

# Computergestützte Verifikation

27.5.03

(1. Systeme 2. Spezifikationen 3. explizites Model Checking)

## **4. Symbolisches Model Checking**

4.1 CTL Model Checking mit Binary Decision Diagrams

(4.2 SAT-basiertes Model Checking)

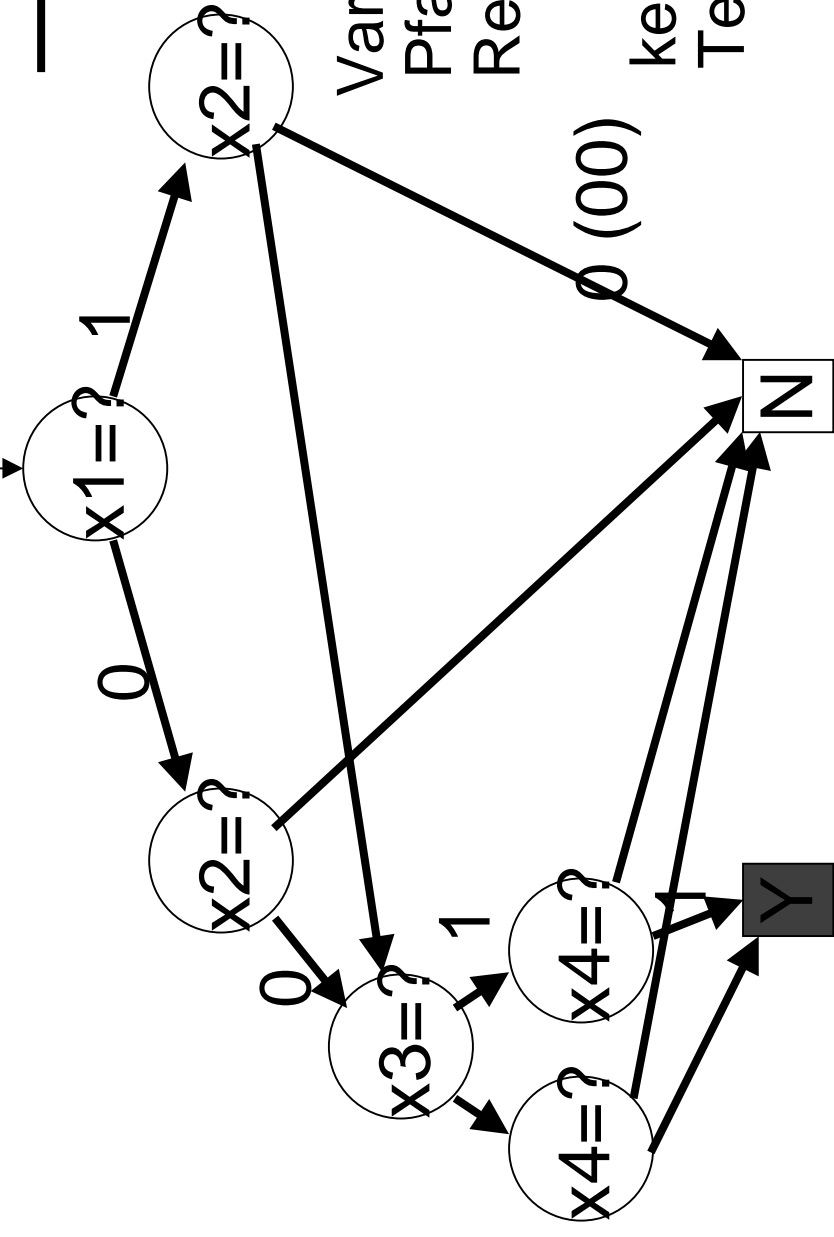
(4.3 Tools)

# Ordered Binary Decision Diagram

## Diagram

→ 0

→ 1



Variablen auf jedem Pfad in gleicher Reihenfolge

keine äquivalenten Teilbäume

Keine redundanten Knoten

0011 ∈ M

1000 ∉ M

# Operationen auf BDD

APPLY(op, f1, f2)

