Karlsruhe Institute of Technology

Entscheidungsverfahren mit Anwendungen in der Softwareverifikation

II: Aussagenlogische Erfüllbarkeit

Dr. Stephan Falke
Dr. Carsten Sinz
Institut für Theoretische Informatik

22.04.2013

Definition SAT-Problem

KIT Karlsruhe Institute of Technology

- Gegeben: Formel F in CNF.
- Frage: Ist F erfüllbar, d.h. gibt es ein Modell α von F?
  - Modell: Abbildung α: Var(F) → {0,1}, so dass in jeder Klausel von F mindestens ein Literal mit wahr belegt ist
- Beispiel: F = {{x,¬y}, {¬x,¬z}, {y}}
  - Für x=y=1 und z=0 evaluiert F zu 1 (also ist F erfüllbar)
- Anmerkung:
  - Ja/Nein-Antwort oft nicht ausreichend
  - Begründung in vielen Anwendungen erforderlich

CNF-Darstellung: DIMACS-Format

- Format zur Darstellung von Formeln in CNF auf dem Computer.
- Aufbau einer DIMACS-Datei:
  1. Optionale Kommentarzeilen: `c Kommentar`
  2. Präambel: `p cnf n m` (n: Anzahl Variablen, m: Anzahl Klauseln)
  3. Klauseln:
     - Liste der Literale, durch Leerzeichen getrennt, 0 als Abschluss
     - Variablen repräsentiert durch Ganzzahlen (>0), „-“ als Negationssymbol

DIMACS-Format: Beispiel

KIT Karlsruhe Institute of Technology

$F = \{\{x_1,x_2,\neg x_3\}, \{x_3,\neg x_4\}, \{\neg x_1,\neg x_3,x_4\}, \{\neg x_2,x_3,x_5\}\}$
im DIMACS-Format:

```
c Dies ist eine Formel in CNF
c mit 5 Variablen und 4 Klauseln.
p cnf 5 4
1 2 -3 0
3 -4 0
-1 -3 4 0
-2 3 5 0
```

(letzte 0 optional)

Vereinfachung durch Unit-Klauseln

- Unit-Klausel: enthält nur ein Literal, d.h. $C = \{m\}$.
- Beobachtung: In jedem Modell α muss *m* mit wahr belegt sein.
- Dadurch Vereinfachung möglich:
  - ¬m kann aus allen anderen Klauseln in F gestrichen werden (da ¬m immer unter α mit falsch belegt ist).
  - Wenn dadurch die leere Klausel (□, entspricht 0) entsteht, ist das Problem unerfüllbar.
- Bezeichnung: Unit-Resolution

DPLL-Algorithmus: Übersicht

- DPLL: Davis-Putnam-Logemann-Loveland
- Grundlegender Algorithmus ~1965
- Grundidee: Fallunterscheidungen und Vereinfachung
- Vereinfachungen:
  - Unit-Resolution
  - Unit-Subsumption
  - Löschen purer Literale (pure literal deletion)

} Unit-Propagation

Grundlegender DPLL-Algorithmus

```
boolean DPLL(ClauseSet S)
{
  while ( S contains a unit clause {L} ) {
    delete from S clauses containing L;   // unit-subsumption
    delete ¬L from all clauses in S;      // unit-resolution
  }
  if ( □ ∈ S ) return false;              // empty clause?
  while ( S contains a pure literal L )
    delete from S all clauses containing L;
  if ( S = ∅ ) return true;               // no clauses?
  choose a literal L occurring in S;      // case-splitting
  if ( DPLL(S ∪ {{L}} ) return true;      // first branch
  else if ( DPLL(S ∪ {{¬L}} ) return true; // second branch
  else return false;
}
```

Beispiel Unit-Propagation

S = {{¬x,y,¬z}, {¬x,z}, {¬y,x}, {y}}

1. Unit-Klausel vorhanden? Ja: {y}
2. Unit-Subsumption: lösche Klauseln, in denen y vorkommt:
   S_1 = {{¬x,z}, {¬y,x}}
3. Unit-Resolution: lösche ¬y aus allen Klauseln:
   S_2 = {{¬x,z}, {x}}
4. Unit-Klausel vorhanden? Ja: {x}
5. Unit-Subsumption: S_3 = {{¬x,z}}
6. Unit-Resolution: S_4 = {{z}}
7. Unit-Klausel vorhanden? Ja: {z}
8. Unit-Subsumption: S_5 = {}
9. Unit-Resolution: keine Klauseln mehr vorhanden

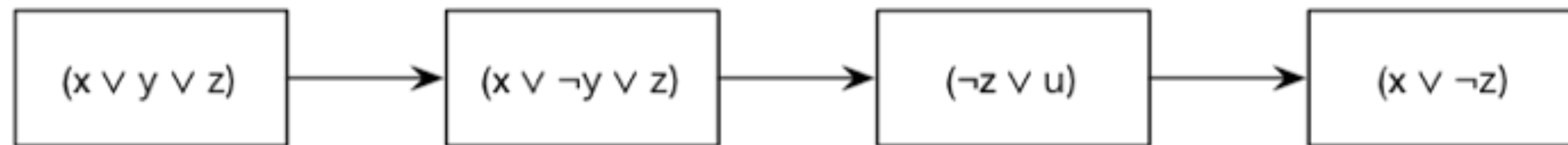
Ergebnis: S erfüllbar!

```
boolean DPLL(ClauseSet S)
{
  while ( S contains a unit clause {L} ) {
    delete from S clauses containing L;   // unit-subsumption
    delete ¬L from all clauses in S;      // unit-resolution
  }
  if ( □ ∈ S ) return false;               // empty clause?
  while ( S contains a pure literal L )
    delete from S all clauses containing L;
  if ( S = ∅ ) return true;                // no clauses?
  choose a literal L occurring in S;       // case-splitting
  if ( DPLL(S ∪ {{L}} ) return true;       // first branch
  else if ( DPLL(S ∪ {{¬L}} ) return true; // second branch
  else return false;
}
```

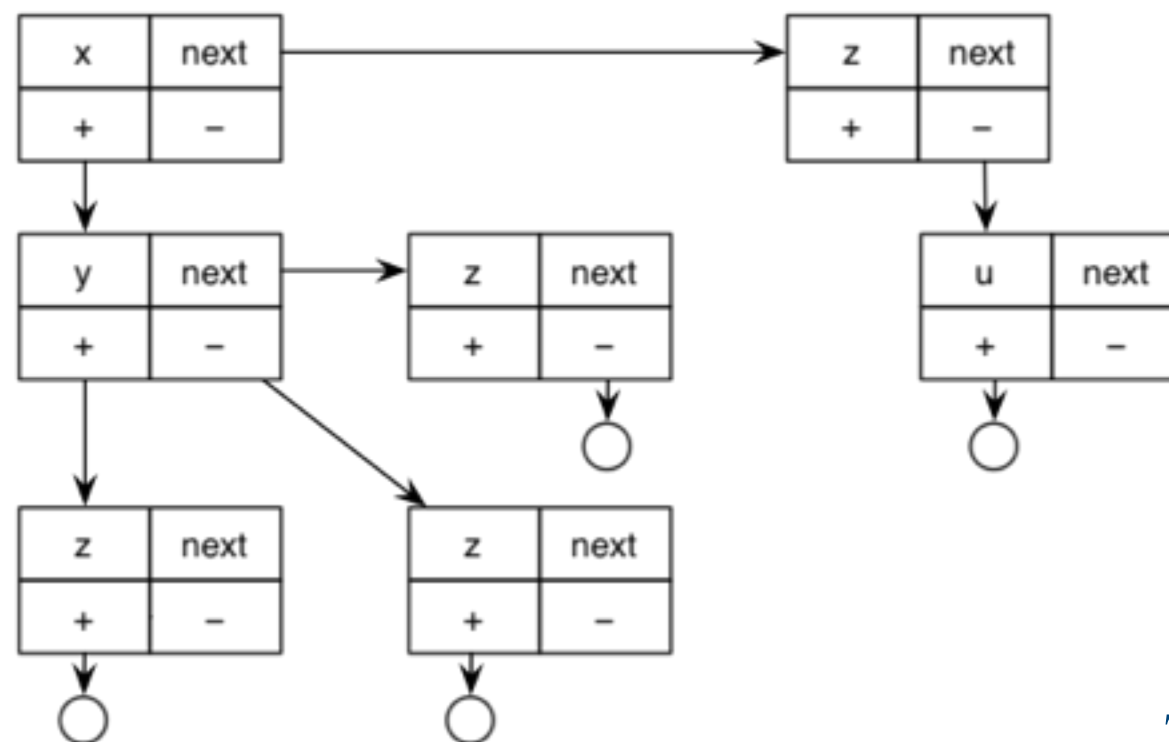
Datenstrukturen zur Repräsentation von Formeln in CNF

Wie können Klauselmengen dargestellt werden?

