MiniSAT

SAT-Solving

Vortrag gehalten von
Andreas Weis

im Rahmen des Seminars
Perlen der Informatik IV, SS2006
TU München

Mail: weisa@in.tum.de

MiniSAT

Erfüllbarkeit boolescher Gleichungen (Satisfiability, kurz: *SAT*)

Verknüpfung von Variablen durch Boolesche Operationen (AND, OR, NOT, ...)

Ziel: Gibt es eine Belegung, die die Gleichung erfüllt? Wenn ja, welche?

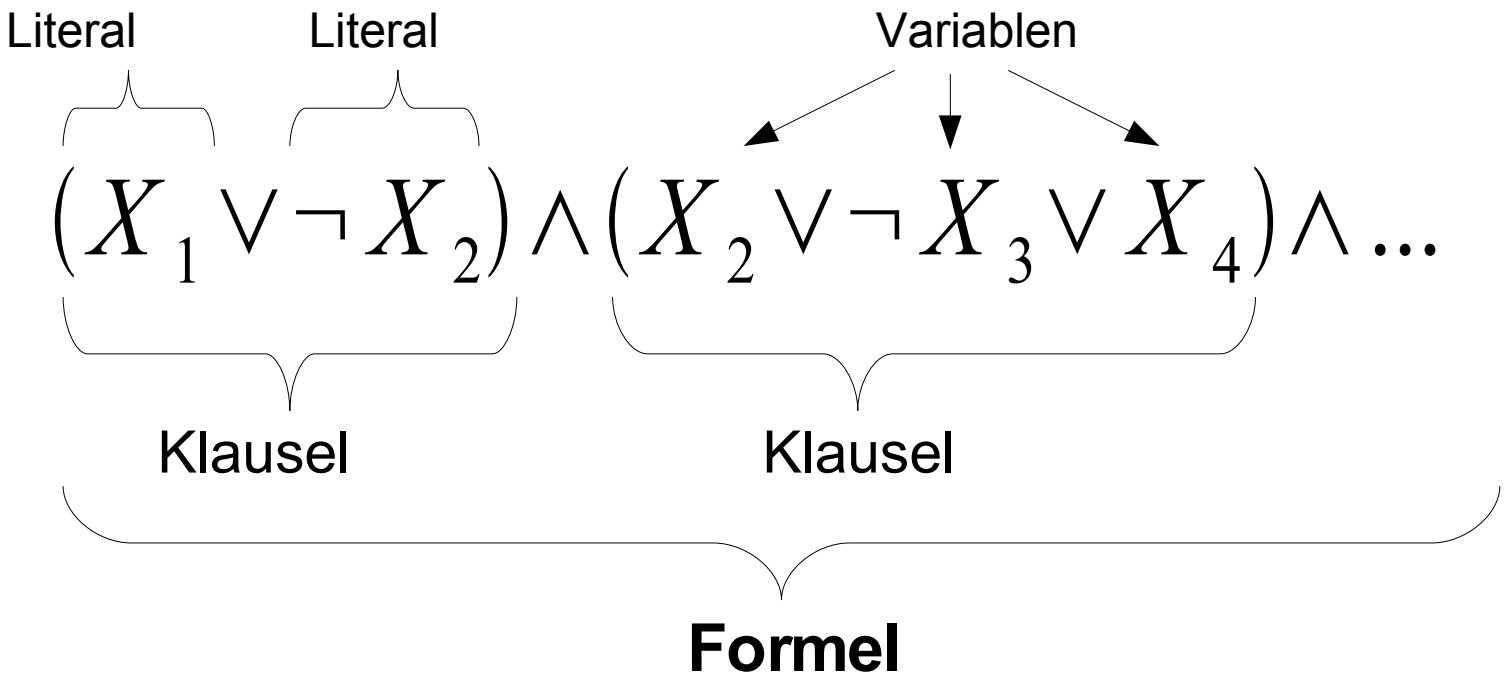
- NP-vollständig => wahrscheinlich nicht effizient lösbar
- Größe des Suchraums wächst exponentiell mit der Zahl der Variablen (2^n)
- Gute Average-Case-Laufzeit nur durch starke Heuristiken möglich

MiniSAT

Spezifikation des Problems

Eingabe: Boolesche Formel in *konjunktiver Normalform* (CNF)

Konjunktion von *Klauseln*, die ihrerseits Disjunktionen von *Literalen* sind:

