Karlsruhe Institute of Technology

Entscheidungsverfahren mit Anwendungen in der Softwareverifikation

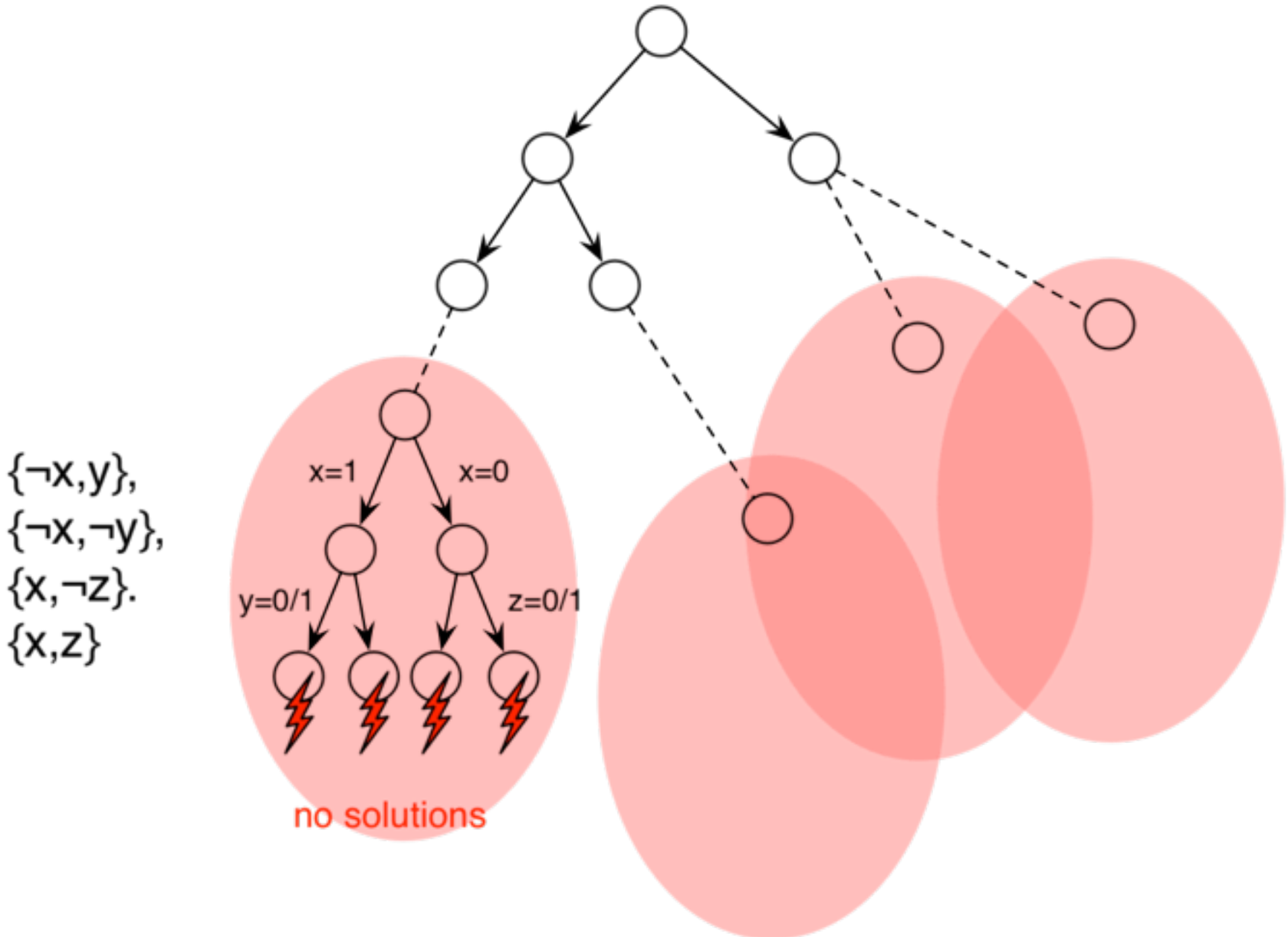
II: Aussagenlogische Erfüllbarkeit

Dr. Stephan Falke
Dr. Carsten Sinz

Institut für Theoretische Informatik

29.04.2013

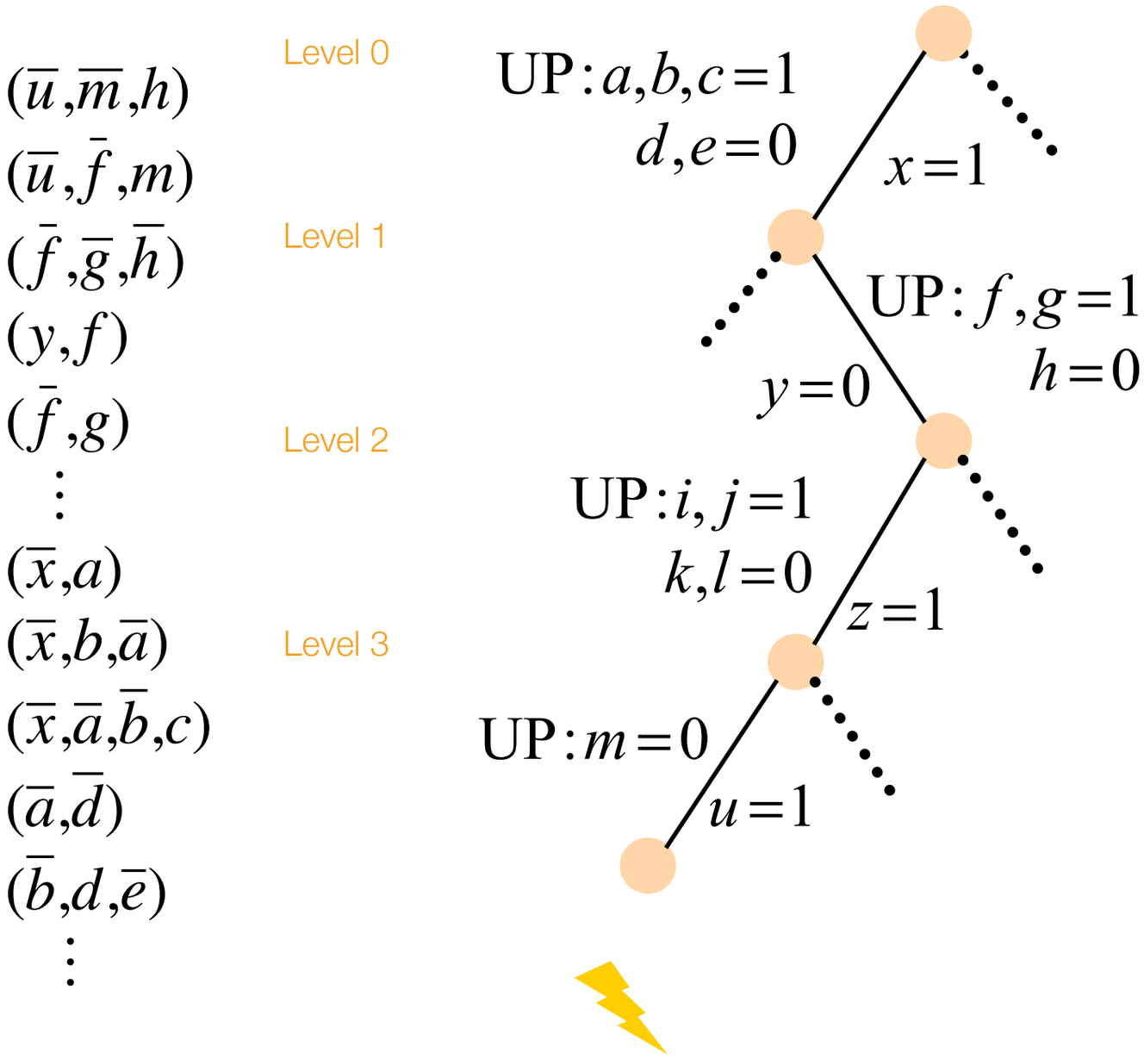
Konfliktanalyse: Motivation



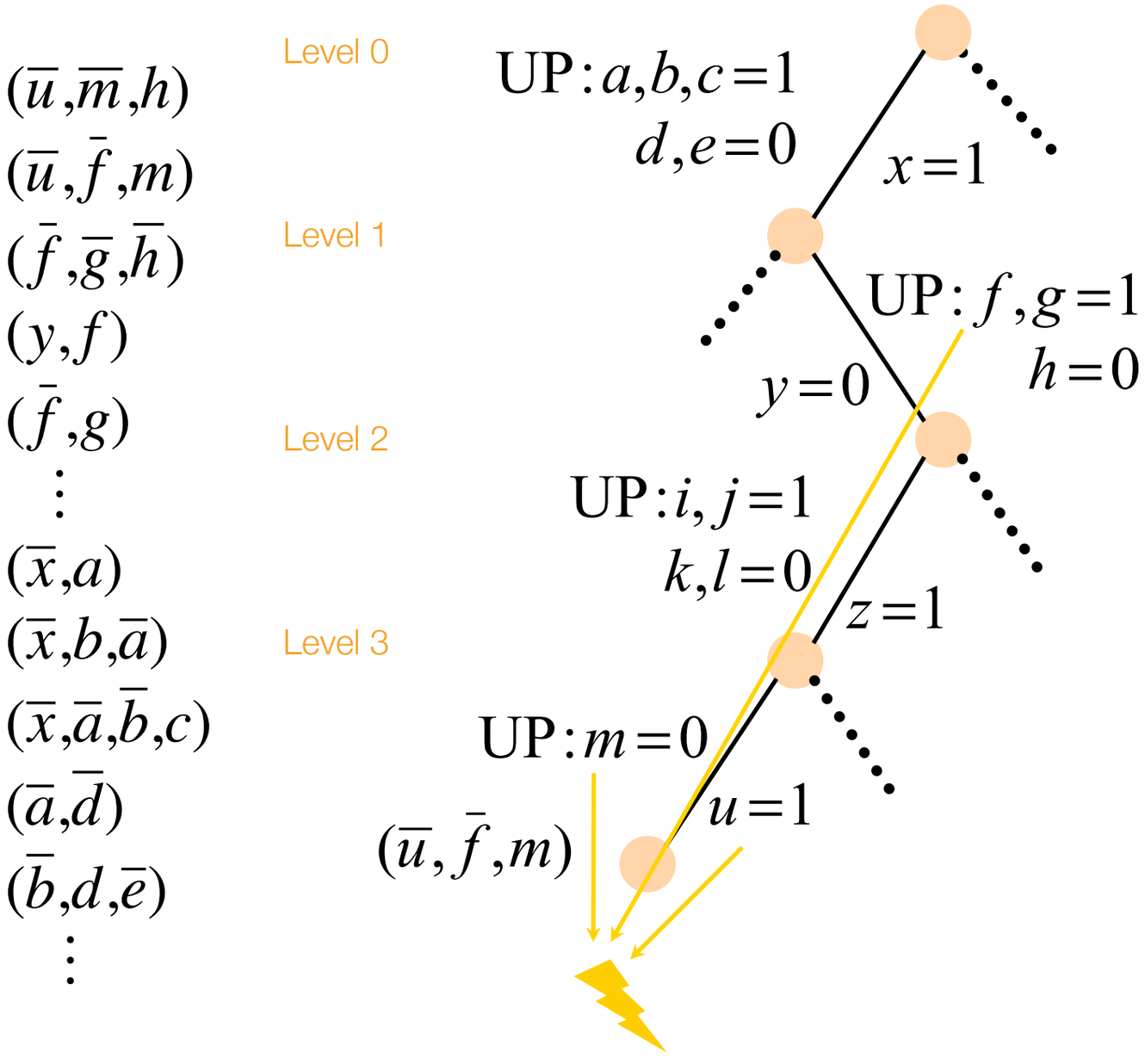
Konfliktanalyse & Klausel-Lernen

- Versuche wiederholte Untersuchung von Teilen des Suchraums ohne Lösungen zu vermeiden.