A) Als Liste von Klauseln



B) Als Trie (= prefix tree) Datenstruktur



Zhang, Stickel (1994)

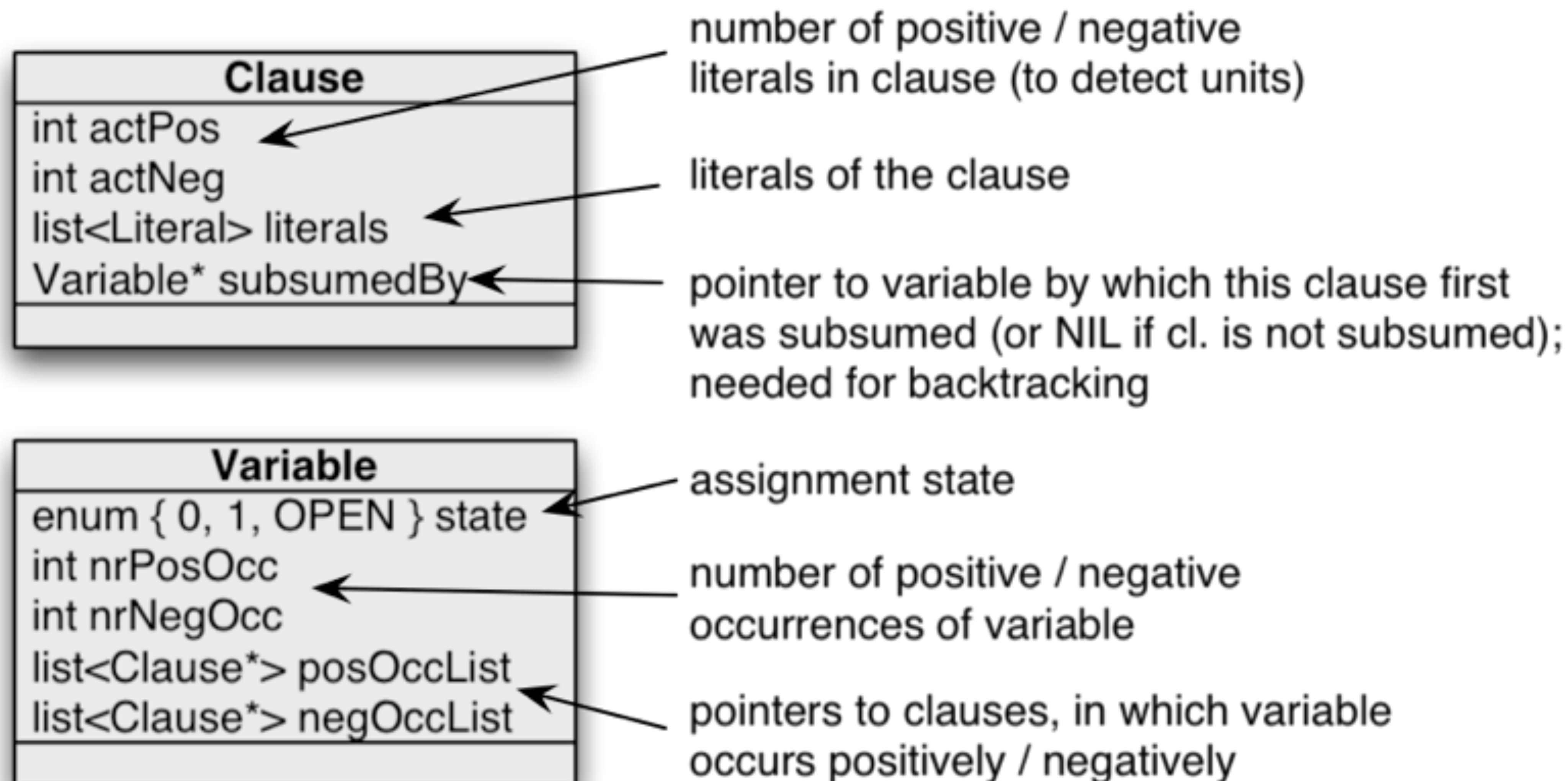
Datenstrukturen für DPLL: Anforderungen

- Sollen schnelle Unit-Propagation ermöglichen
  - Erkennen neuer Units
  - Propagierung von Units
- Unterstützung von Back-Tracking (Wiederherstellung von Klauselmengen)

  Implementationsalternativen:
  - Speichere Kopie der Klauselmenge bei jedem rekursiven Aufruf
  - Speichere nur Änderungen an der Datenstruktur (undo-stack)

  Ziel: Minimiere Aufwand zur Wiederherstellung
- Kompakte Darstellung großer Klauselmengen