f1, f2 n-stellig

$f(x_1, \dots, x_n) := f_1(x_1, \dots, x_n) \text{ op } f_2(x_1, \dots, x_n)$

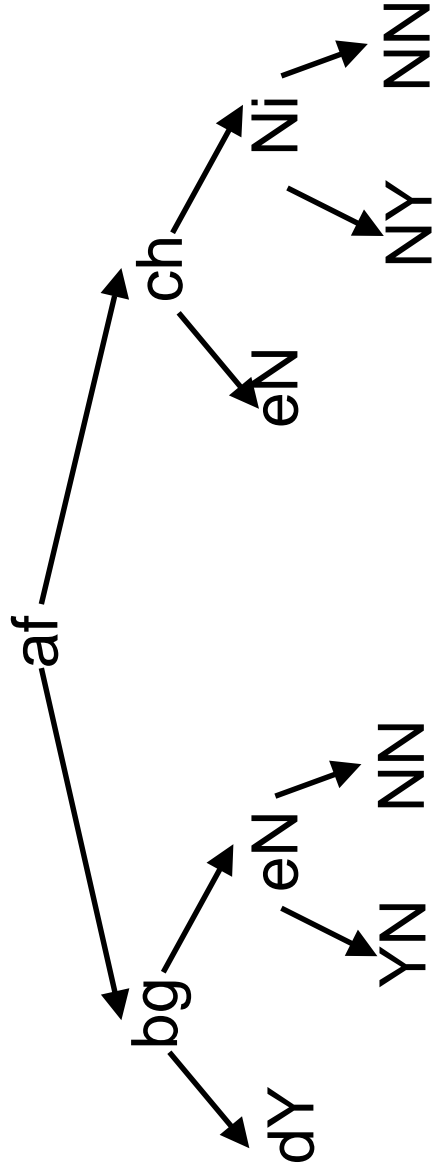
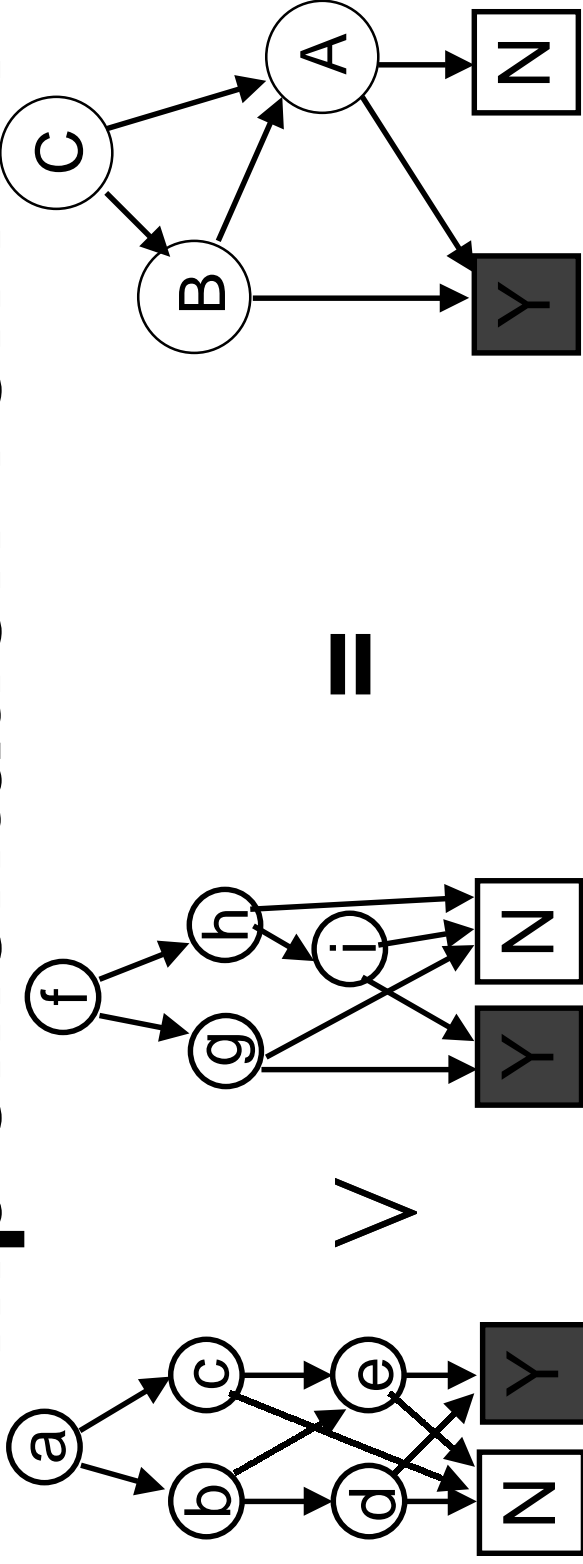
“Shannon-Expansion”

$$f(x_1, \dots, x_n) \Leftrightarrow (x_1 \wedge f(1, x_2, \dots, x_n)) \vee (\neg x_1 \wedge f(0, x_2, \dots, x_n))$$

$$x_1 \wedge (f_1(1, x_2, \dots, x_n) \text{ op } f_2(1, x_2, \dots, x_n)) \vee$$

$$\neg x_1 \wedge (f_1(0, x_2, \dots, x_n) \text{ op } f_2(0, x_2, \dots, x_n))$$

