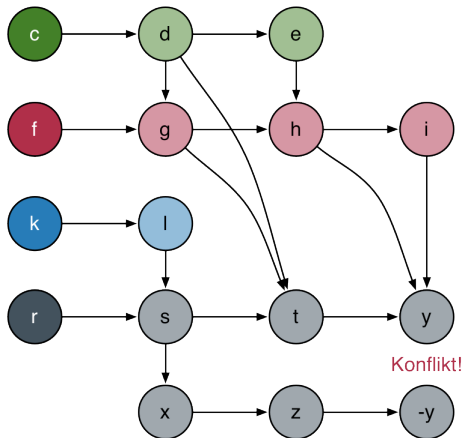
## Implementierung

- ① Berechne die 1-UIP Klausel  $u$ , markiere alle ihre Literale
- ② Alle implizierten Variablen  $\lambda$  in  $u$  sind Kandidaten zum Löschen
- ③ *Kann  $\lambda \in u$  gelöscht werden?*

### Suche im Implikationsgraph:

- ① Starte bei den Literalen der Reason von  $\lambda$
  - ② Durchsuche den Implikationsgraph rückwärts
  - ③ Stoppe bei markierten Literalen oder Decisions
  - ④ Wenn jede Suche bei markierten Literalen endet, kann  $\lambda$  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

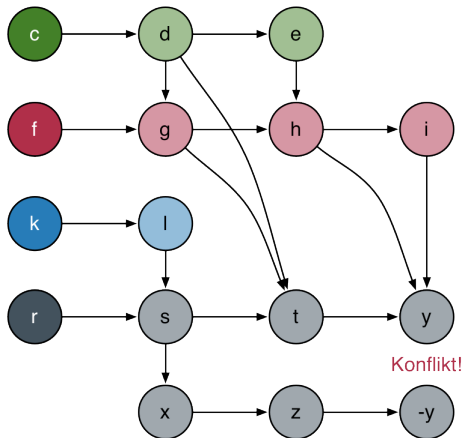
- 1-UIP Klausel:

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

- Reasons:

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

# Clause Minimization - Beispiel — Lokale Minimierung



## Implikationsgraph für einen Konflikt

- 1-UIP Klausel:

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

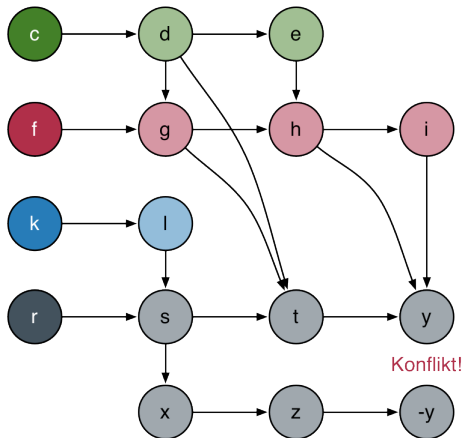
- Reasons:

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

- Lokale Minimierung**

- Markiere alle Variablen in der 1-UIP

# Clause Minimization - Beispiel — Lokale Minimierung



## Implikationsgraph für einen Konflikt

- 1-UIP Klausel:

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

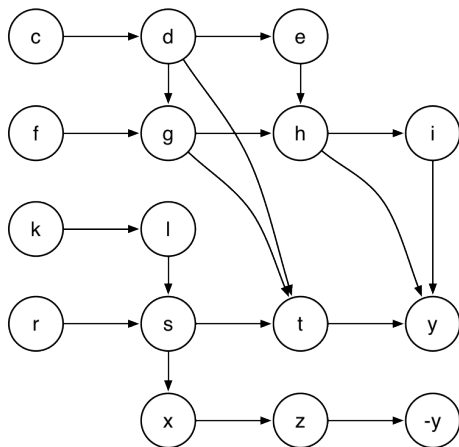
- Reasons:

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

- Lokale Minimierung**

- Markiere alle Variablen in der 1-UIP
- Lösche jedes Literal, deren Reason komplett markiert ist (hier: *i*)