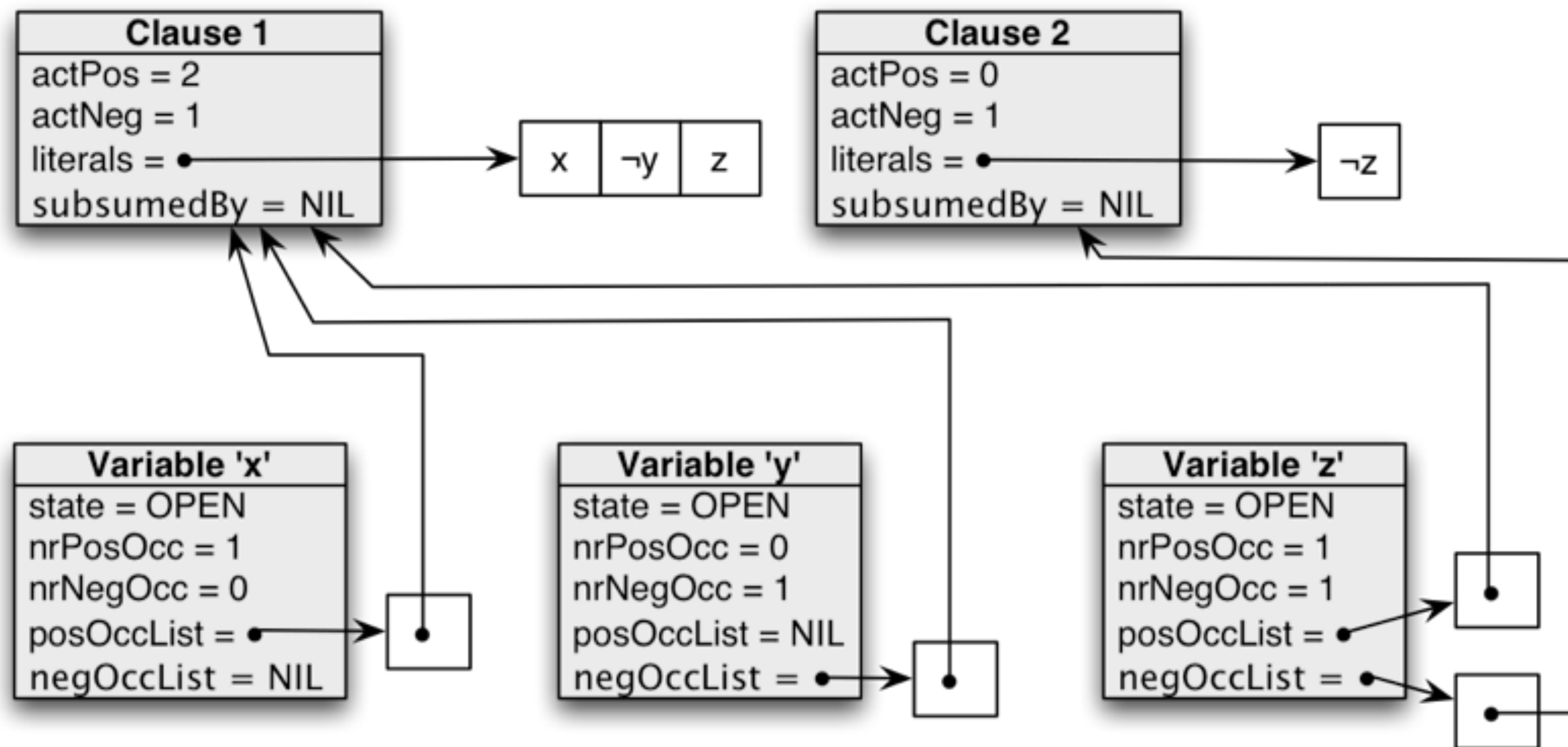
„Traditioneller“ Ansatz



Crawford, Auton (1993)

Traditioneller Ansatz: Beispiel

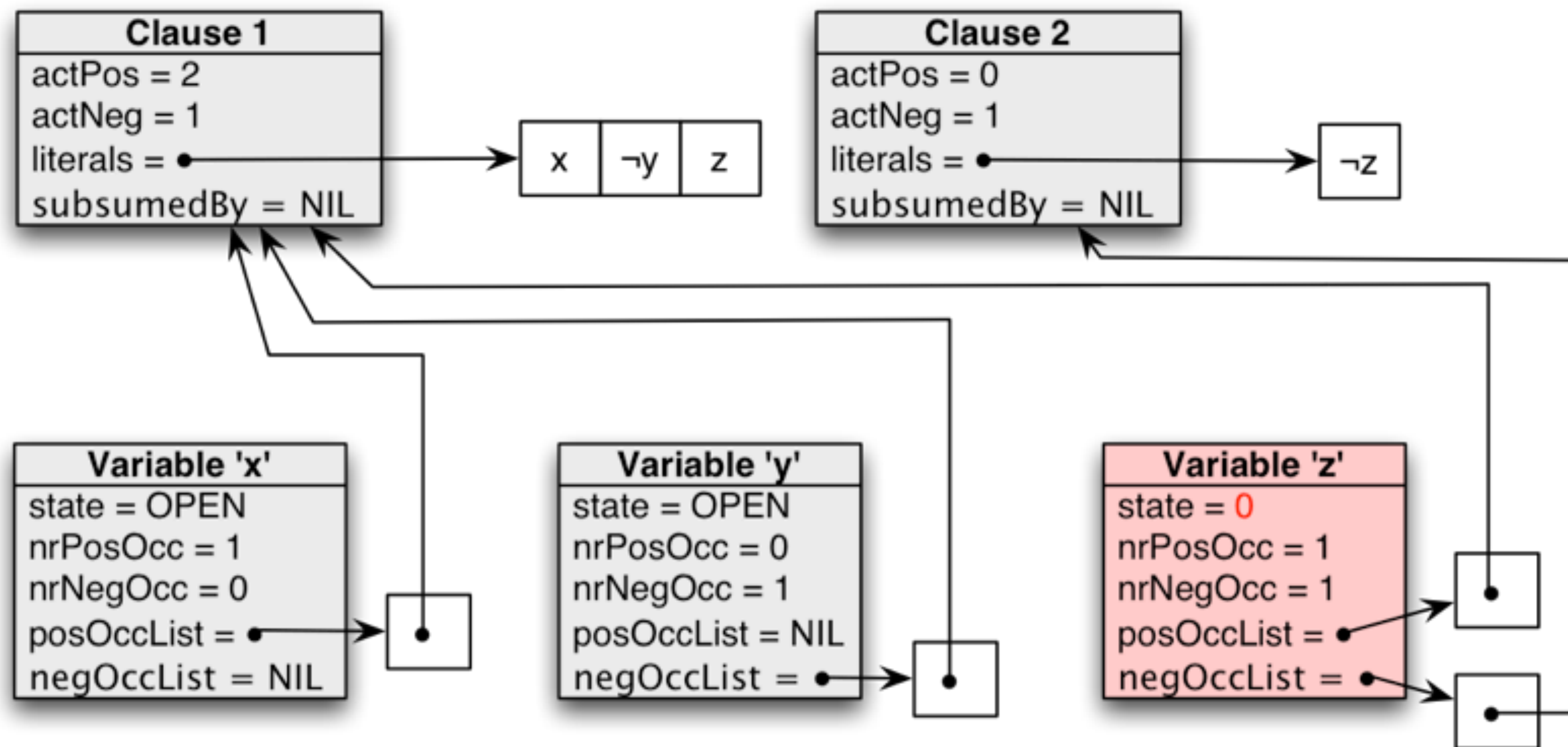
$$F = \{\{x, \neg y, z\}, \{\neg z\}\}$$