  - Dies kann eine „schlechte“ Variablenauswahl-Heuristik kompensieren
- Methode: finde schwächste Annahme (weakest precondition), unter der ein Widerspruch auftritt.
  - Jedes für die Fallunterscheidung gewählte Literal gilt als „atomarer“ Grund.
  - Finde minimal erforderliche Bedingung (d.h. kleinste Literalmenge), die denselben Konflikt hervorruft.
- Klausel-Lernen wird auch als „no-good-learning“ bezeichnet (CSP).

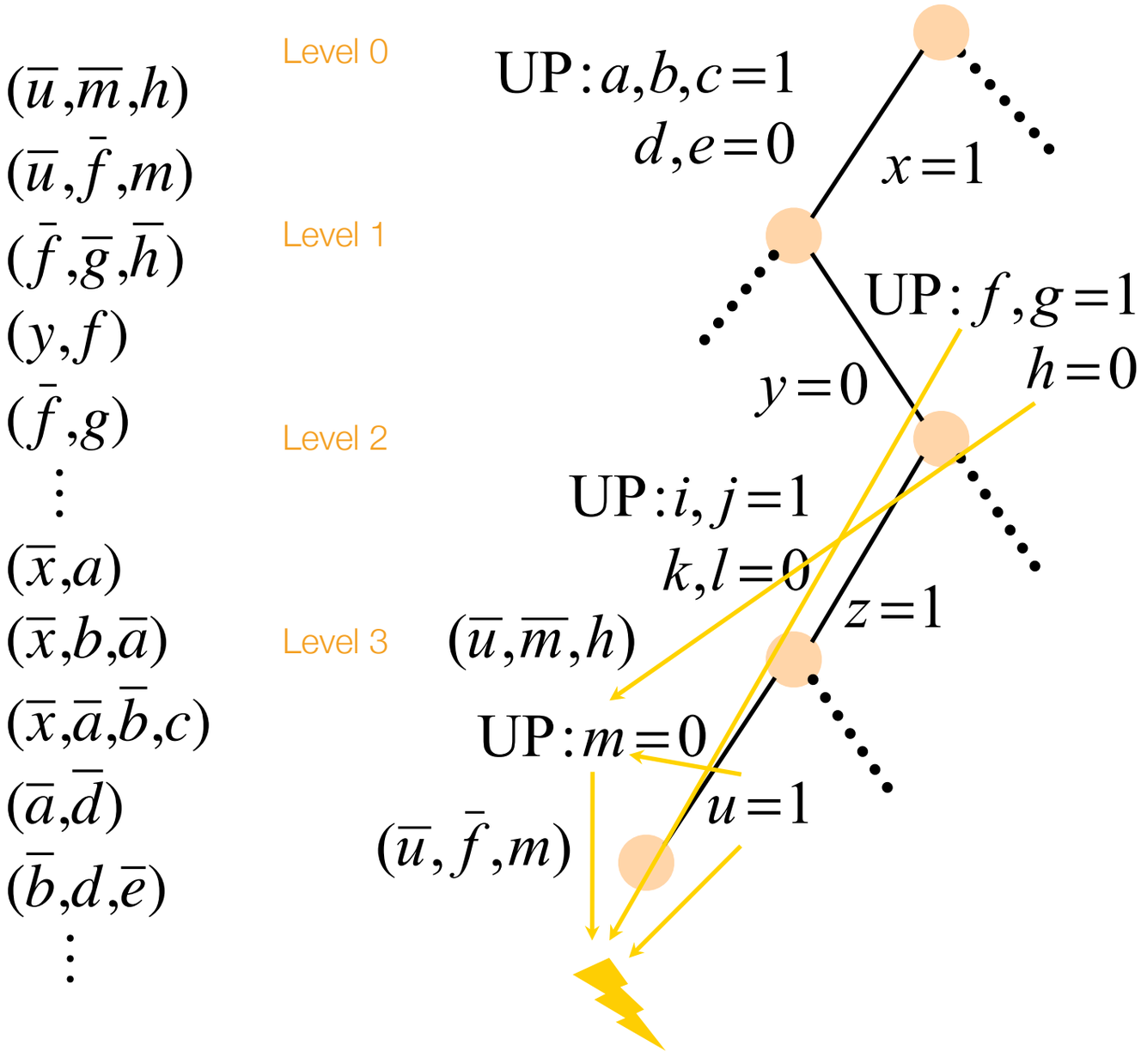
Konfliktklausel-Generierung (Marques-Silva, Sakallah, 1996)