# Implementation von APPLY



Y	N	A
Y	A	B
B	A	C

dY	YN	NN	eN	bg	NY	Ni	ch	af
Y	Y	N	A	B	Y	A	A	C

$O(|BDD1| |BDD2|)$

## 4.1.3 CTL Model Checking

Arbeiten mit  $\text{SAT}_\phi = \{s \mid s \models \phi\}$  - repräsentiert als BDD

und  $T(s,s')$  – der Zustandsübergangsrelation, auch als BDD

$T(s,s') = 1$  gdw.  $(s,s')$  in  $E$

$T(s,s')$  kann aus einer Systembeschreibung als nextstate-Funktion generiert werden

System erfüllt  $\phi$  gdw.  $I \subseteq \text{SAT}_\phi$

# Model Checking

## EU

## EG

$Z := \text{SAT}_{\psi}$   
do  
     $Z := Z \vee (\text{SAT}_{\phi} \wedge \text{SAT}_{\text{EX}Z})$   
until nothing changes  
 $\text{SAT}_{E(\phi \cup \psi)} := Z$

$Z := \text{SAT}_{\phi}$   
do  
     $Z := Z \wedge \text{SAT}_{\text{EX}Z}$   
until nothing changes  
 $\text{SAT}_{E\phi} := Z$

Berechnen kleinsten Fixpunkt  
eines monoton *wachsenden*  
Mengenoperators

Berechnen größten Fixpunkt  
eines monoton *fallenden*  
Mengenoperators

# Partitionierung der Übergangsrelation

Idee:  $T$  ist meistens Konjunktion Teilformeln

Partitionen kleiner als  $T$ , günstigenfalls auch in der Summe

mindestens: Eine Partition hängt normalerweise nicht von allen Variablen ab, ist also auf jeden Fall flacher als das BDD von  $T$

- Existenzquantifizierung kann wenigstens über einige Partitionen hinweg nach vorn verschoben werden
- Abflachung, also meist Verkleinerung der Zwischen-BDDs

# 4.1.4 Fairness

Arbeiten mit zustandsbasierter Fairness und schwachen Fairnessannahmen

geg: BDDs  $C_1, \dots, C_n$ , stehen für Fairnessannahmen  
GF  $C_1, \dots, GF C_n$

Ansatz: trickreiche Adaption der Fixpunktoperationen

# Faires EG

$\phi \phi \phi \phi \phi \phi \phi \phi \dots$

$\phi \phi \phi \phi \phi \phi \wedge C_i \phi \phi \phi \phi \phi \phi \wedge C_j \phi \phi \phi \phi \phi \phi \wedge C_k \phi \phi \phi \dots$

unfair:

$Z := \text{SAT}_{\phi}$

do

$Z := Z \wedge \text{SAT}_{\text{EX } Z}$

until nothing changes

$\text{SAT}_{\text{EG } \phi} := Z$

fair:

$Z := \text{SAT}_{\phi}$

do

$Z := Z \wedge \bigwedge_{k=1}^n \text{SAT}_{\text{EX } E(\phi \cup (Z \wedge C_k))}$

until nothing changes

$\text{SAT}_{\text{EG } \phi} := Z$

# Andere faire Operatoren

$$E_C(\phi \cup \psi) = E(\phi \cup (\psi \wedge E_C G \text{ true}))$$

$$E_C X \phi = EX(\phi \wedge E_C G \text{ true})$$

Rest: Tautologien

# Letzte Folie zu BDD

1992: Paper “Symbolic model checking  
–  $10^{20}$  States and beyond”

= Durchbruch für die Methode Model Checking

1996: Partitioned transition relations  $\rightarrow 10^{100}$

1999-2000 SAT-basiertes Model Checking  $\rightarrow 10^{1000}$

# 4.2 SAT-basiertes Model Checking

Ansatz: Übersetze das Model Checking Problem in ein aussagenlogisches Erfüllbarkeitsproblem und löse dieses.

- Inhalt
- 4.2.1 Ein effizienter SAT-Solver
- 4.2.2 Noch ein effizienter SAT-Solver
- 4.2.3 LTL Model Checking als SAT-Problem

# Das Problem SAT

geg: aussagenlogischer Ausdruck  $\phi$

Frage: Ist  $\phi$  erfüllbar?