Beispiel: Unit Propagation

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

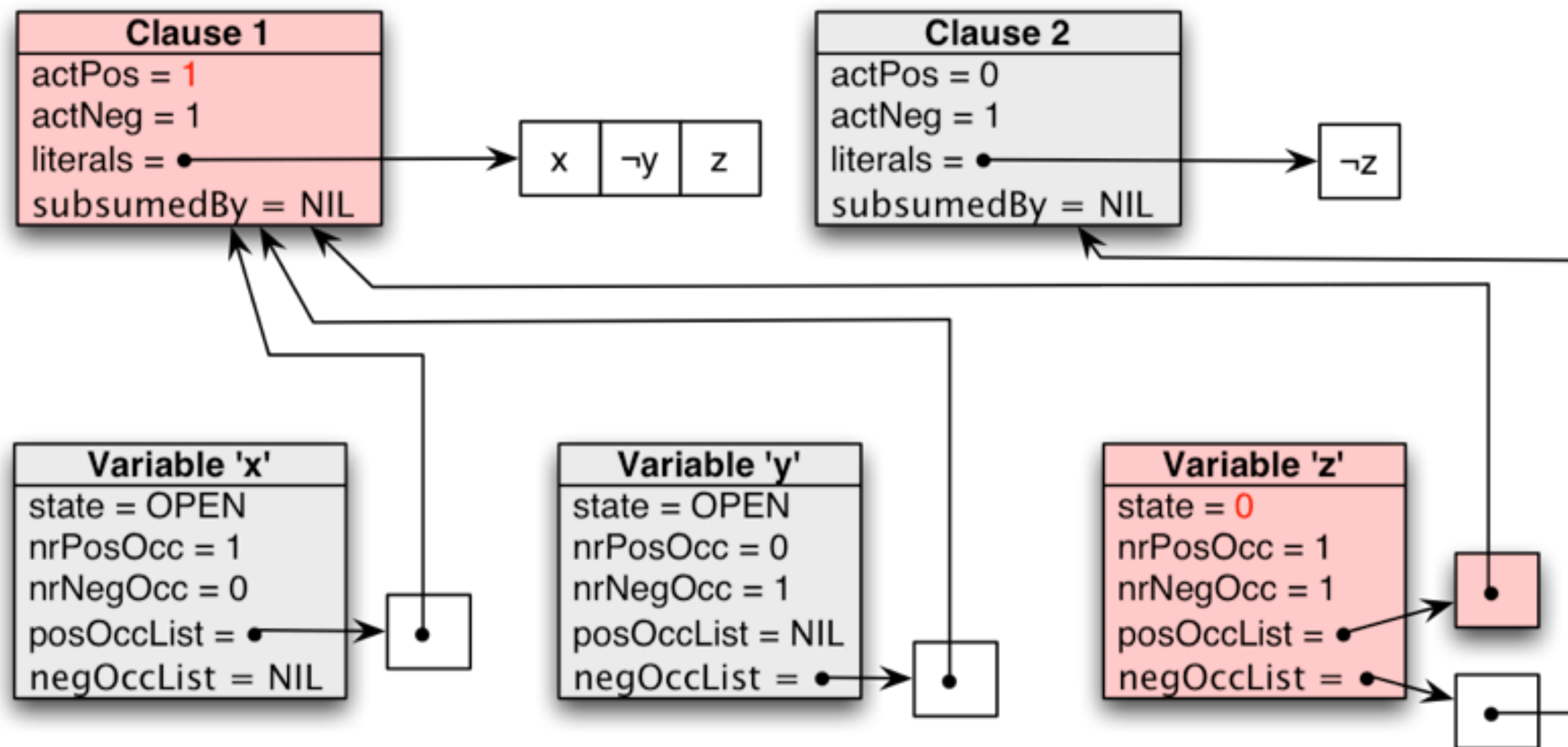
Unit propagation: setze z=0



Beispiel: Unit Propagation

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

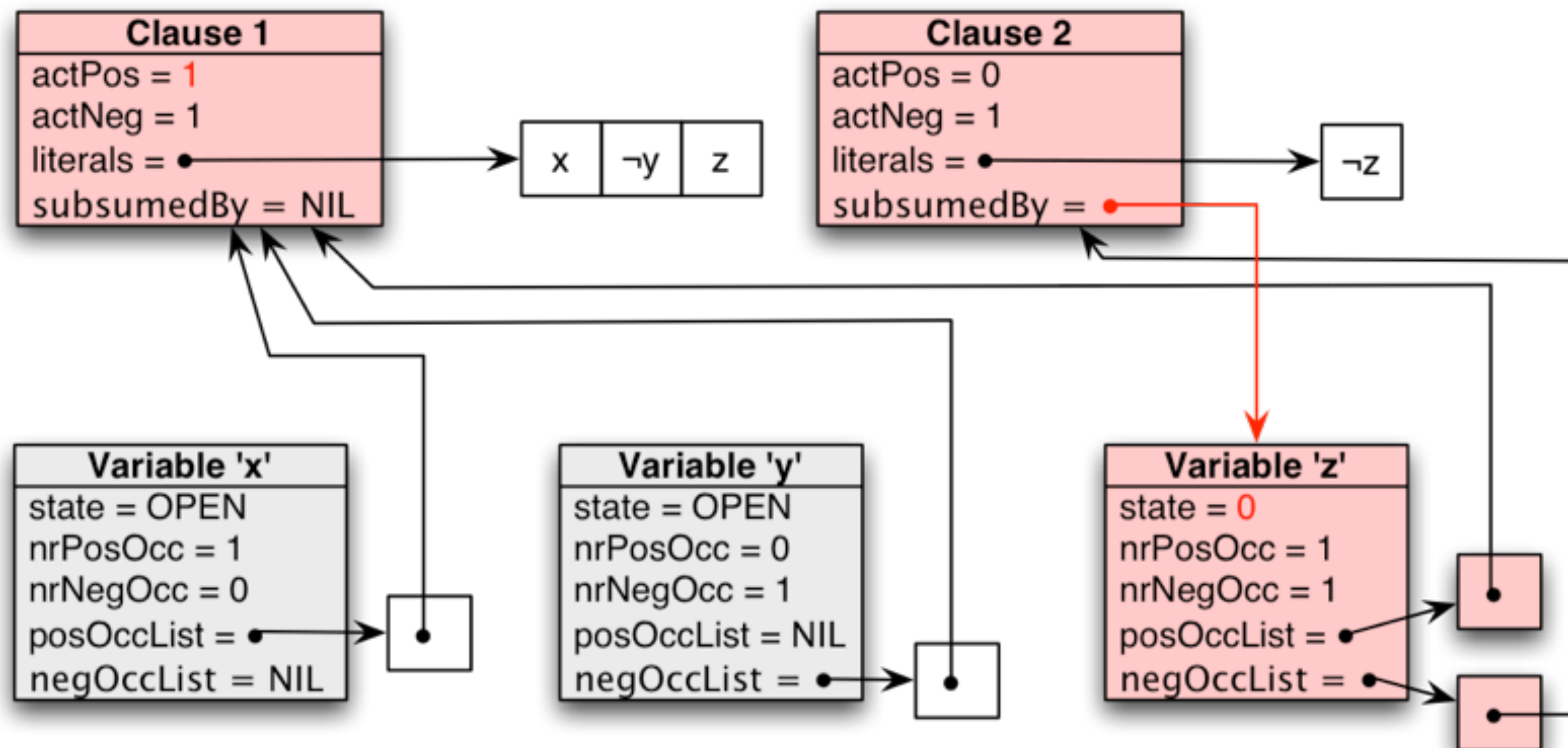
Unit propagation: setze z=0



Beispiel: Unit Propagation

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

Unit propagation: setze z=0



Algorithmus Unit-Propagation

Karlsruhe Institute of Technology

```
boolean UnitProp(Literal L)          // L: open literal; ‘UnitProp’ returns
{    if(L.isPositive()) {            // false on contradiction
          v = L.var(); v.state = 1;
          // process subsumed clauses
          for(it = v.posOccList.begin(); it != v.posOccList.end(); it++) {
               clause = *it;
               if(clause.subsumedBy == NIL) clause.subsumedBy = v;
          }
          // unit resolution
          for(it = v.negOccList.begin(); it != v.negOccList.end(); it++) {
               clause = *it;  if(clause.subsumedBy == NIL) {
                 clause.actPos--;          // shorten clause
                 if(v.actPos + v.actNeg == 1) {       // new unit clause detected
                      ok = HandleNewUnit(clause);
                      if(!ok) return false;         // conflicting units?
     }         }        } }
     else { ... // L is negative literal }
     return true;
}
```