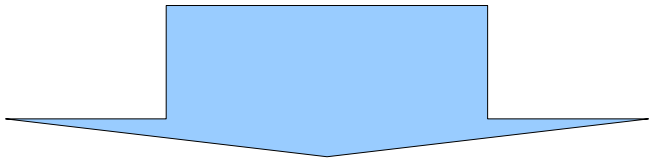


MiniSAT

$$(X_1 \vee \neg X_2) \wedge (X_2 \vee \neg X_3 \vee X_4) \wedge \ldots$$

Eigenschaften dieser Form der Darstellung:

- Wertet eine Klausel zu FALSE aus, so wird automatisch die gesamte Formel FALSE
- Pro Klausel muss immer mindestens ein Literal zu TRUE auswerten



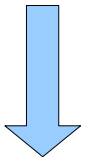
Boolean Constraint Propagation (BCP)

MiniSAT

Unit clause rule:

Eine Klausel heisst *unit*, gdw. sie genau ein freies (ungebundenes) Literal enthält und alle gebundenen Literale zu FALSE auswerten.

$$X_1 \vee \neg X_2 \vee X_3 \vee X_4$$



$$X_1 \vee F \vee F \vee F$$

Freies Literal X1 muss unter der aktuellen Belegung TRUE werden!

MiniSAT

Boolean Constraint Propagation:

Sobald eine Klausel *unit* wird, kann der Wert des freien Literals propagiert werden.

Durch die Propagation werden u.U. andere Klauseln unit, die dann ebenfalls durch BCP verarbeitet werden können.

Bsp.:

$$(X_1) \wedge (\neg X_1 \vee X_2) \wedge (\neg X_2 \vee \neg X_3) \wedge (X_3 \vee X_4)$$

BCP

X_1 = **TRUE**

X_2 = **TRUE**

X_3 = **FALSE**

X_4 = **TRUE**

Lösen der Booleschen Gleichung

Anschaulich: Suchen einer erfüllenden Variablenbelegung

Naiver Ansatz:
Brute Force – Offensichtlich nicht brauchbar

Aber: Wie optimieren?

MiniSAT

Davis-Putnam-Logemann-Loveland-Suche (DPLL)

Grundlage für fast alle modernen SAT-Solver

```
while( !solved() ) do

  var x = decide();

  if( !propagate(x) ) then
    //a conflict occured!
    int bt = analyze_conflict();
    if(bt > root_level) then
      return UNSATISFIABLE;
    else
      perform_backtracking(bt);
    end
  end

end
```

MiniSAT

Mögliche Ansätze für Optimierungen:

- **Suche** – Welche Variablen zuerst zuweisen?
- **Propagation** – Effiziente Verfahren zum Erkennen und Auflösen von Unit Clauses
- **Backtracking** – Konflikte erkennen und durch kluges Backtracking umgehen
- **Lernen** – Nicht den selben Fehler ständig wiederholen

Entscheidungen im Suchbaum

Im Allgemeinen: 2 mögliche Belegungen für jede Variable

Ziel: Möglichst wenig Zuweisungen, „aussagekräftige“ Variablen zuerst belegen

Naiver Greedy-Ansatz:
Wähle immer diejenige Variable, bei deren Belegung die meisten Unit Clauses entstehen würden

MiniSAT

Variable State Independent Decaying Sum (VSIDS)

Idee: Löse die Variablen zuerst, die die meisten Probleme verursachen.

Umsetzung:

- Jeder Variable wird ein *Zähler* zugewiesen
- Tritt ein Konflikt auf, so wird der Zähler aller am Konflikt beteiligten Variablen erhöht
- Um Akkumulationseffekte zu vermeiden, werden die Zähler periodisch mit einer Konstanten <1 multipliziert.

Erstmals eingeführt in: *Chaff*, 2001
(Moskewicz, Madigan et al.)

MiniSAT

Variable State Independent Decaying Sum (VSIDS)

Zusätzliche Optimierungen in MiniSAT:

- Vermeiden von Counter-Downscaling (floating point!) durch wachsendes Zählerinkrement

- Vorsortieren der Variablenliste

- Randomisierung der Auswahl

MiniSAT

Variable State Independent Decaying Sum (VSIDS)

Und dann?

Propagieren!

MiniSAT

Propagieren

Im Allgemeinen: Finden von Unit Clauses und durch Anwendung der BCP neue Variablenbelegungen finden

Ziel: Speed!