NP-vollständig

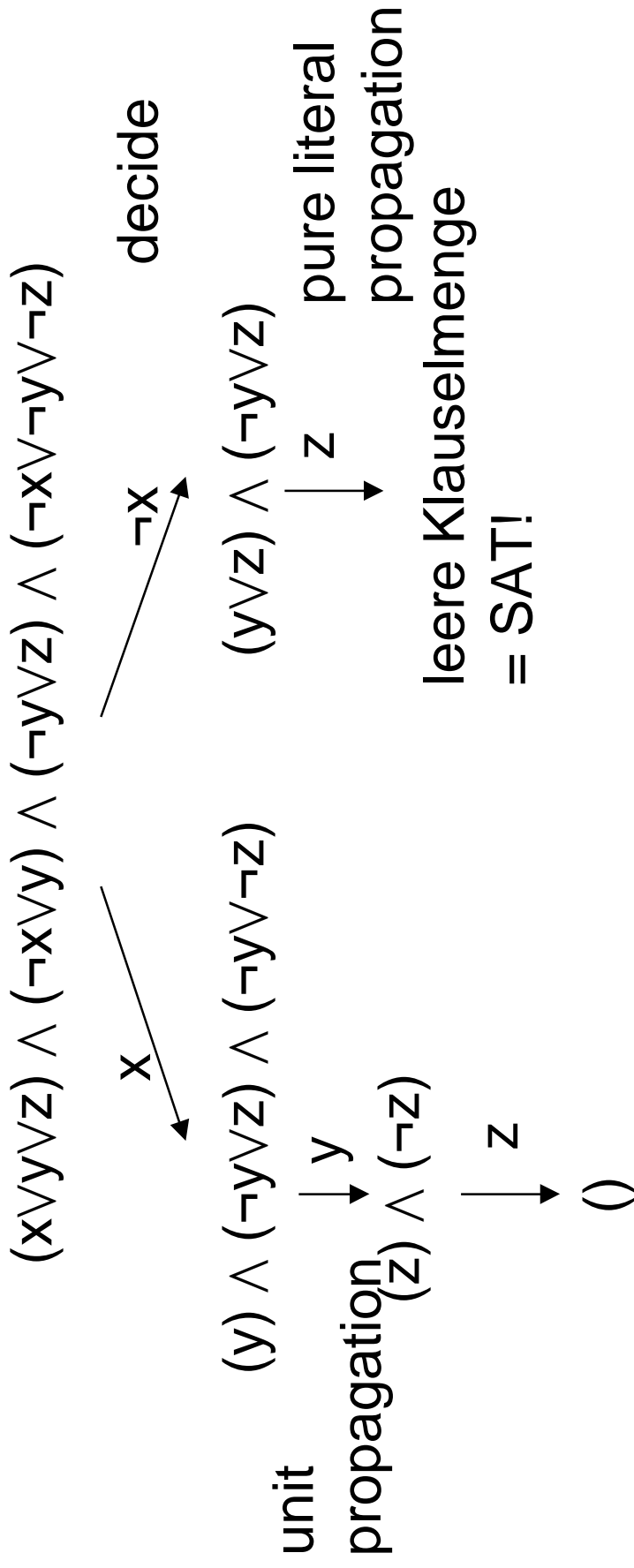
Viele Algorithmen setzen  $\phi$  in bestimmter Form voraus,  
meist CNF (Klauselmenge) (  $\rightarrow$  4.2.1).

Manche Algorithmen arbeiten auf *beliebigen* Ausdrücken  
(  $\rightarrow$  4.2.2)

# 4.2.1 SAT-Solver für CNF

(Suche nach erfüllender Belegung)

Ausgangspunkt: Algorithmus von Davis-Putnam aus den 60ern



→ Backtracking zur letzten offenen Entscheidung

# Davis-Putnam-Algorithmus

```
DP(K)      K – Klauselmeng  
IF K =  $\emptyset$  THEN RETURN SAT  
IF ()  $\in$  K THEN RETURN UNSAT  
IF  $\kappa = (\dots \vee 1 \vee \dots) \in K$  THEN RETURN DP(K \ { $\kappa$ })      Tautologie  
IF (0  $\vee \dots \vee 0 \vee [\neg]x_i) \in K$  THEN RETURN DP(K/  $x_i \leftarrow 1$ [0])      unit  
IF K enthält Literal l, aber nicht seine Negation THEN  
    RETURN DP(K \ { $\kappa$  | l  $\in$   $\kappa$ })      pure literal  
choose i  
IF DP(K/  $x_i \leftarrow 1$ ) = SAT THEN RETURN SAT      decide/  
RETURN DP(K/  $x_i \leftarrow 0$ )      backtrack
```

# 1. Trick: Schnelles Finden von Unit Klauseln

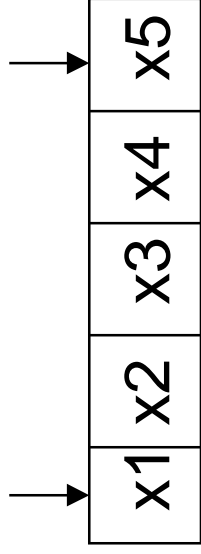
Unit-Klausel = Klausel mit  $n-1$  Literalen auf 0, 1 Literal auf ?