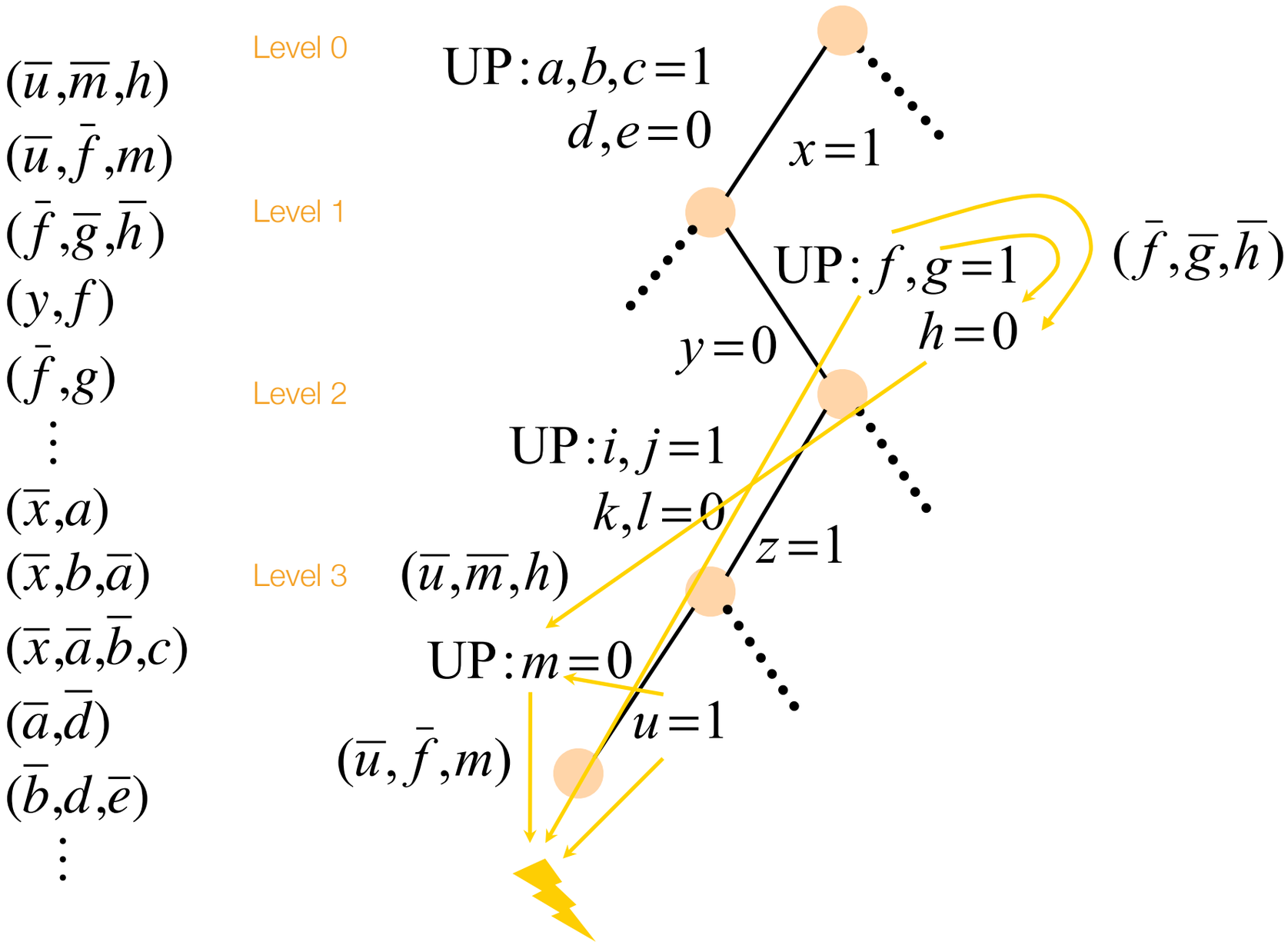
Konfliktklausel-Generierung (Marques-Silva, Sakallah, 1996)