SAT-Solver verbringen im Schnitt über 80% der gesamten Rechenzeit mit Propagation

MiniSAT

Watch Lists: Schnelles Finden von Unit Clauses

- Jede Variable verfügt über eine Watch-List (Liste von Klauseln in denen die Variable vorkommt)
- Jede Klausel beobachtet genau zwei ihrer Variablen, denen noch kein Wert zugewiesen wurde

Ändert sich die Belegung einer Variablen, informiert sie alle Klauseln auf ihrer Watch-List darüber.

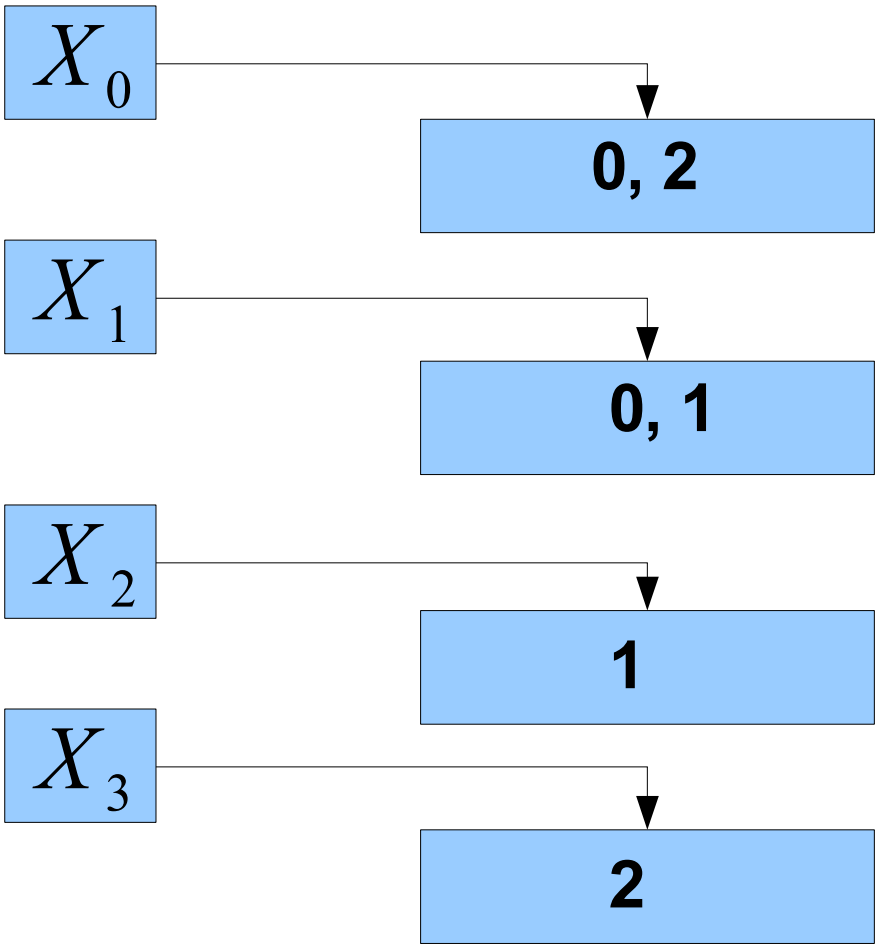
Kann die Klausel eine neue freie Variable finden, beobachtet sie diese. Falls nicht, heißt das, dass die Klausel unit ist und die verbleibende freie Variable propagiert werden kann!

MiniSAT

Watch Lists: Schnelles Finden von Unit Clauses

Bsp:

$$(X_0 \vee \neg X_1) \wedge (X_1 \vee X_2 \vee X_3) \wedge (\neg X_0 \vee X_3)$$