Lösung (z.B. in *chaff*): Pro Klausel 2 Beobachter, die auf ?-Literale zeigen

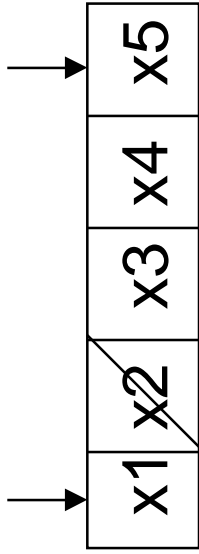
Solange beide Beobachter auf ?-Literale zeigen, wird Klausel nicht angerührt

# Beispiel

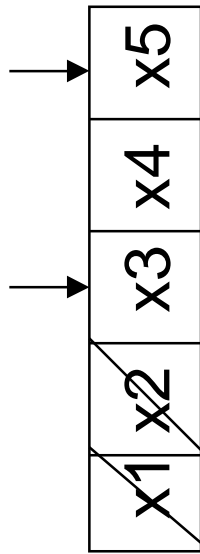
Beim Backtracking bleiben Beobachter, wo sie sind  
 → konst. Zeit



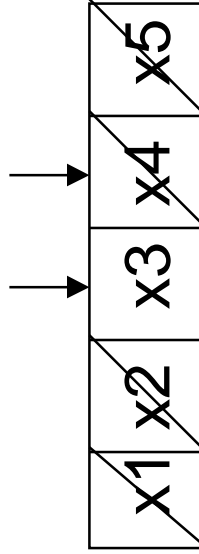
$$\underline{x_2=0}$$



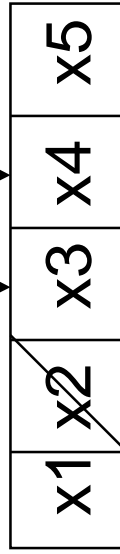
$$\underline{x_1=0}$$



$$\underline{x_{4,5}=0}$$



$$\frac{\text{backtrack}}{x_4, x_5 := ? \quad x_1 := 1}$$



Beobachter “wandern” zu selten gesetzten Literalen  
 → Klausel muß seltener besucht werden  
 → “Lernstrategie”

## 2. Trick: Konfliktanalyse

Konflikt = leere Klausel = Literal, für das sowohl 0 als auch 1 propagiert wird

Idee: “Grund” für den Widerspruch wird explizit als Klausel zur Klauselbasis hinzugefügt “Lernen”

→ “denselben Fehler nicht noch mal machen”  
→ Suchraum einschränken

“Grund” wird durch Analyse des Implikationsgraphen generiert, der Ursache/Wirkung von Wertzuweisungen dokumentiert

# 3. Trick: Nichtchronologisches Backtracking, Zufällige Restarts

Auf Grund der Konfliktanalyse nicht die letzte, sondern eine frühere Entscheidung rückgängig machen

Von Zeit zu Zeit einfach von vorn anfangen, und (randomisiert) an anderen Variablen verzweigen

→ Wissen über bisherige Suche in Form der gelernten Klauseln verfügbar

→ Möglichkeit, komplizierten Teilen des Suchraums zu entfliehen

# 4. Trick: Heuristiken für Entscheidungen

*Grasp*: Setze die Variable, die in den meisten offenen Klauseln vorkommt

*Chaff*: Setze Variable, die häufig in gelernten Klauseln vorkommt

## 4.2.2 Der Stålmarck- Algorithmus

patientierter Algorithmus, um Tautologie/Erfüllbarkeit  
*beliebiger* Formeln zu entscheiden

weicht vom üblichen Decide/Deduce/Backtrack ab

# Datenstruktur: Triplets

1. Beseitige Negationen: de Morgan +  $\neg x \Leftrightarrow (x \Rightarrow \perp)$

nehme für jede Teilformel  $\psi$  von  $\phi$  eine frische Variable  $x_\psi$  her,  
substituiere jede Benutzung von  $\psi$  durch  $x_\psi$ , und nehme  
 $x_\psi \Leftrightarrow \psi$  in die Formelmenge auf

Beispiel:

$$p \Rightarrow (q \Rightarrow p)$$

wird zu

$$\begin{aligned} x1 &\Leftrightarrow (q \Rightarrow p) \\ x2 &\Leftrightarrow (p \Rightarrow x1) \end{aligned}$$

Triplet:  $\langle \text{var} \rangle \Leftrightarrow (\langle \text{op} \rangle \langle \text{var} \rangle)$

# Start des Algorithmus

Tautologie:

Setze Top-Level-Variablen auf 0, suche konsistente Belegung

SAT:

Setze Top-Level-Variablen auf 1, suche konsistente Belegung

# Simple Rules

$$\frac{0 \Leftrightarrow (y \Rightarrow z)}{y/1 \quad z/0}$$

$$x/z$$

$$\frac{x \Leftrightarrow (y \Rightarrow 1)}{x/1}$$

$$x/z$$

$$\frac{x \Leftrightarrow (1 \Rightarrow z)}{x/z}$$

$$\frac{x \Leftrightarrow (0 \Rightarrow z)}{x/1}$$

$$x/\neg y$$

$$\frac{x \Leftrightarrow (y \Rightarrow 0)}{x/\neg y}$$

$$x/1$$

$$\frac{x \Leftrightarrow (x \Rightarrow z)}{x/1}$$

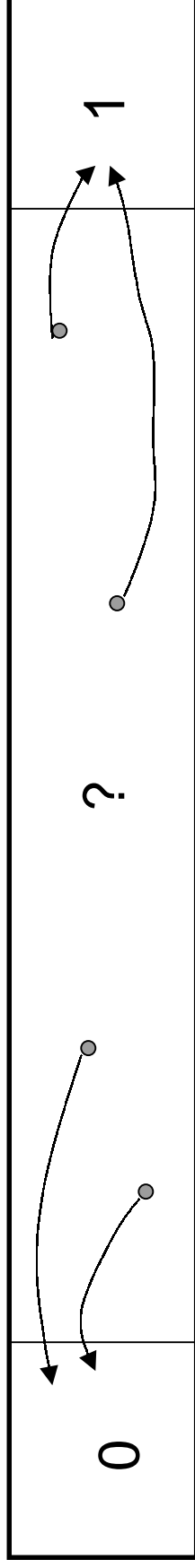
$$x/1$$

$$\frac{x \Leftrightarrow (y \Rightarrow y)}{x/1}$$

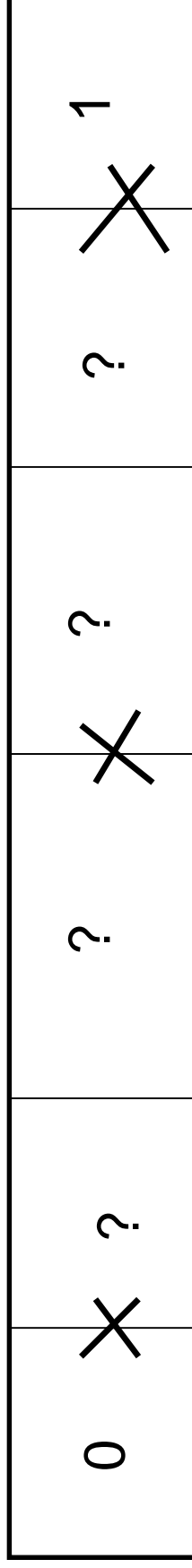
$$x/1$$

# Wirkung der simple rules

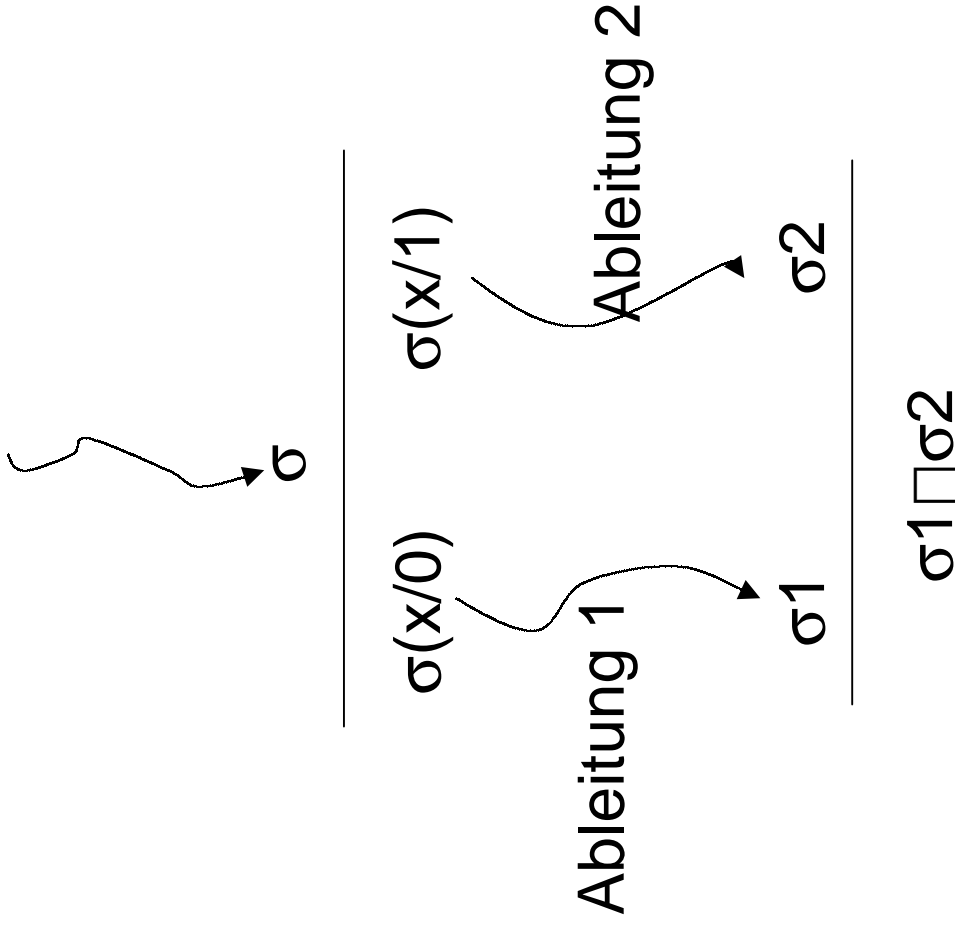
naiv:



Stålmarck:



# Dilemma rule

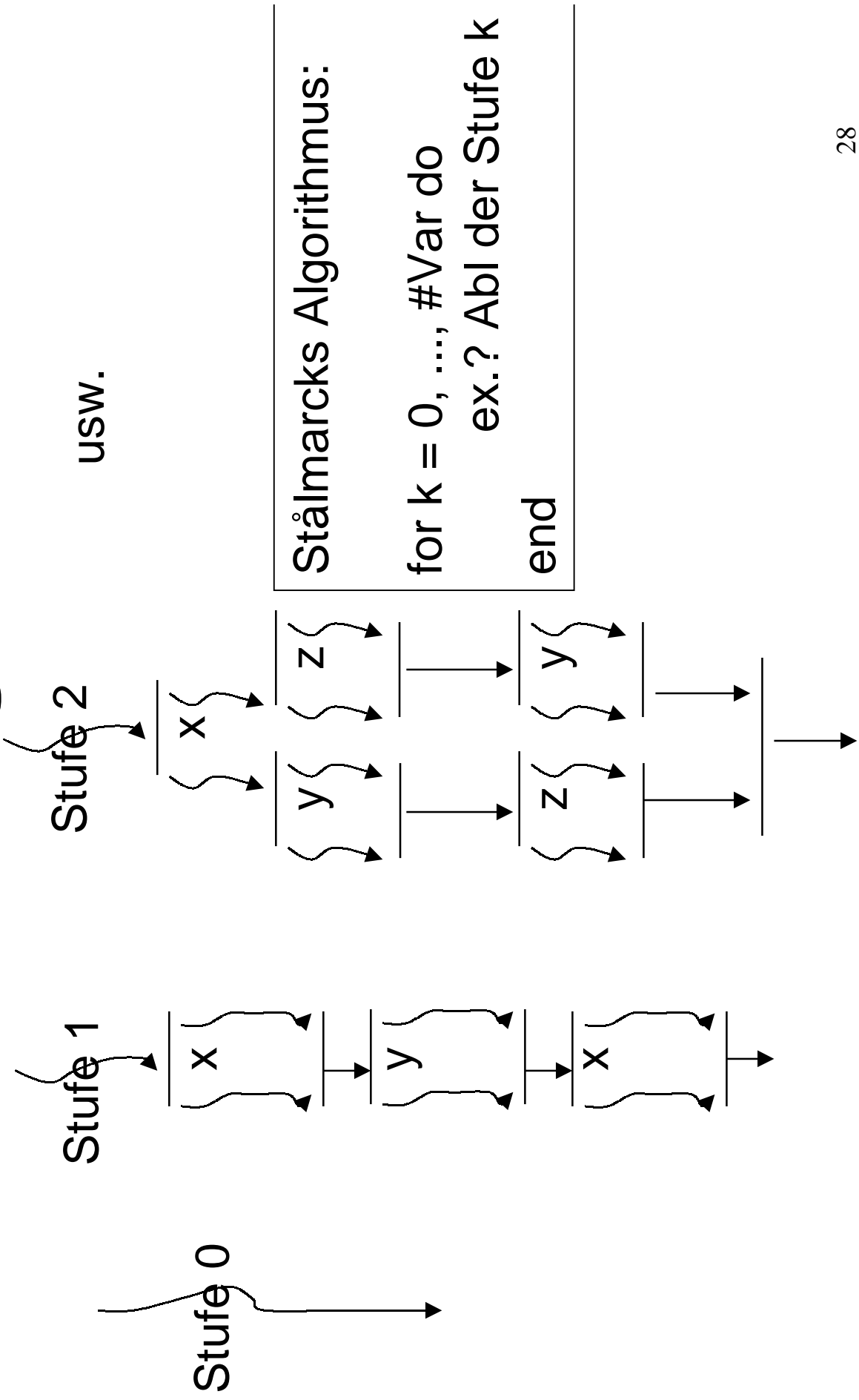


Zusammenführen der  
Zweige → Vermeide  
redundante Arbeit in  
verschiedenen Zweigen

= Eine der Subst., falls andere zu Konflikt führt;

= diejenigen Subst., die in beiden Zweigen gleich sind, sonst

# Ableitungsstufen



# Härte von Formeln

- $\phi$  ist k-hart, wenn Algorithmus frühestens in Runde k terminiert
- $\phi$  ist k-leicht, wenn Algorithmus spätestens in Runde k terminiert