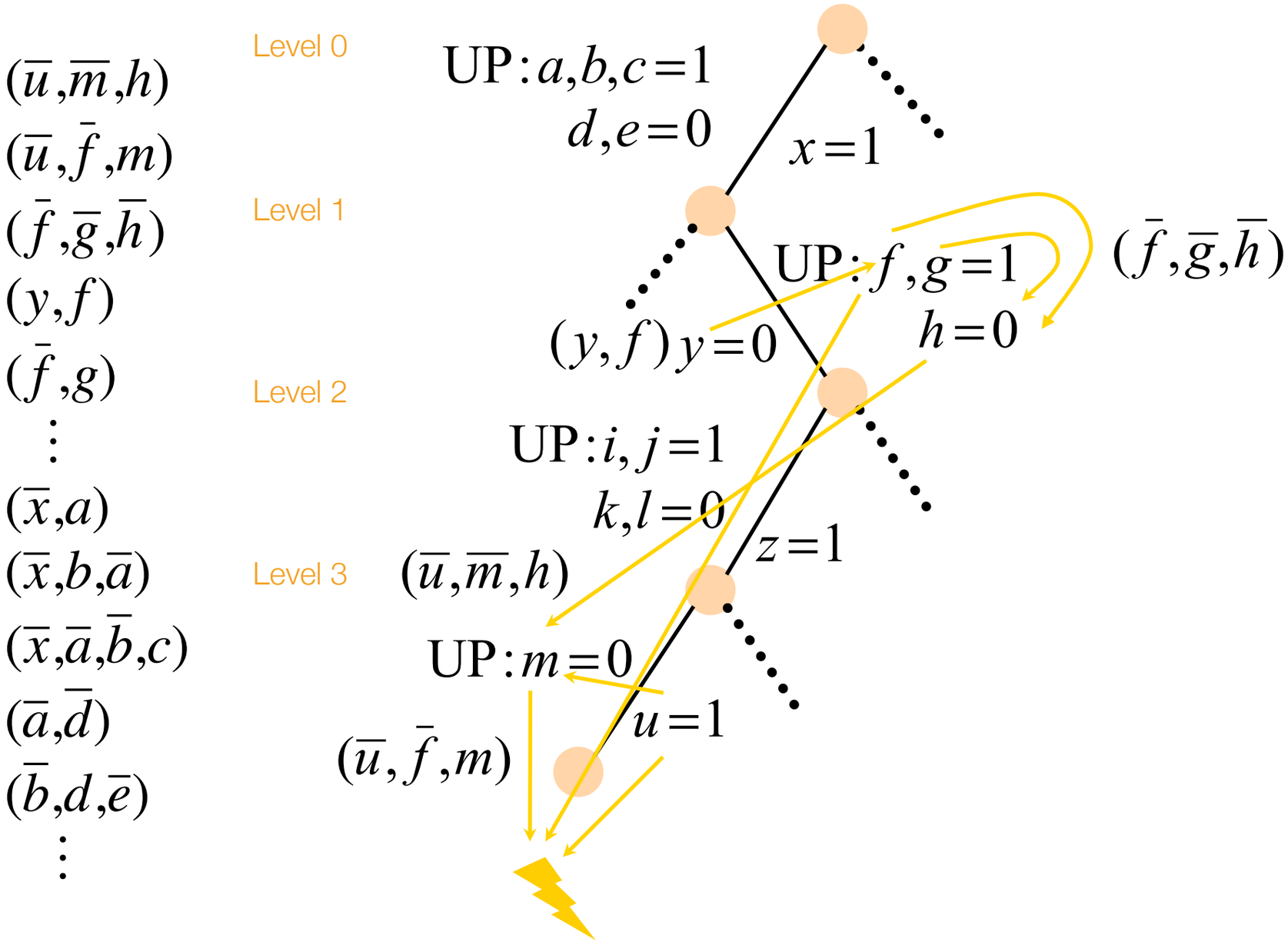
Konfliktklausel-Generierung (Marques-Silva, Sakallah, 1996)



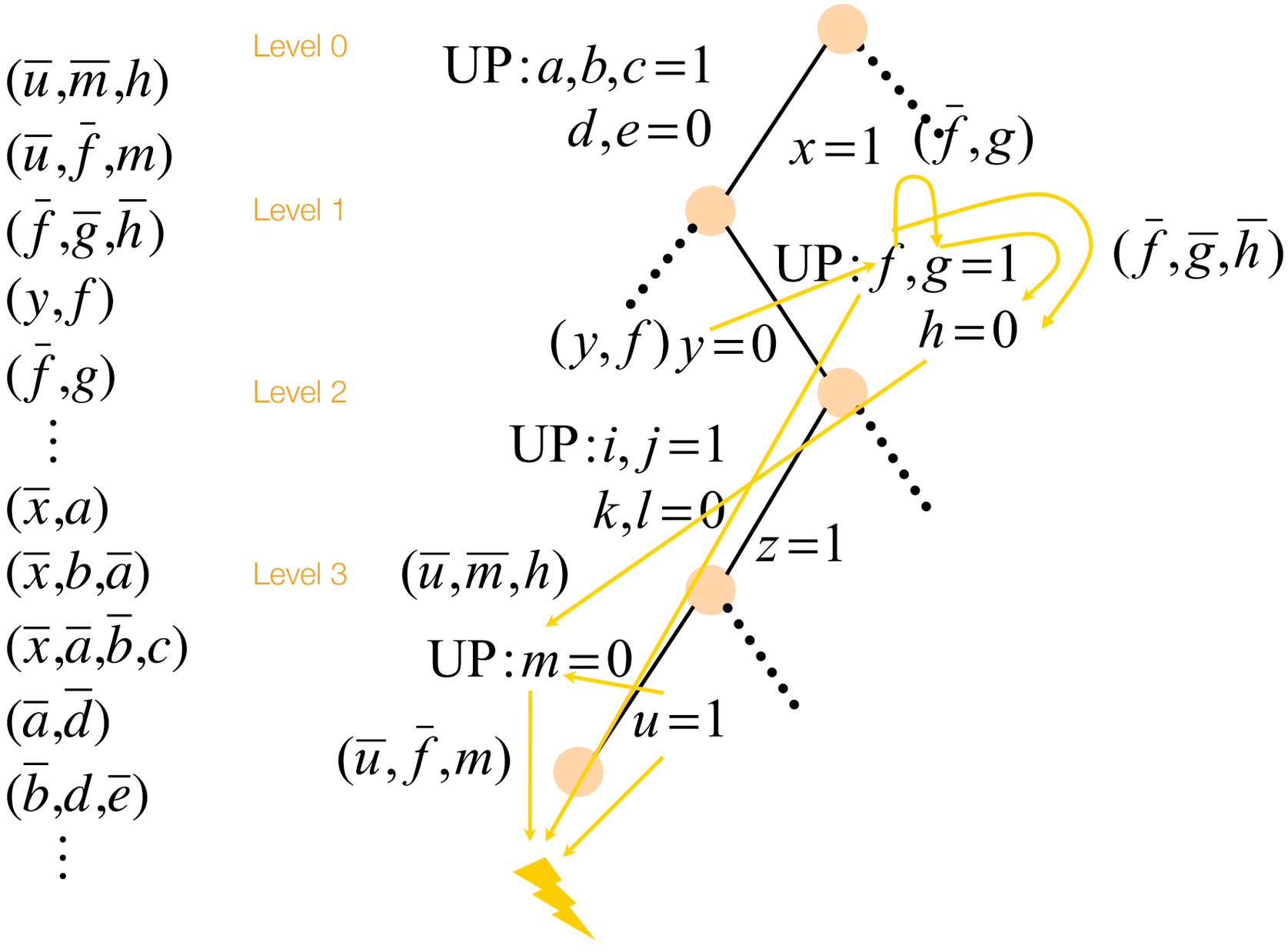
Konfliktklausel-Generierung (Marques-Silva, Sakallah, 1996)



Konfliktklausel-Generierung (Marques-Silva, Sakallah, 1996)