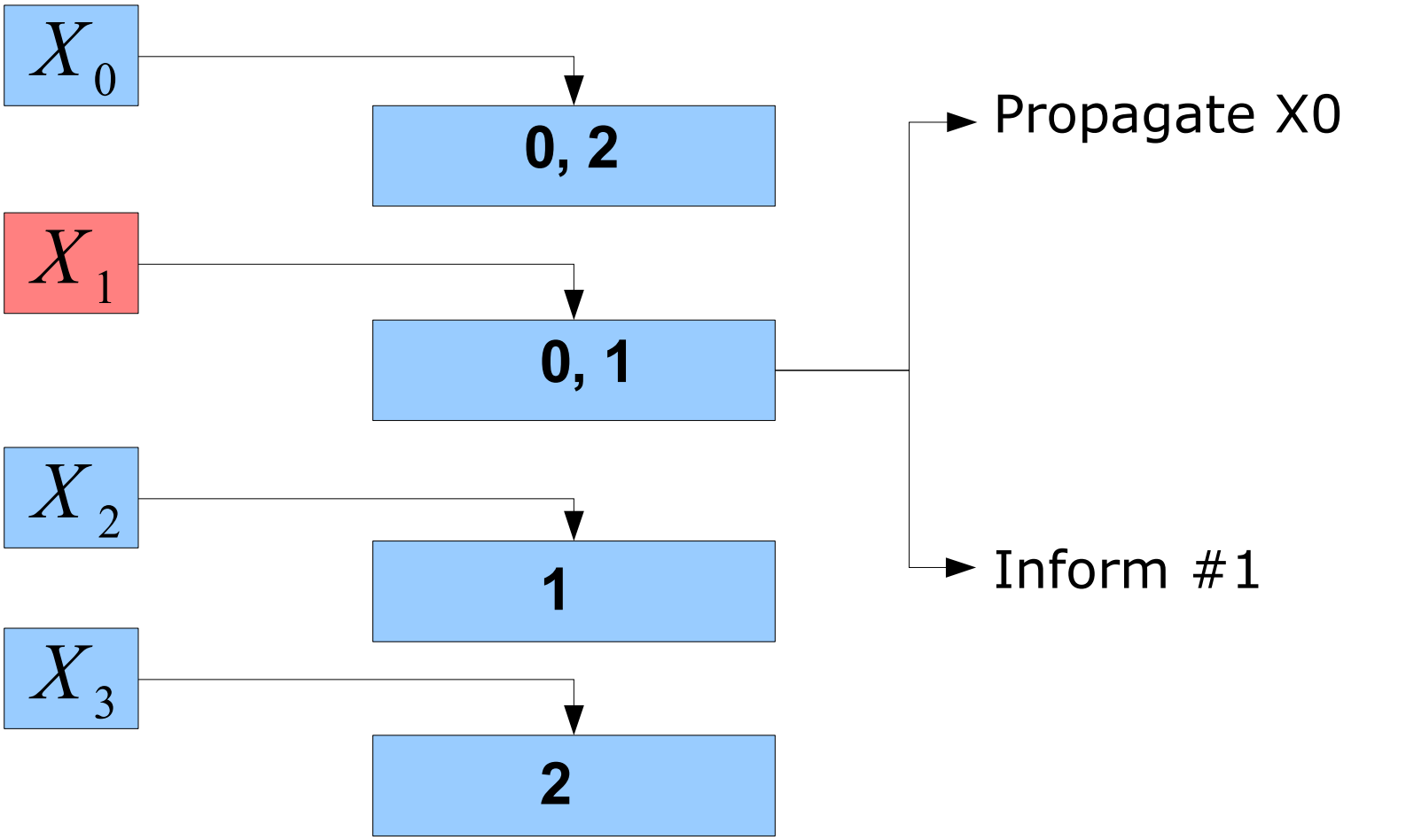


MiniSAT

Watch Lists: Schnelles Finden von Unit Clauses

Bsp:

$$(X_0 \vee \neg X_1) \wedge (X_1 \vee X_2 \vee X_3) \wedge (\neg X_0 \vee X_3)$$