# Clause Minimization — Beispiel — Rekursive Minimierung



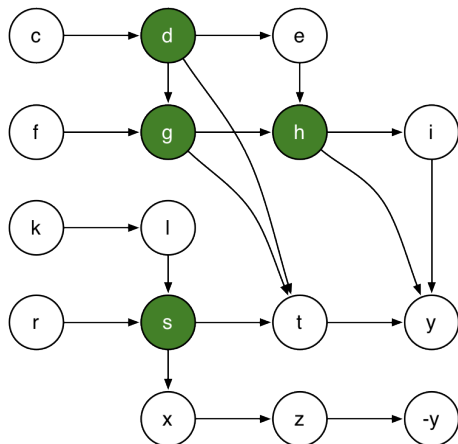
## Implikationsgraph für einen Konflikt

- 1-UIP Klausel (lokal minimiert):

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



# Clause Minimization — Beispiel — Rekursive Minimierung



## Implikationsgraph für einen Konflikt

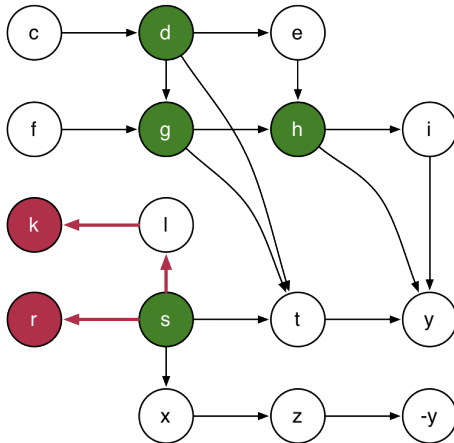
- 1-UIP Klausel (lokal minimiert):

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

- **Rekursive Minimierung**

- Markiere alle Literale der 1-UIP im Implikationsgraph

# Clause Minimization — Beispiel — Rekursive Minimierung



## Implikationsgraph für einen Konflikt

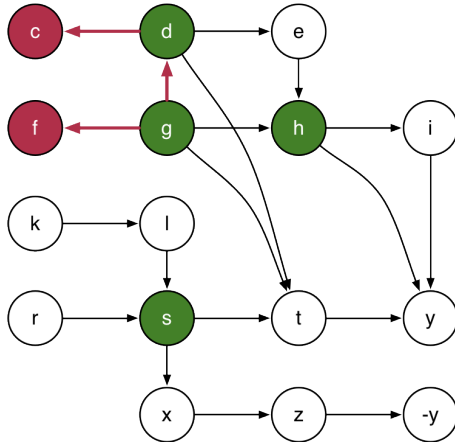
- 1-UIP Klausel (lokal minimiert):

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

- **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 e (nicht markiert), d.h. s kann nicht gelöscht werden

# Clause Minimization — Beispiel — Rekursive Minimierung



## Implikationsgraph für einen Konflikt

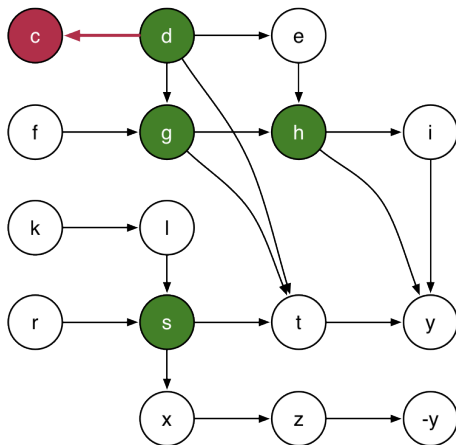
- 1-UIP Klausel (lokal minimiert):

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

- **Rekursive Minimierung**

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

# Clause Minimization — Beispiel — Rekursive Minimierung



## Implikationsgraph für einen Konflikt

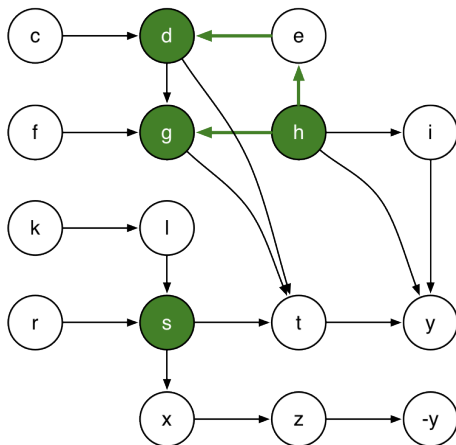
- 1-UIP Klausel (lokal minimiert):