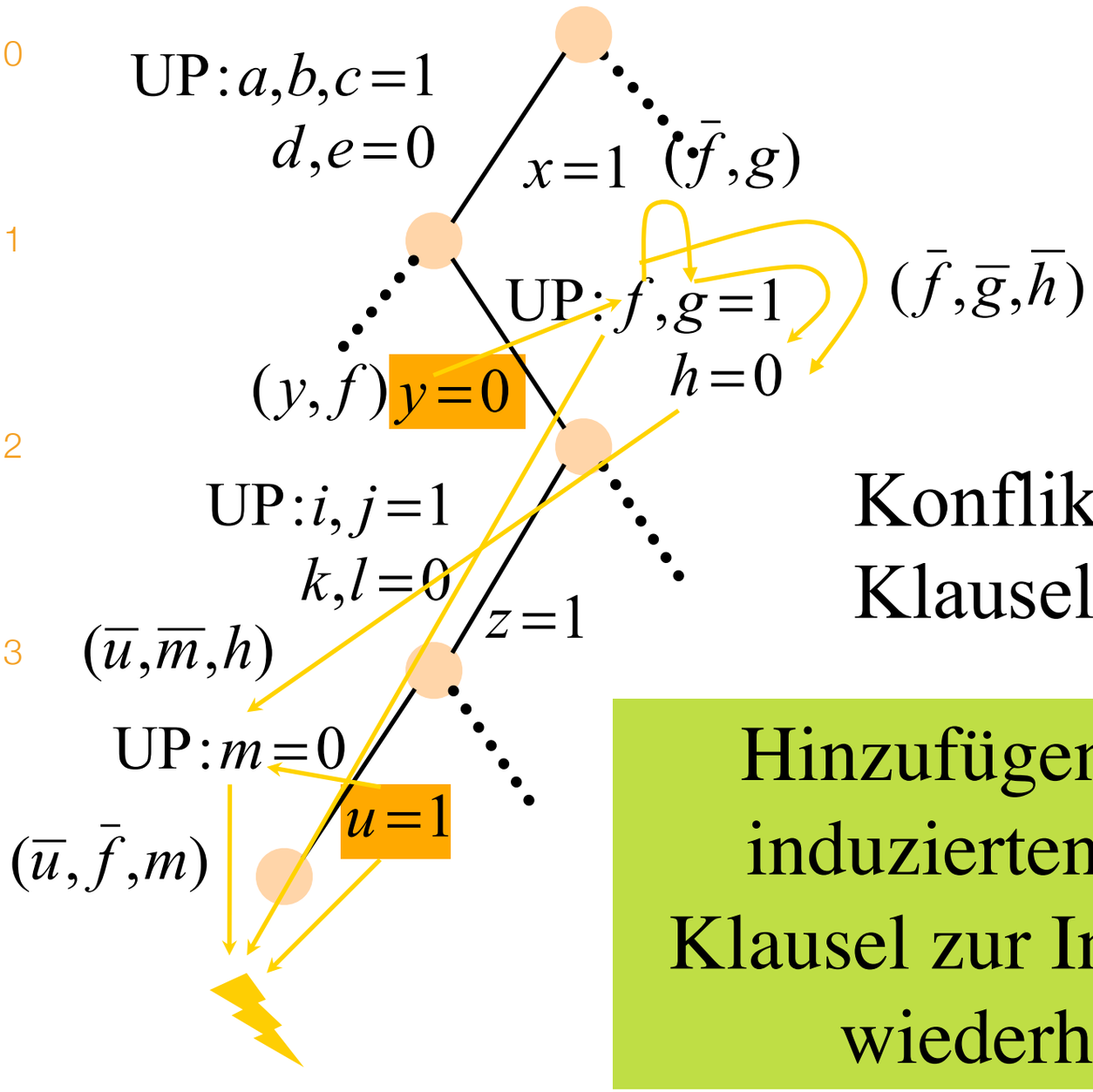
Konfliktklausel-Generierung (Marques-Silva, Sakallah, 1996)



Konfliktklausel-Generierung (Marques-Silva, Sakallah, 1996)

$(\bar{u},\bar{m},h)$
$(\bar{u},\bar{f},m)$
$(\bar{f},\bar{g},\bar{h})$
(y,f)
$(\bar{f},g)$
$\vdots$
$(\bar{x},a)$
$(\bar{x},b,\bar{a})$
$(\bar{x},\bar{a},\bar{b},c)$
$(\bar{a},\bar{d})$
$(\bar{b},d,\bar{e})$
$\vdots$

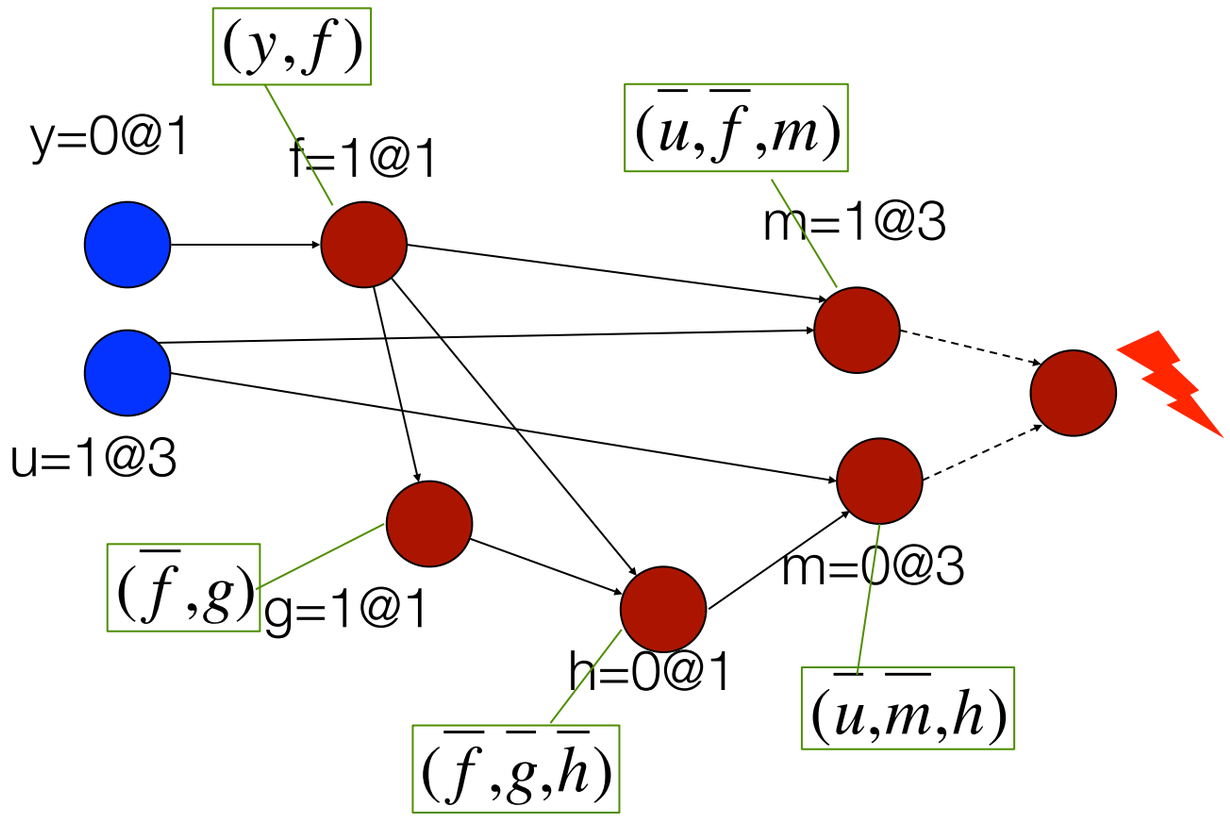
Level 0
Level 1
Level 2
Level 3



Konflikt-induzierte Klausel: $(y,\bar{u})$

Hinzufügen der Konflikt-induzierten ("gelernten") Klausel zur Instanz vermeidet wiederholte Suche.