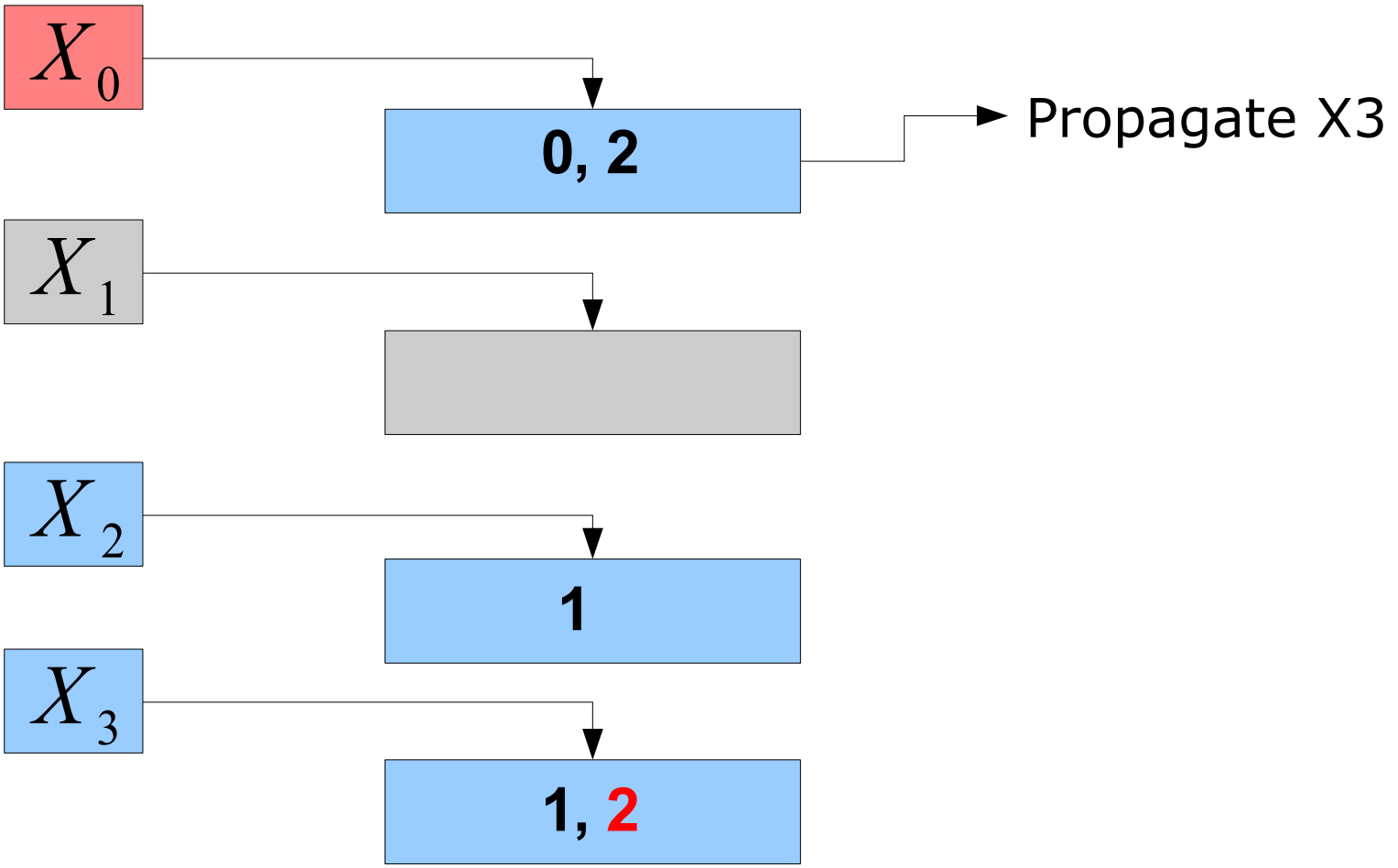


MiniSAT

Watch Lists: Schnelles Finden von Unit Clauses

Bsp:

$$(X_0 \vee \neg X_1) \wedge (X_1 \vee X_2 \vee X_3) \wedge (\neg X_0 \vee X_3)$$