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

- **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

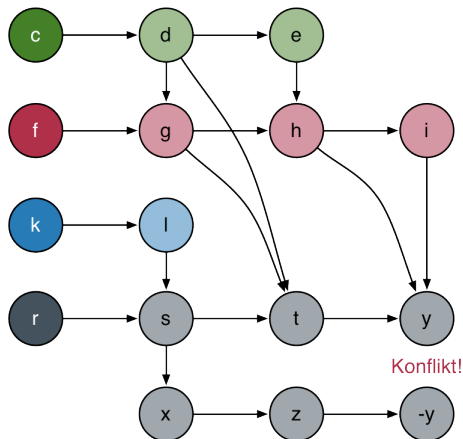
- 1-UIP Klausel (lokal minimiert):

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

- **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 Effiziente Unit Propagation ✓

- Lazy Datenstrukturen
- Head/Tail Listen
- Watched Literals

## ② Verbesserungen beim Lernen ✓

- Lokale Minimierung
- Rekursive Minimierung

## ③ Clause Deletion

- Wann löschen wir welche gelernten Klauseln wieder?

## ④ Restart Strategien

- Wann starten wir den Solver neu?

# Clause Deletion

**Problem:** Unbegrenztes Klausellernen 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
- Klauselmengen 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 Effiziente Unit Propagation ✓

- Lazy Datenstrukturen
- Head/Tail Listen
- Watched Literals

## ② Verbesserungen beim Lernen ✓

- Lokale Minimierung
- Rekursive Minimierung

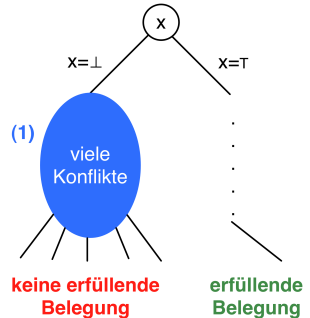
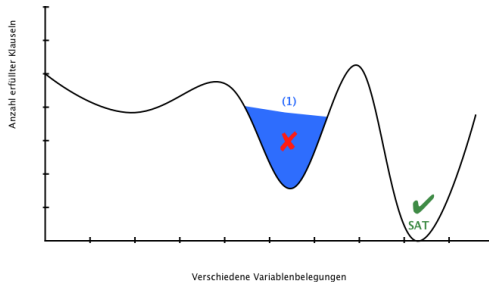
## ③ Clause Deletion ✓

- Statisch:  $n$ -order,  $m$ -size relevance-based,  $k$ -bounded
- Dynamisch: Aktivitätsheuristik

## ④ 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

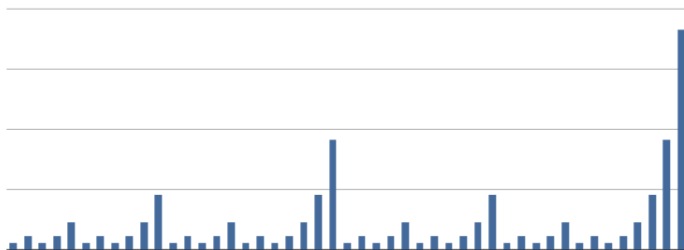
## Restart

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 Zurücksetzen läuft die Sequenz einen Schritt mehr



# Fahrplan für heute: Wo stehen wir?

## ① Effiziente Unit Propagation ✓

- Lazy Datenstrukturen
- Head/Tail Listen
- Watched Literals

## ② Verbesserungen beim Lernen ✓

- Lokale Minimierung
- Rekursive Minimierung

## ③ Clause Deletion ✓

- Statisch:  $n$ -order,  $m$ -size relevance-based,  $k$ -bounded
- Dynamisch: Aktivitätsheuristik

## ④ Restart Strategien ✓

- Luby Sequenz