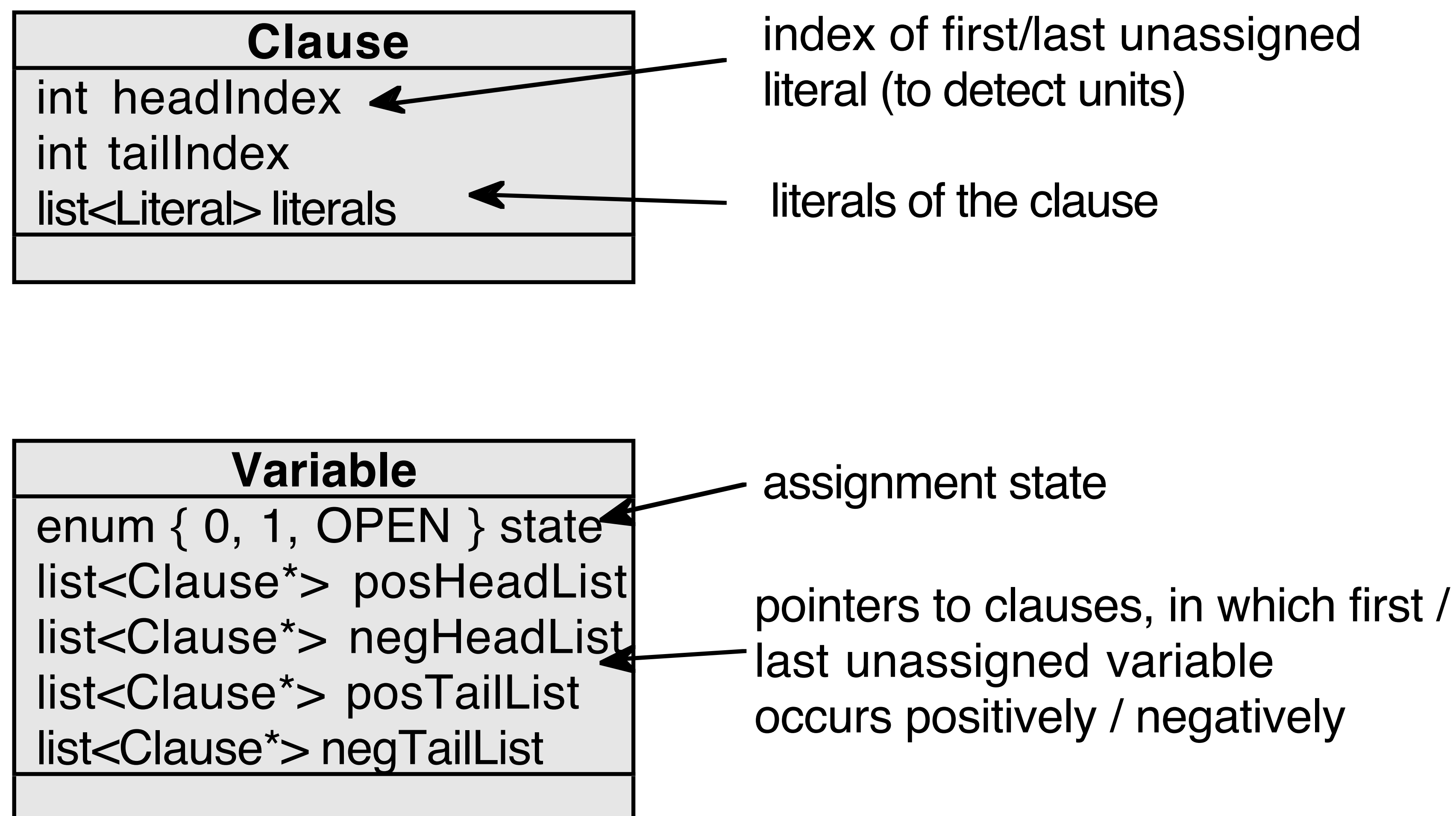
UP Algorithmus:

- Beim Setzen von Variable x (auf true) ...
  - werden (max.) |posOccList(x)| Klauseln subsumiert
  - werden (max.) |negOccList(x)| durch UR verkürzt
  - insgesamt: #occ(x) Klauseln werden modifiziert
- Lässt sich dies verbessern?
  - Beobachtung: Verkürzen von Klauseln (durch UR) ist erst dann kritisch, wenn eine neue Unit-Klausel entsteht.
  - Idea (Zhang, Stickel (1996)):
    - **Verzögerter Subsumtions-Test**

      'subsumedBy' wird nicht mehr verwendet; stattdessen wird bei Entstehen einer neuen Unit-Klausel geprüft, ob diese bereits subsumiert wurde.
    - **Unit-Resolution beschränkt auf erstes und letztes offenes (d.h. noch nicht gesetztes) Literal in Klausel**

      Arbeite mit pos/negHeadList und pos/negTailList anstelle von pos/negOccList.

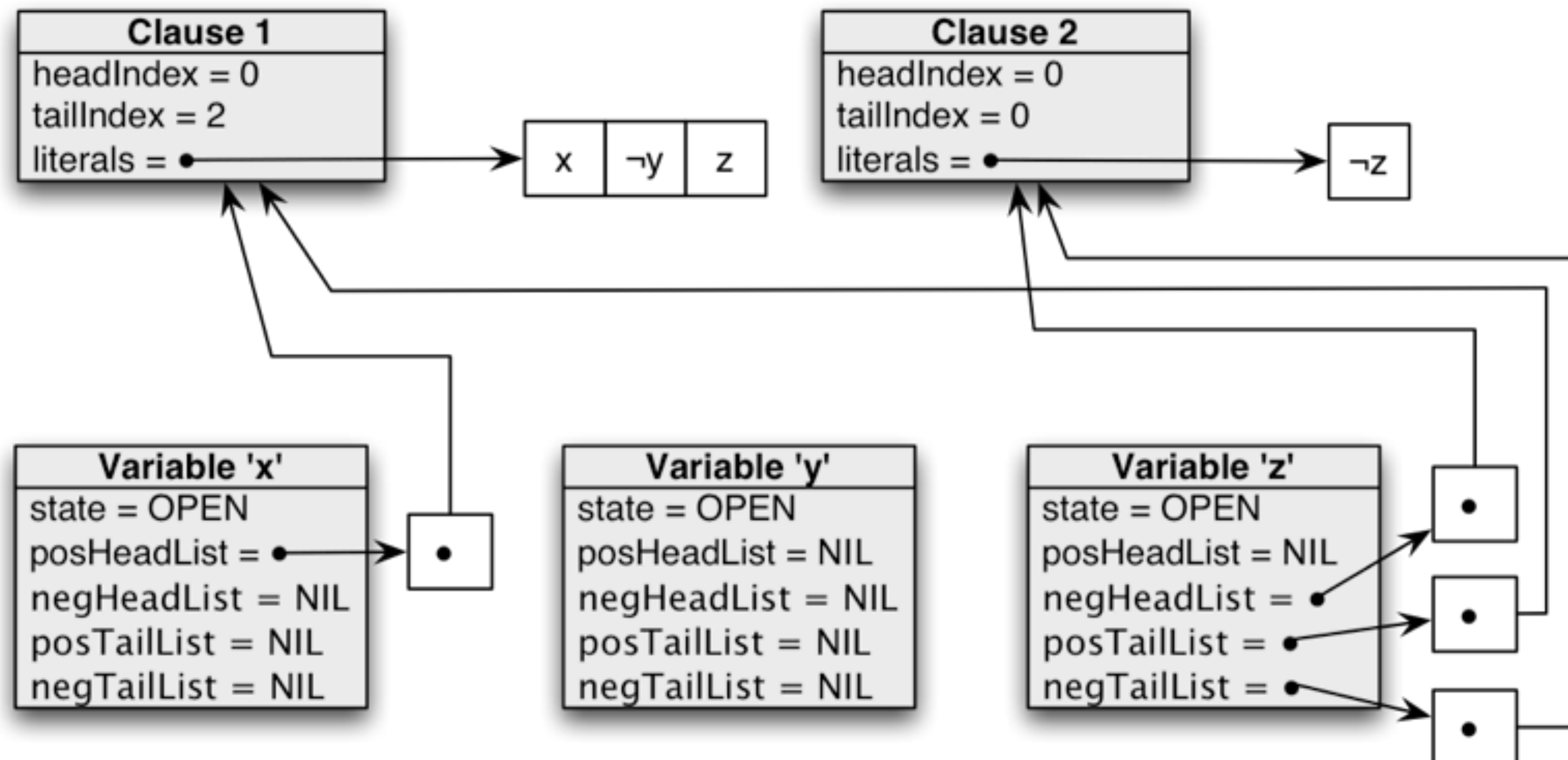
Head- und Tail-Listen: Datenstrukturen



Zhang, Stickel (1996)