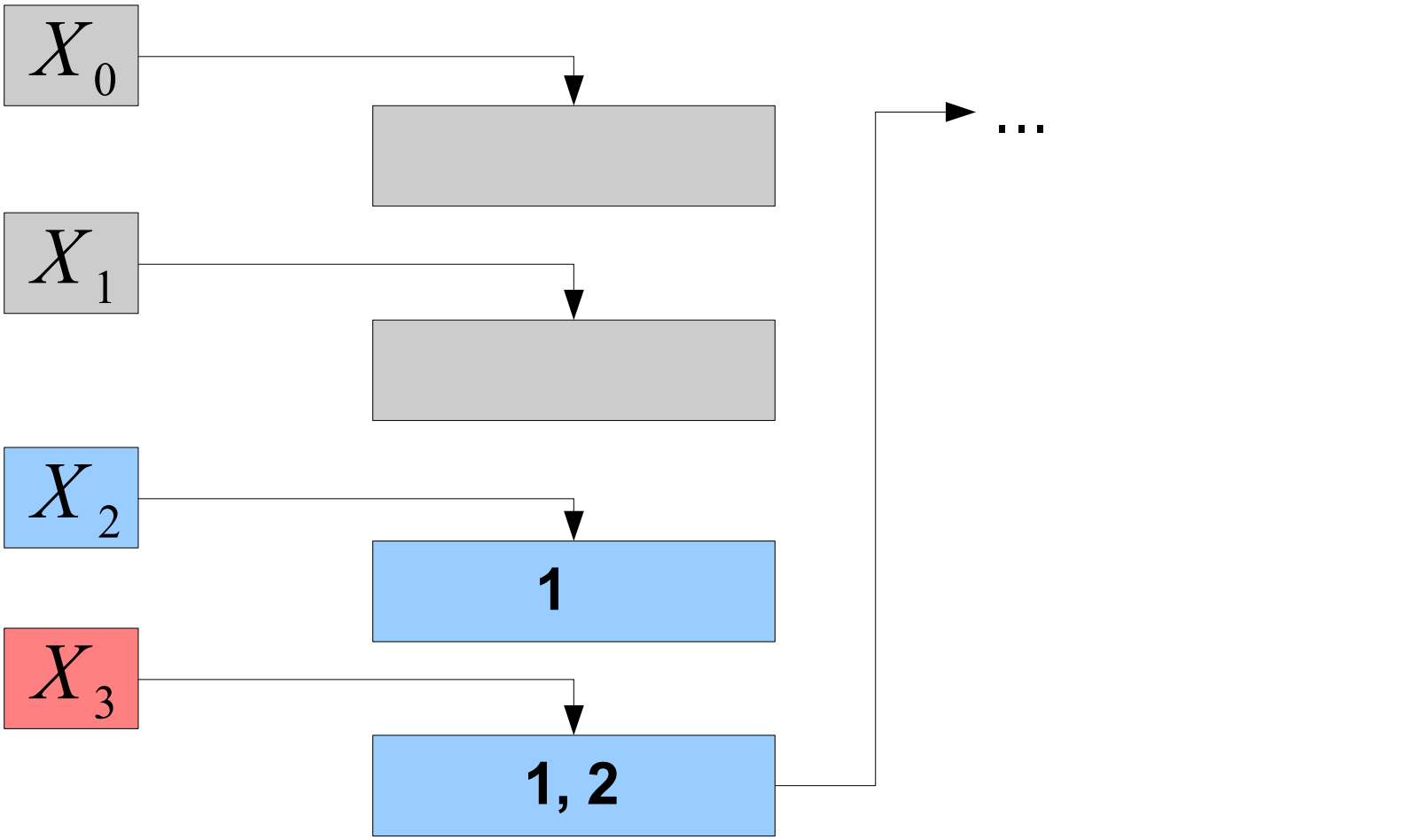


MiniSAT

Watch Lists: Schnelles Finden von Unit Clauses

Bsp:

$$(X_0 \vee \neg X_1) \wedge (X_1 \vee X_2 \vee X_3) \wedge (\neg X_0 \vee X_3)$$



MiniSAT

Watch Lists: Schnelles Finden von Unit Clauses

Vorteile gegenüber naiver Implementierung eines Zählers:

- Änderungen an der Watch-List fallen vergleichsweise selten an
- Backtracking erfordert überhaupt keine Änderung der Watch-List!

MiniSAT

Propagation Queue

Propagation wird nicht sofort ausgeführt!

Stattdessen werden zu propagierende Variablen in eine Warteschlange eingereiht, die pro Suchtiefe genau einmal abgearbeitet wird.

Vorteil: Sequentialisierung der Propagation, Vermeidung von Schachtelungen

Führt die Propagation zu einem Widerspruch ist *Backtracking* nötig...

MiniSAT

Backtracking

Beheben von Konflikten, Rückführen des Solvers in einen widerspruchsfreien Zustand

Ziel: Erkennen von Fehlern, rechtzeitiges Verlassen von widersprüchlichen Suchzweigen

Naiver Ansatz: Single-Step-Backtracking

MiniSAT

Hilfskonstrukte

- *Trail* – Stackstruktur, enthält alle Variablenzuweisungen in der Reihenfolge, in der sie gemacht wurden
- *Reasons* – Wird einer Variable durch Propagation ein Wert zugewiesen, merkt sie sich die Klausel die für die Propagation verantwortlich war
- *Decision Levels* – Eine Variable merkt sich die Tiefe im Suchbaum, in der sie zugewiesen wurde

Non-Chronological Backtracking

Fehleranalyse

Single-Step-Backtracking braucht u.U. Viele teure Einzelschritte, ehe ein widerspruchsfreier Zustand erreicht wird.