Konfliktgraph (I)



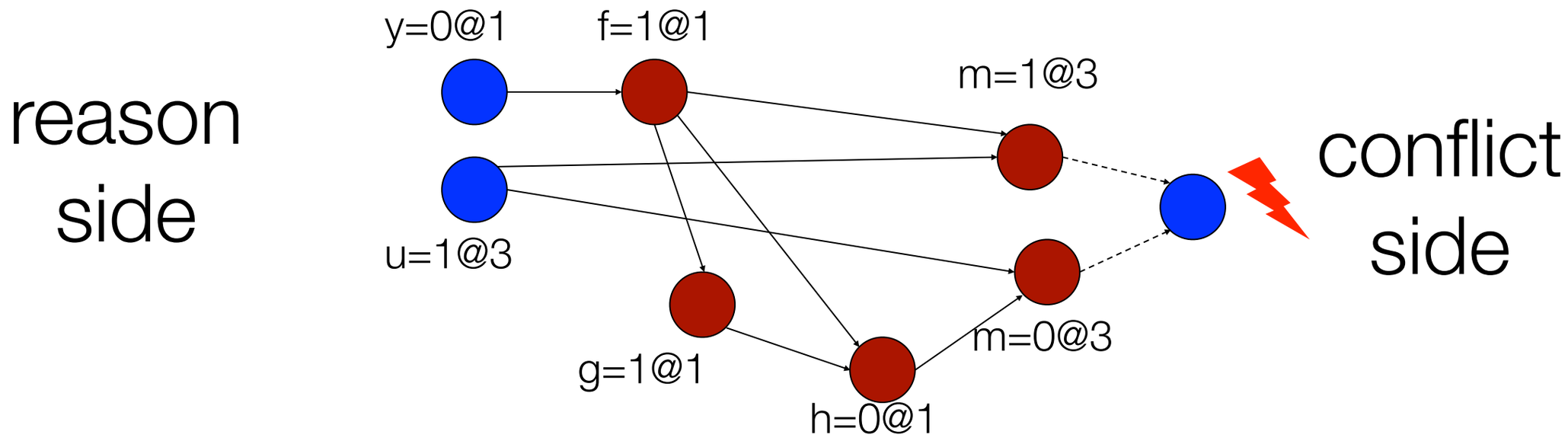
y=0 (auf Level 1) impliziert:
f=1, g=1, h=0

u=1 (auf Level 3) impliziert:
m=0, m=1, Widerspruch

$(\bar{u},\bar{m},h)$
$(\bar{u},\bar{f},m)$
$(\bar{f},\bar{g},\bar{h})$
(y,f)
$(\bar{f},g)$
$\vdots$

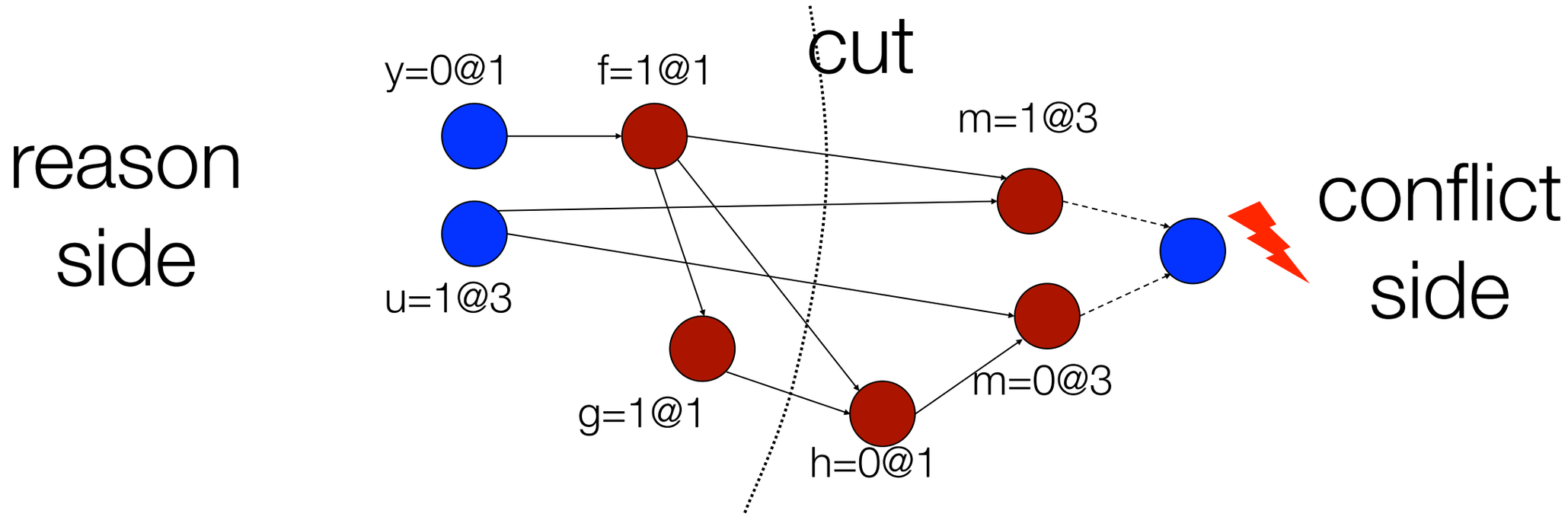
$(\bar{x},a)$
$(\bar{x},b,\bar{a})$
$(\bar{x},\bar{a},\bar{b},c)$
$(\bar{a},\bar{d})$
$(\bar{b},d,\bar{e})$
$\vdots$

Konfliktgraph (II)



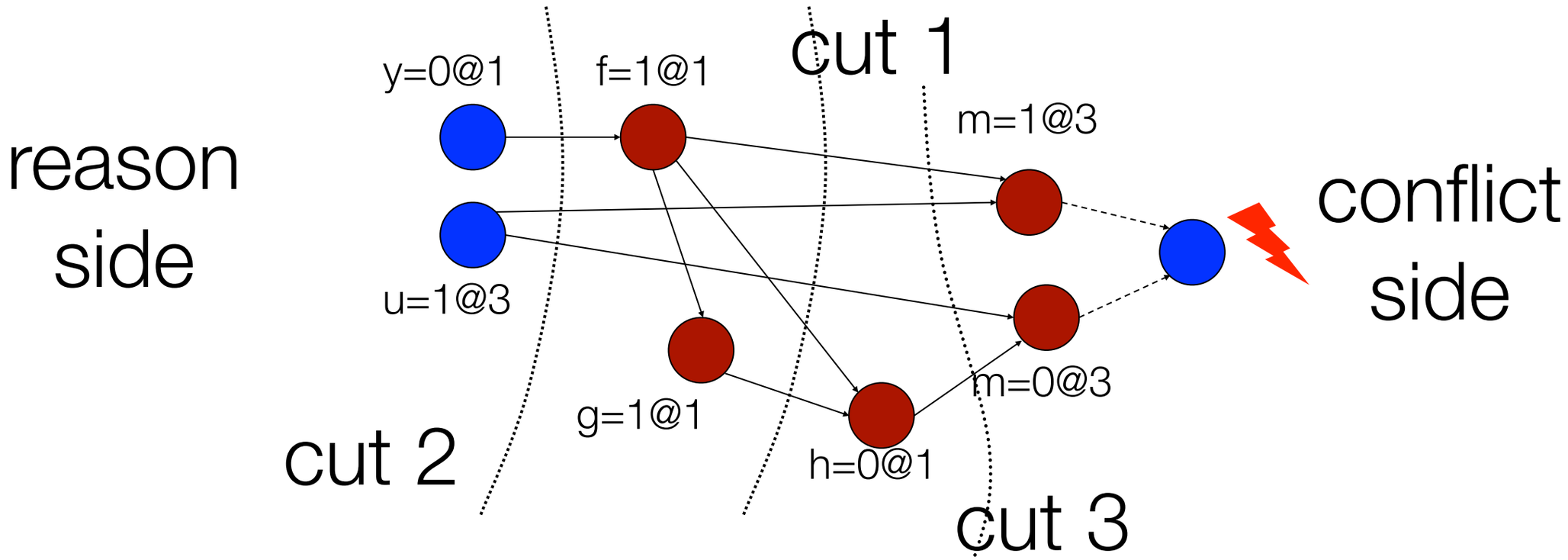
- Jede Konfliktklausel ist durch einen Cut durch den Konfliktgraphen bestimmt (Knoten-Partitionierung in *reason side* und *conflict side*)
  - *Decision nodes* sind auf der *reason side.*
  - *Widerspruch* ist auf der *conflict side.*
  - Konfliktklausel ergibt sich aus Negation aller Literale, von denen eine