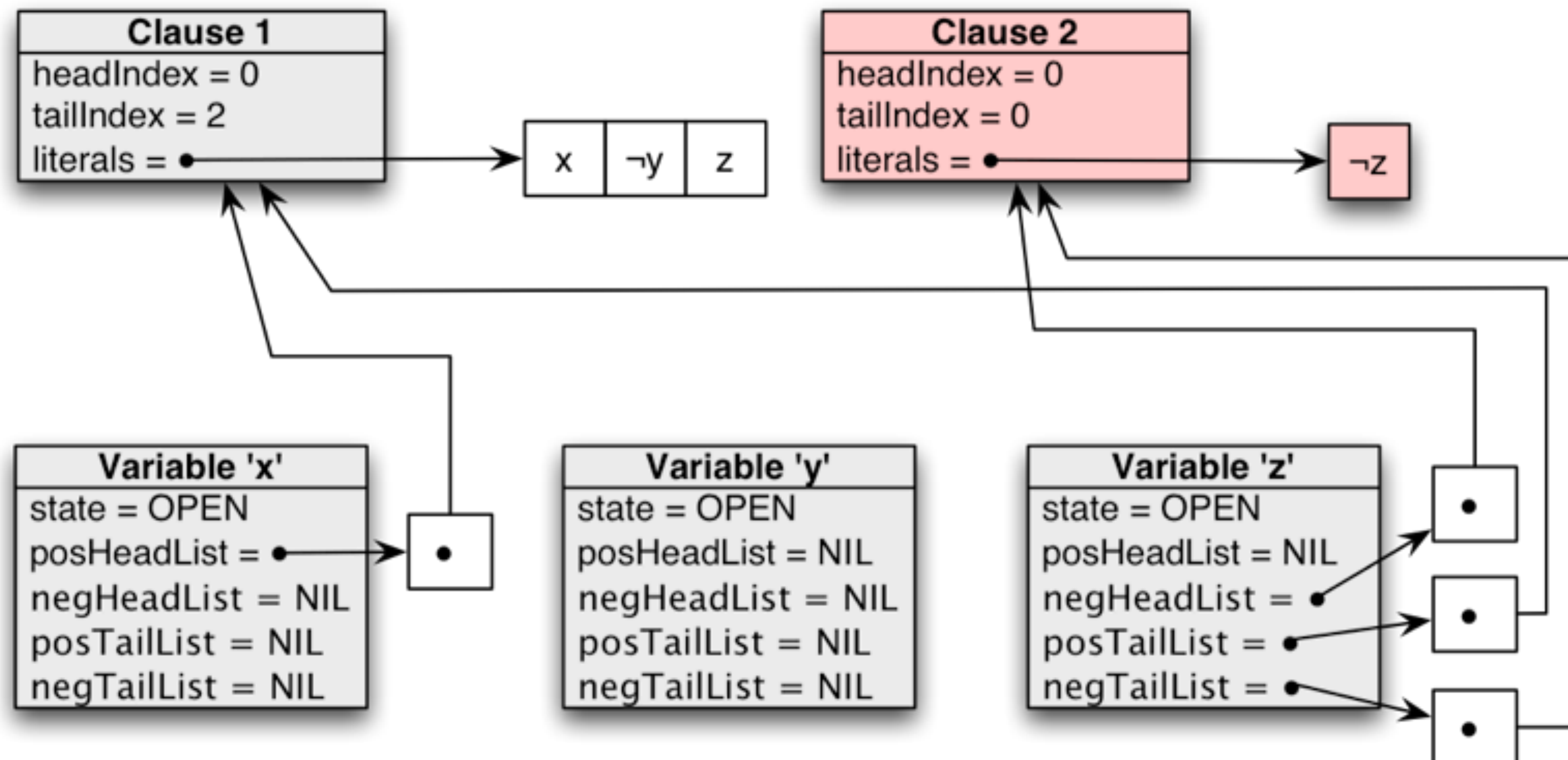
Head- und Tail-Listen: Beispiel

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



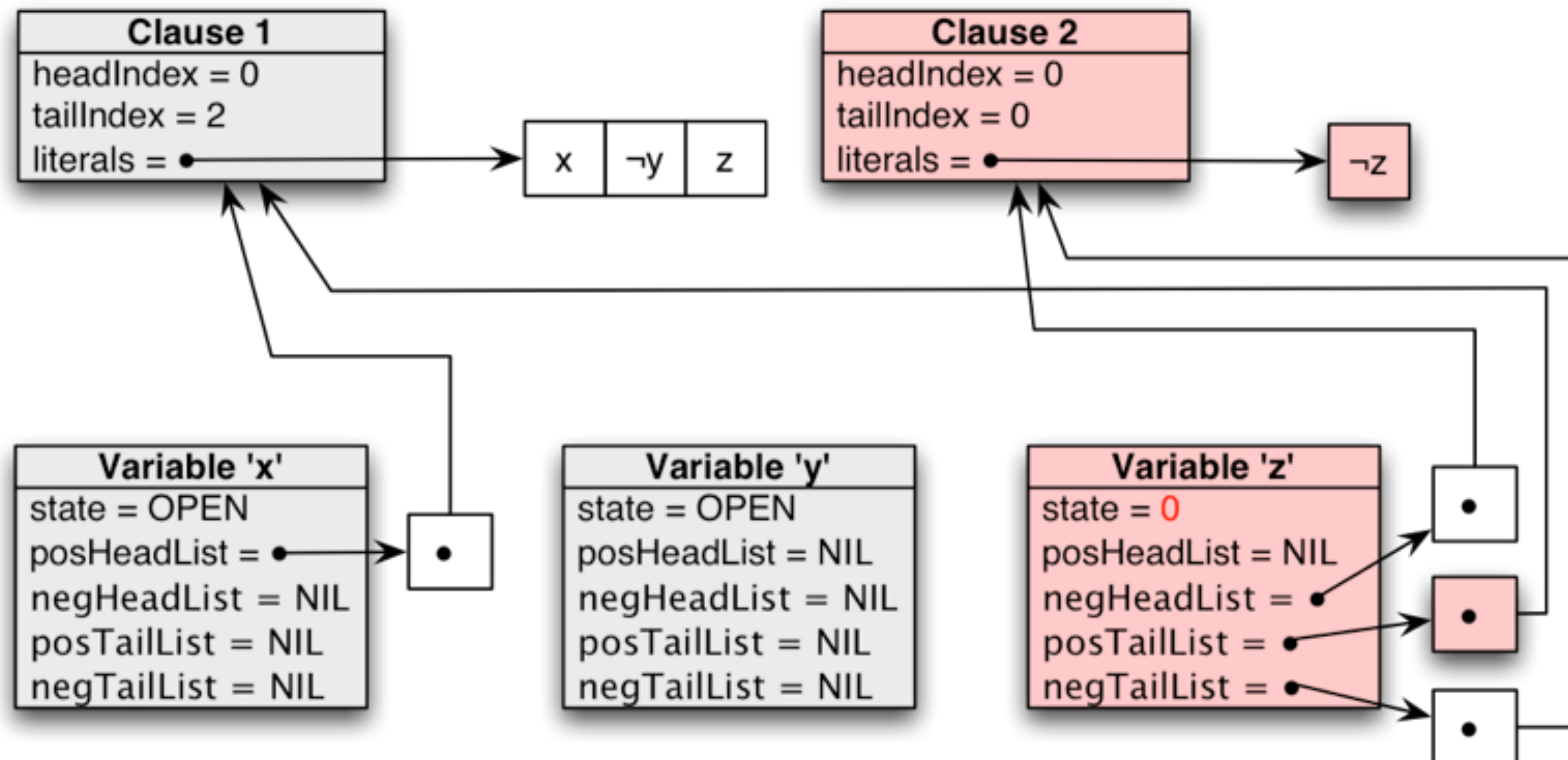
Head- und Tail-Listen:

$F = \{\{x, \neg y, z\}, \{\neg z\}\}$ Unit propagation: setze z=0



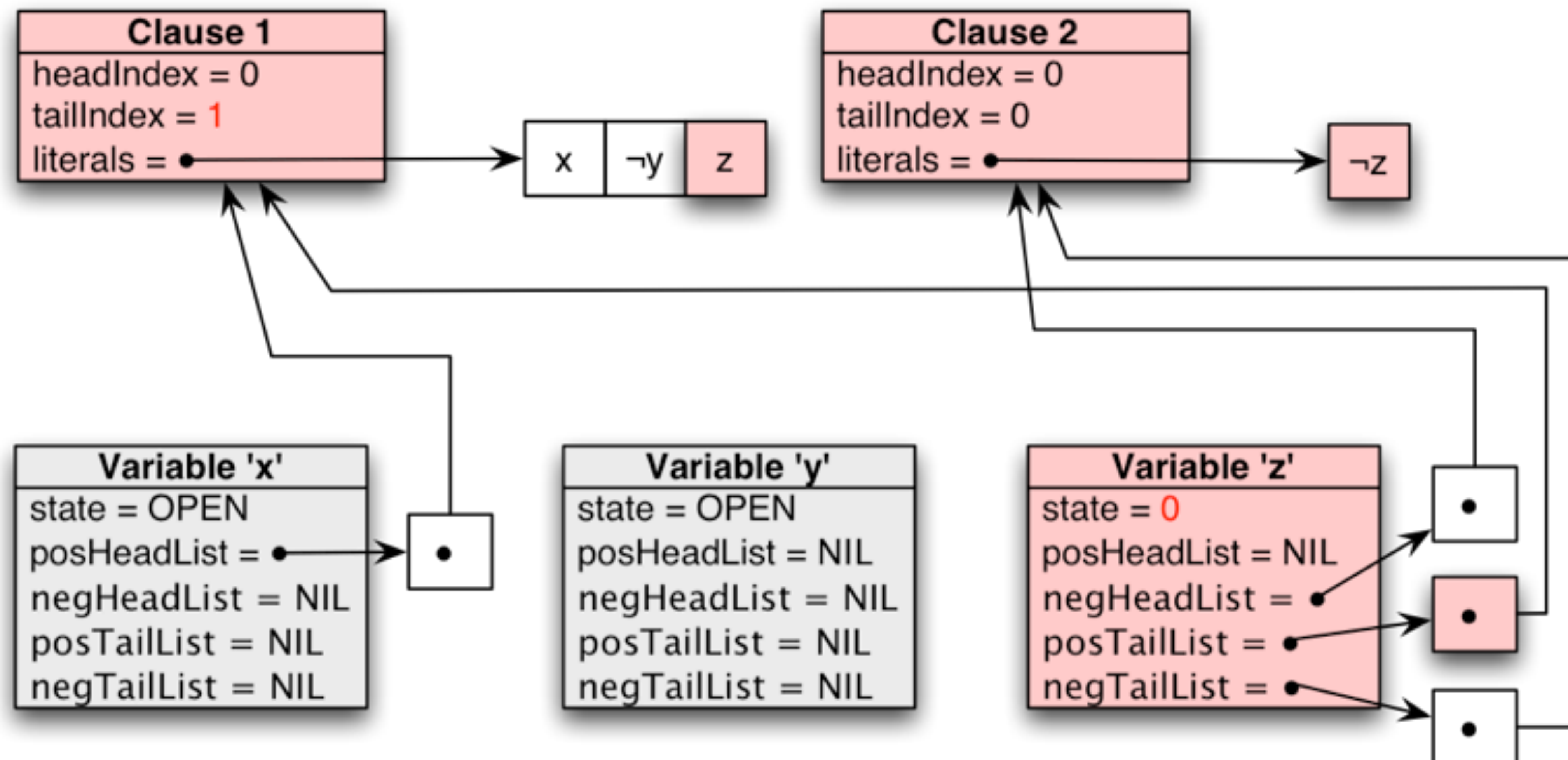
Head- und Tail-Listen:

$F = \{\{x, \neg y, z\}, \{\neg z\}\}$ Unit propagation: setze z=0



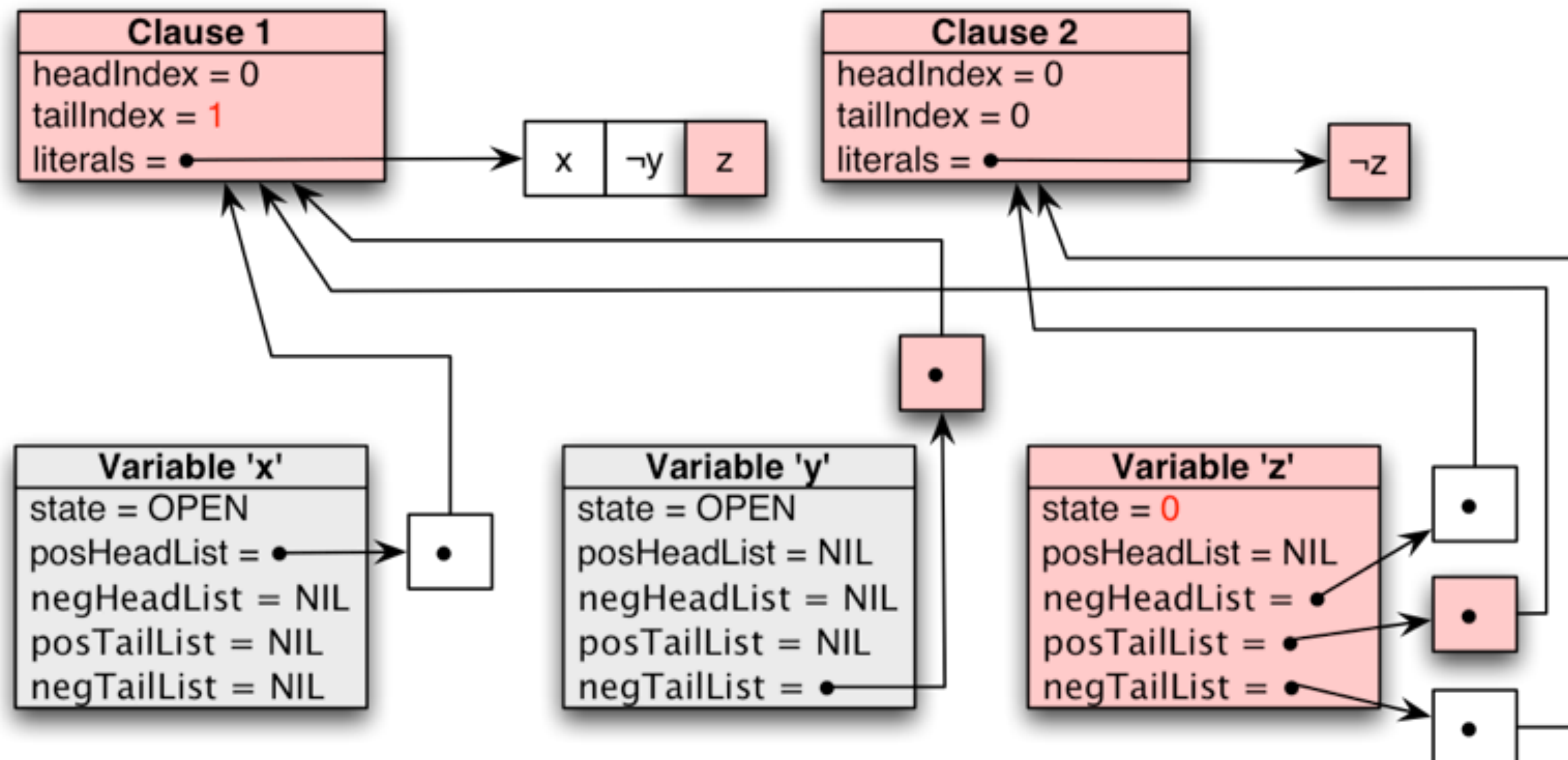
Head- und Tail-Listen:

$F = \{\{x, \neg y, z\}, \{\neg z\}\}$ Unit propagation: setze z=0



Head- und Tail-Listen:

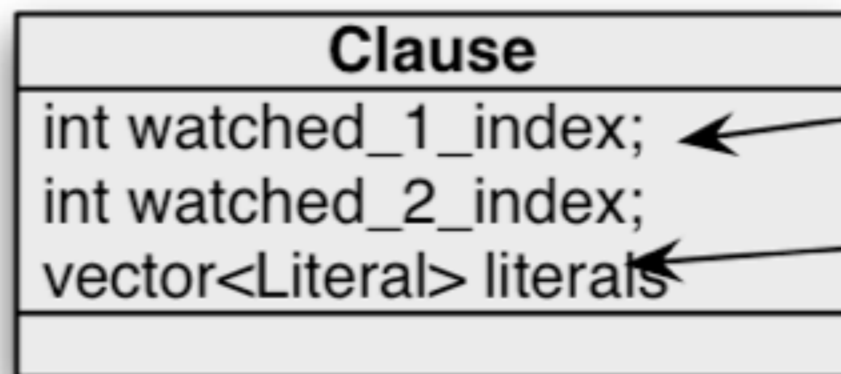
$$F = \{\{x, \neg y, z\}, \{\neg z\}\}$$ Unit propagation: setze z=0



Head/Tail-Listen:

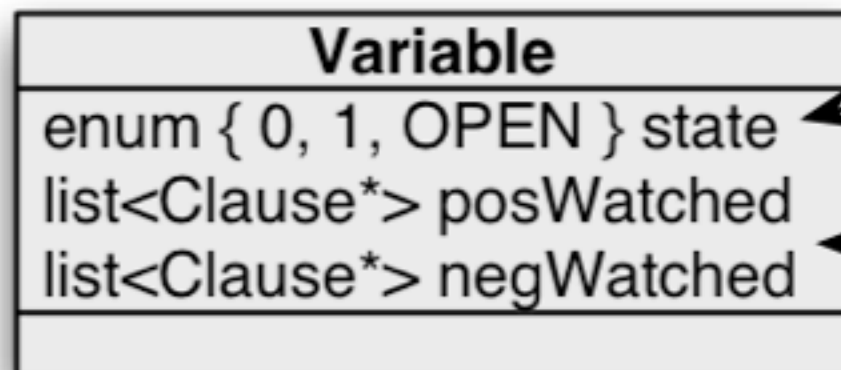
- Positiv: Schnellere Unit-Propagation
- Negativ: Backtracking wird komplizierter (Head- und Tail-Listen müssen wiederhergestellt werden)
- Weitere Verbesserung: Watched Literals
  - Anstelle von Head/Tail-Indizes: 2 Watched Literals pro Klausel
  - Watched Literals sind zwei *beliebige* (verschiedene) offene Literale
  - Beim Backtracking: kein Update der Datenstruktur erforderlich
  - Erstmals implementiert in **zChaff** (Moskewicz *et al*., 2001)

Watched Literals: Datenstrukturen



watched literals
(indices in literal vector)

literals of the clause

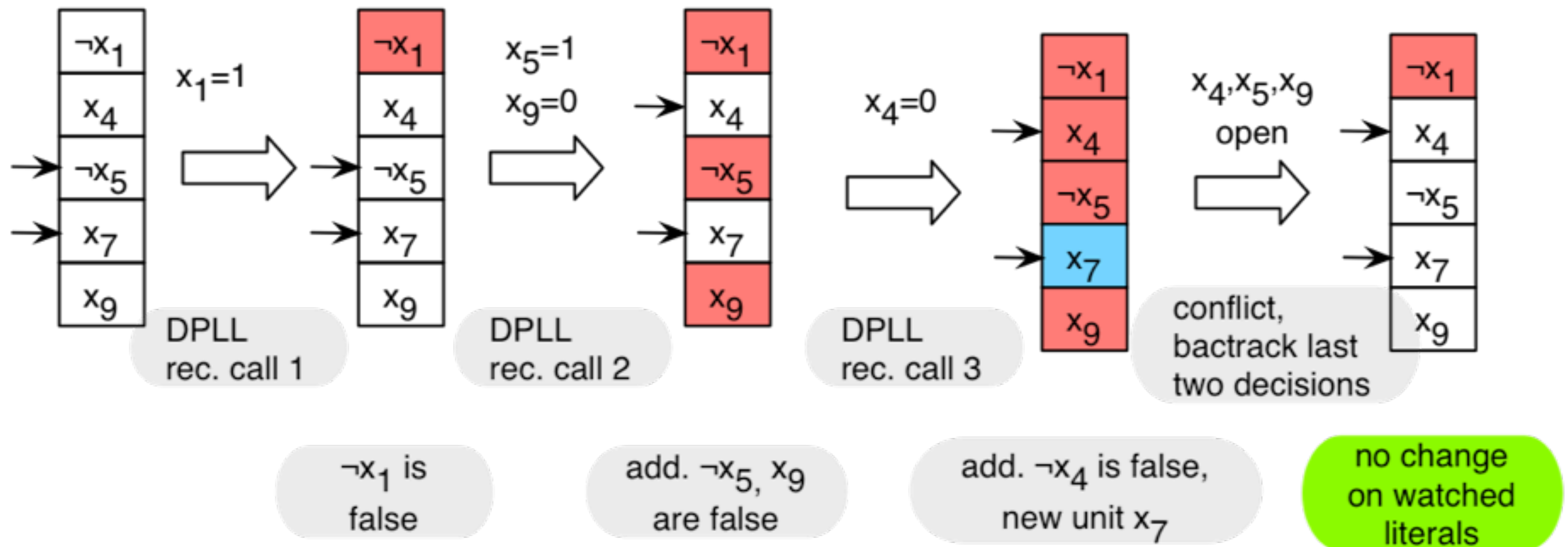


assignment state

pointers to clauses, in which variable
is watched (positively / negatively)

Moskewicz *et al*. (2001)

Watched Literals: Beispiel



Literalauswahl-Strategien

KIT Karlsruhe Institute of Technology

- Welches Literal soll im Fallunterscheidungs-Schritt ausgewählt werden?
  - Größe des Suchraums (und damit die Laufzeit des DPLL-Algorithmus) kann sehr stark von der Literalauswahl-Strategie abhängen.
  - Strategie stark problemabhängig; keine allgemeine "beste" Strategie.
- Ideen im Zhg. mit Auswahlheuristik:
  1. Maximale Vereinfachung; z.B. maximiere die Anzahl der subsumierten (d.h. zu „löschenden“) Klauseln
  2. Versuche, handhabbare Teilklasse von SAT zu erreichen; z.B. 2-SAT, Horn-SAT, nur positive Klauseln
  3. Basierend auf Konfliktanalyse bzw. Klausel-Lernen (Präferenz für Literale in kürzlich gelernten Klauseln)

Literalauswahl-Strategien

- **MOM** (maximum occurrences in minimal clauses):

  maximiere $(occ_2(l) + occ_2(\neg l)) \cdot 2^\alpha + occ_2(l) \cdot occ_2(\neg l)$

  (wobei α eine „genügend große“ Zahl ist)

- **SATO**: erstelle Testmenge der k kürzesten positiven Klauseln und wähle ein Literal, das $(pos_occ(l)+1)(neg_occ(l)+1)$ maximiert.

  (wobei $pos_occ(l)$ ($neg_occ(l)$) die Anzahl der positiven (negativen) Vorkommen von Literal l im Gesamtproblem bezeichnet)

- **VSIDS** (variable state independent decaying sum)
  - Berechne Bewertung (*score*) für jedes Literal.
  - Initiale Bewertung ist die Anzahl der Literalvorkommen.
  - Erhöhe für jede gelernte Klausel die Bewertung aller Literale der Klausel um einen konstanten Betrag c.
  - Teile periodisch alle Bewertungen durch einen Faktor f.
  - Wähle Literal mit dem höchsten Score.