Ziel: Genaue Lokalisierung der Fehlerursache und gezieltes Backtracking

MiniSAT

Non-Chronological Backtracking

Fehleranalyse

Eingabe: Widersprüchliche Klausel

- Für jedes Literal der Klausel:
  - Falls das Literal im aktuellen Dlevel zugewiesen wurde, suche die *reason* für die Zuweisung
  - Falls das Literal früher zugewiesen wurde, füge es in eine Liste ein
- Für jede reason aus dem obigen Schritt:

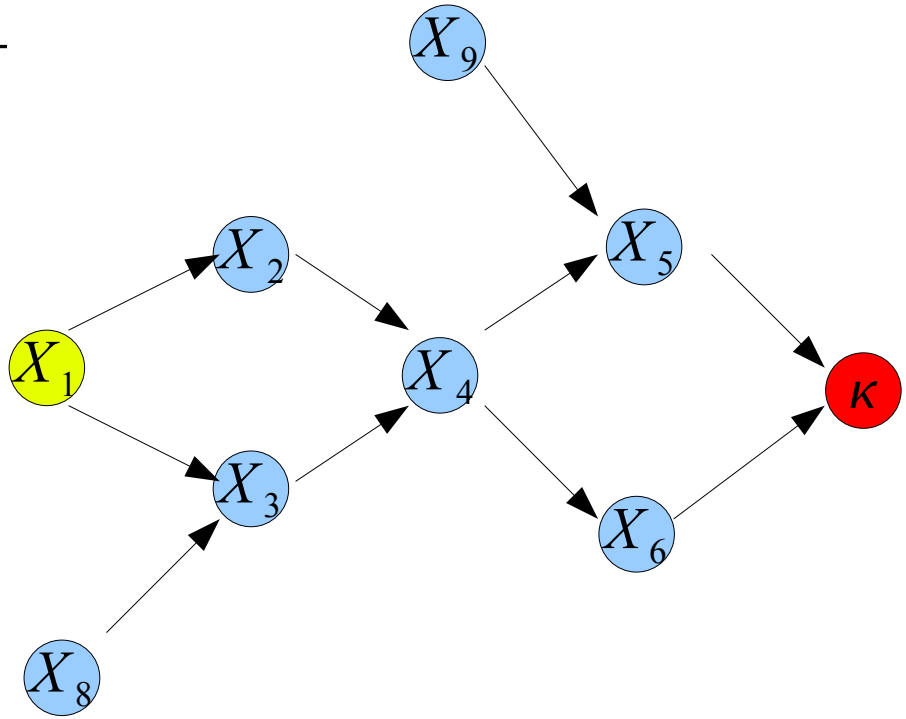
Wiederhole die obigen Schritte

Ergebnis: Liste der Literale, die den Fehler verursachen

MiniSAT

Non-Chronological Backtracking

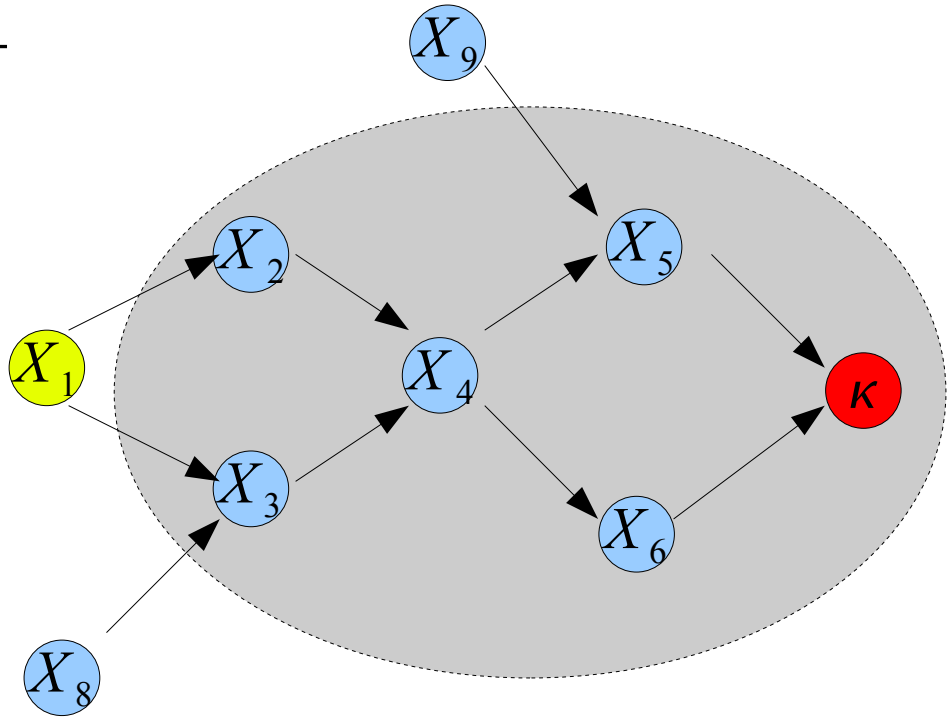
Bsp:



MiniSAT

Non-Chronological Backtracking

Bsp:



$$A_c(\kappa) = (X_1 = 0, X_8 = 1, X_9 = 0)$$

$$\omega_c(\kappa) = (X_1 \vee \neg X_8 \vee X_9)$$ (*Conflict Induced Clause*)

Non-Chronological Backtracking

Aus der Liste der fehlerverursachenden Literale wird der Backtracking-Level bestimmt.

Eigentliches Backtracking ist als Rückwärtsgehen auf dem Trail implementiert

Failure Driven Assertions

- Falls die aktuelle Entscheidung einen Konflikt impliziert, wird die Entscheidung negiert (*Failure Driven Assertion*)

Restarts

- Treten mehr als n Konflikte während eines Durchlaufs auf, so startet der Solver die Suche komplett neu

Problem: Wie verhindert man Wiederholung von Fehlern?

Learning

Aus alten Fehlern lernen

Ziel: Fehler möglichst nicht wiederholen

Erstmals eingeführt: *GRASP*, 1996 (Marques-Silva, Sakallah, et al.)
Einer der wichtigsten Fortschritte im SAT-Solving der letzten Jahre!

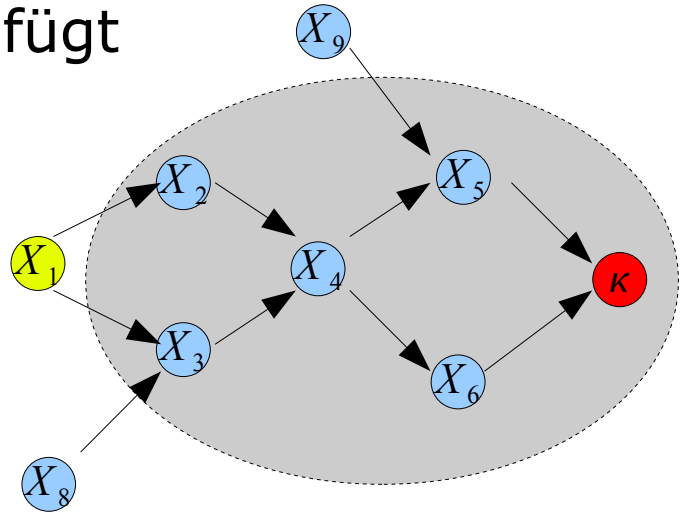
MiniSAT

Learning

Eng verknüpft mit der Analyse des Backtrackings

Durch Negation der Fehlerklausel erhält man eine neue Klausel.

Diese Conflict-induced Clause wird zur Datenbank hinzugefügt



$$A_c(\kappa)=(X_1=0, X_8=1, X_9=0)$$

$$\omega_c(\kappa)=(X_1 \vee \neg X_8 \vee X_9)$$

MiniSAT

Learning

Die gelernten Klauseln sind für das gesamte Problem gültig und bleiben auch nach einem Backtracking oder Restart erhalten

Folge: Der Solver wird immer besser, je länger er an einem bestimmten Problem arbeitet

Einziger Nachteil: Clause-Database wird evt. sehr groß

MiniSAT

Demo!

$$(X_0 \vee X_1 \vee X_2) \qquad (X_0 \vee \neg X_1)$$
$$(X_3 \vee X_2) \qquad (\neg X_1 \vee X_2)$$
$$(\neg X_5 \vee X_0 \vee X_1 \vee \neg X_2)$$
$$(X_4 \vee X_5)$$
$$(\neg X_4 \vee \neg X_5)$$
$$(X_4 \vee X_6)$$
$$(\neg X_6)$$
$$(X_5 \vee X_6)$$

MiniSAT

Verschiedene Dinge

- Top-Level simplification
- Clause deacaying
- Incremental SAT-Solving

Conclusion

Fragen? Unklarheiten?

Vielen Dank für die Aufmerksamkeit!

Feierabend