Konfliktgraph (II)



- Jede Konfliktklausel ist durch einen Cut durch den Konfliktgraphen bestimmt (Knoten-Partitionierung in *reason side* und *conflict side*)
  - *Decision nodes* sind auf der *reason side.*
  - *Widerspruch* ist auf der *conflict side.*
  - Konfliktklausel ergibt sich aus Negation aller Literale, von denen eine

Konfliktgraph (III)



Cut 1: Konfliktklausel {¬f, ¬u, ¬g}

Cut 2: Konfliktklausel {y, ¬u}

Cut 3: Konfliktklausel {¬f, ¬u, h}

Jede Konfliktklausel (oder auch mehrere) kann bei einem Konflikt zur Klauselmenge hinzugefügt werden.

Welche Konfliktklausel hinzufügen?

- Decision clause
  - enthält nur decision nodes (cut 2)
- First new cut clause
  - enthält nur sich widerspr. Literale (komplementäres Literalpaar) auf conflict side (cut 3)
- 1UIP clause
  - UIP (unique implication point): Knoten, über den alle Pfade von *conflict side* zu *reason side* laufen
    - Anmerkung: alle decision nodes sind UIPs
  - 1: Ausgehend von *first new cut clause*, gehe „zurück“ im Graph solange Knoten-Level nicht kleiner wird, bis UIP erreicht.
- 1UIP wird in den meisten Solvern verwendet.

CDCL: Conflict-Driven Clause Learning

```
boolean CDCL
{
    forever {
        do {
            ok = propagate_units(); // returns false iff conflict occurred
            if (!ok) {              // conflicting assignment
                generate_and_add_conflict_clause();
                new_level = backtrack();
                if (new_level < 0) return false;
            }
        } while(!ok);
        if no more open variables return true;
        decide();               // select open literal, assign value to it
    }
}
```

Konfliktklausel-basierte Variablenauswahl-Heuristik: VSIDS

- VSIDS:
  - Ordne jeder Variablen einen *score* zu.
    - Initial 0 (oder Anzahl der Vorkommen)
  - Wird eine Konfliktklausel *C* erzeugt, so wird der *score* eines jeden Literals aus *C* um einen Betrag *a* erhöht.
  - Nach *n* Konflikten werden alle *scores* durch einen konstanten Faktor *f* geteilt (*ageing*).
  - Die Heuristik wählt immer das Literal mit dem höchsten *score*.

Weitere Techniken im SAT-Solving

- **Restarts**
  - Breche Suche nach einer vorgegebenen Anzahl von Schritten k ab und starte diese neu
- **Phase memorization**
  - Wähle nach Backtracking in `decide()` gleiche Phase (d.h. gleiches „Vorzeichen“) wie bei letzter Zuweisung
- **Preprocessing**
  - Vereinfache Formel in einem Vorverarbeitungsschritt

SAT-Codierung: Einführendes Beispiel

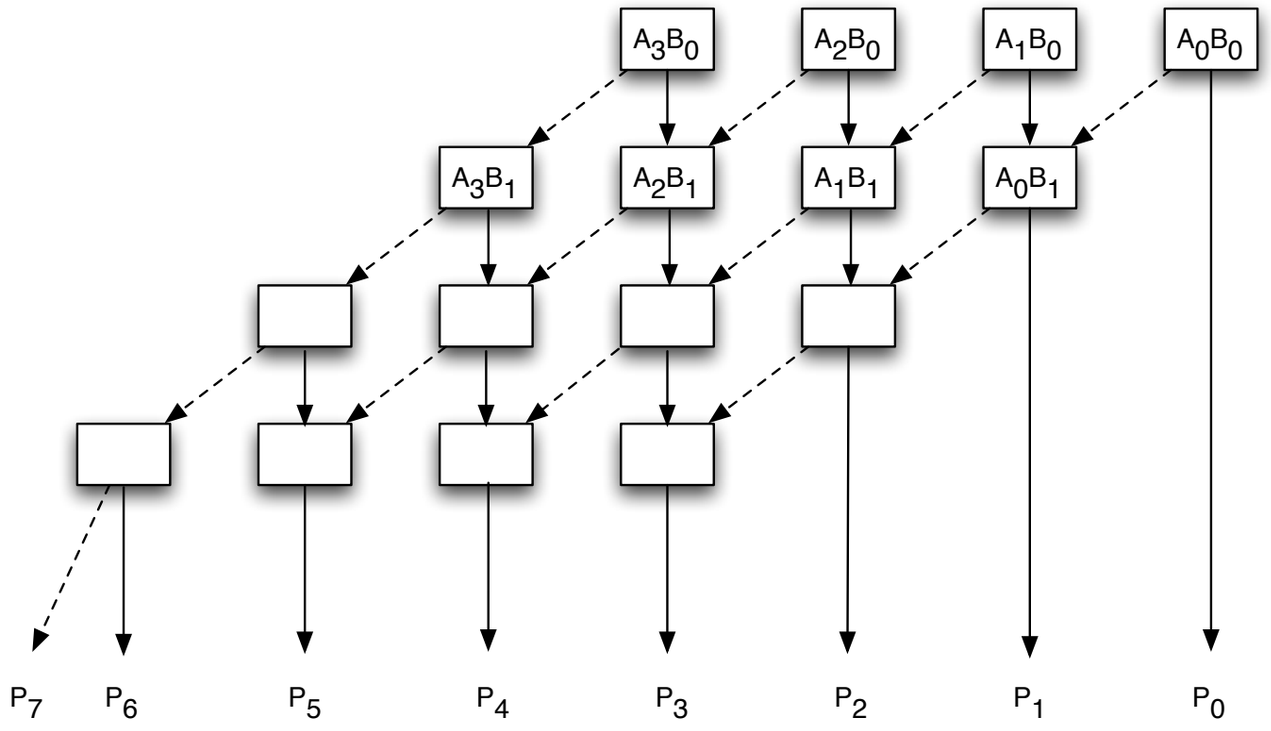
- Problem: Ganzzahl-Faktorisierung
  - Gegeben eine ganze Zahl p, gibt es eine Zerlegung $p = x \cdot y$ mit $1 < x, y < p$?
- Programm:

```
void factorize(unsigned int x, unsigned int y) {
  unsigned int p = 1023;
  if (!(x > 1 && y > 1 && x < p && y < p))
    return;
  if (p == x * y)
    assert(0); // error
}
```

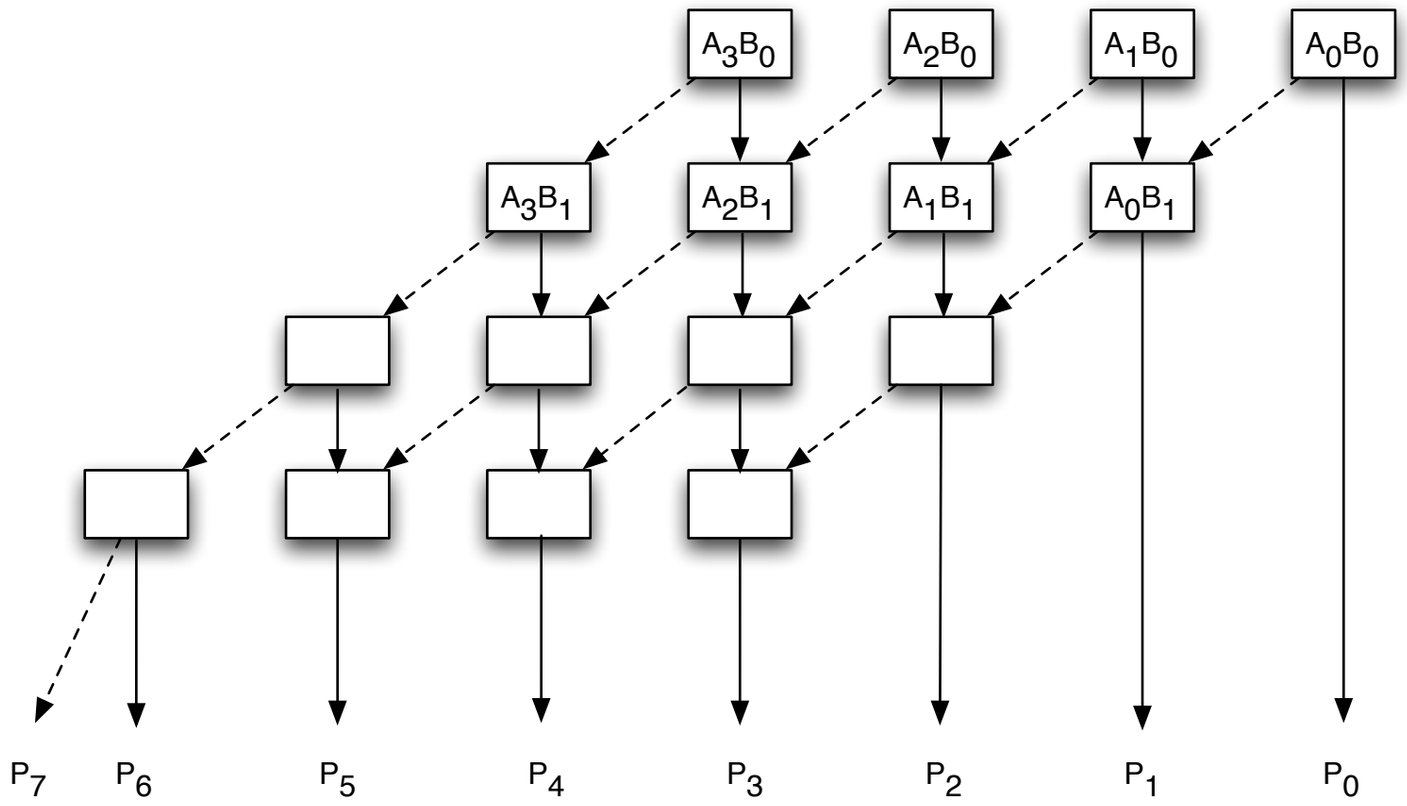
- Wenn es eine Faktorisierung von p gibt, ist die „error“-Zeile erreichbar
- Wie können wir das in SAT codieren?

Modellierung in Hardware

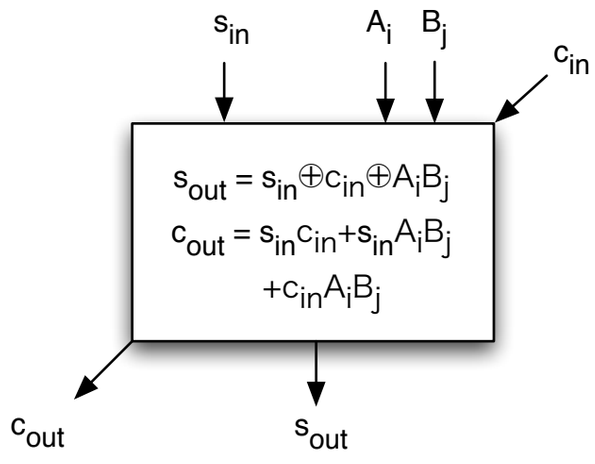
- **Formel zusammengesetzt aus:**
  - n-Bit Multiplizierer zur Multiplikation `x*y`
  - Ausgabe wird auf `p` festgehalten
  - Zusätzliche Klauseln zur Einschränkung der Eingabe (`1 < x, y < p`)
  - Bit-Vektoren für `x` und `y` sind Eingabe-Variablen
- **Multiplizierer (4 Bit):**

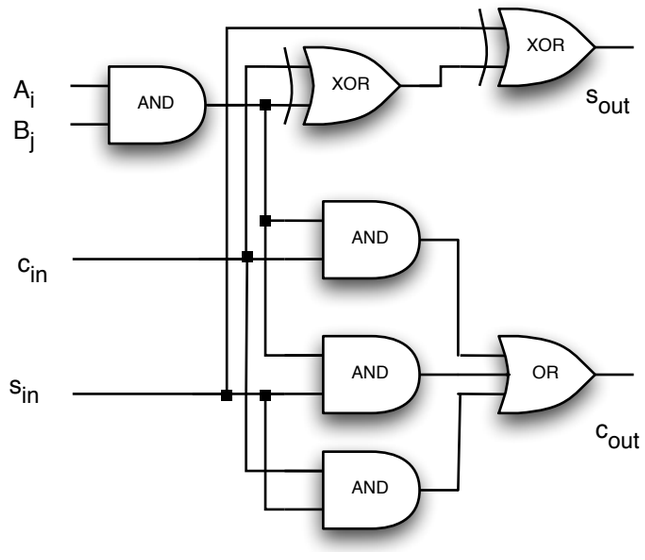


Hardware-Multiplizierer



1-Bit-Multiplizierer mit Carry/Sum-In/Out:





Tseitin-Codierung

- Regeln zur Tseitin-Codierung (mit Optimierungen von Plaisted-Greenbaum):

$$\mathcal{T}(F) = d_F \wedge \mathcal{T}^1(F)$$

$$\mathcal{T}^p(F) = \begin{cases} \mathcal{T}^p_{\mathsf{def}}(F) \wedge \mathcal{T}^p(G) \wedge \mathcal{T}^p(H), & if\ F = G \circ H\ \ and \circ \in \{\wedge, \vee\} \\ \mathcal{T}^p_{\mathsf{def}}(F) \wedge \mathcal{T}^{p \oplus 1}(G), & if\ F = \neg G \\ \top, & if\ F \in \mathcal{V} \end{cases}$$

$$\mathcal{T}^1_{\mathsf{def}}(F) = \begin{cases} (\neg d_F \vee d_G) \wedge (\neg d_F \vee d_H), & if\ F = G \wedge H \\ (\neg d_F \vee d_G \vee d_H), & if\ F = G \vee H \\ (\neg d_F \vee \neg d_G), & if\ F = \neg G \end{cases}$$

$$\mathcal{T}^0_{\mathsf{def}}(F) = \begin{cases} (d_F \vee \neg d_G \vee \neg d_H), & if\ F = G \wedge H \\ (d_F \vee \neg d_G) \wedge (d_F \vee \neg d_H), & if\ F = G \vee H \\ (d_F \vee d_G), & if\ F = \neg G \end{cases}$$