Die meisten Formeln aus industriellem Kontext sind 1-leicht

2-hard = too hard

Laufzeit von Stålmarcks Algorithmus hängt viel mehr von der Härte des Ausdrucks ab als von seiner Länge

# Zusammenfassung SAT- Checker

arbeiten hervorragend auf Problemen “mit Struktur”

einige 10.000 – 100.000 Variablen

neben den vollständigen Checkern ex. noch viele unvollständige Verfahren

## 4.2.3 SAT-basiertes Model Checking

Idee: Übersetze Model Checking Problem in ein Erfüllbarkeitsproblem

Ausgangspunkt:

boolesche Kodierung des Zustandsraums, analog zu BDD-basiertem Model Checking

Zustandsüberführungsrelation als boolesche Formel

$T(x, x')$ , ebenfalls analog BDD-Methode

# Beispiel:3-bit-Zähler

$$\begin{aligned} &x_0' \Leftrightarrow \neg x_0 \\ &\wedge \\ &x_1' \Leftrightarrow (\neg x_1 \Leftrightarrow x_0) \\ &\wedge \\ &x_2' \Leftrightarrow (\neg x_2 \Leftrightarrow (x_1 \wedge x_0)) \end{aligned}$$

# Symbolische Pfade

Zustand mit Eigenschaft  $E$  ist von einem Zustand mit Eigenschaft  $I$  in genau  $k$  Schritten erreichbar:

$$I(x^{(0)}) \wedge T(x^{(0)}, x^{(1)}) \wedge \dots \wedge T(x^{(k-1)}, x^{(k)}) \wedge E(x^{(k)})$$

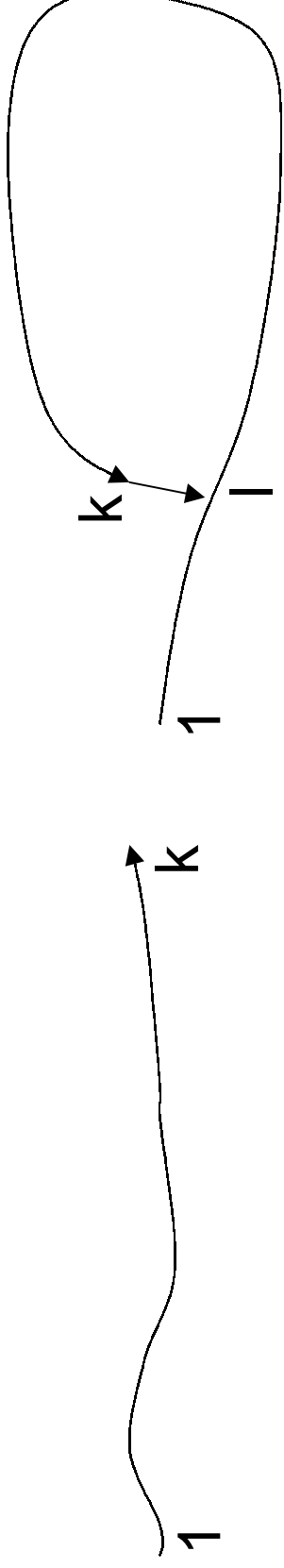
Suche bis zum kleinsten  $k$ , das erfüllt

$$\begin{aligned} & \forall x^{(0)} \dots \forall x^{(k)} \exists y^{(0)} \dots \exists y^{(j-1)} \\ & I(x^{(0)}) \wedge T(x^{(0)}, x^{(1)}) \wedge \dots \wedge T(x^{(k-1)}, x^{(k)}) \implies \\ & \bigvee_{j < k} [ I(y^{(0)}) \wedge T(y^{(0)}, y^{(1)}) \wedge \dots \wedge T(y^{(j-1)}, x^{(k)}) ] \end{aligned}$$

# Beschränkte Semantik von

## LTL

Idee: beschreiben Gegenbeispiel der Länge  $k$



Ziel: Wenn beschränkter Pfad  $\phi$  erfüllt, so auch jede unendliche Fortsetzung

Lassopfade: beschr. Semantik = originale Semantik

kreisfrei:

$$\pi \models_k F \phi \Leftrightarrow \exists i \leq k: \pi^{(i)} \models_{k-i} \phi$$

$$\pi \models_k G \phi \Leftrightarrow \text{false}$$

die anderen Operatoren so, wie man es sich denkt

# Übersetzung der Semantik

$$I(x^{(0)}) \wedge T(x^{(0)}, x^{(1)}) \wedge \dots \wedge T(x^{(k-1)}, x^{(k)}) \wedge \llbracket \phi \rrbracket_k^0$$

kreisfrei:

$$\llbracket p \rrbracket_k^i := p(x^{(i)})$$

$$\llbracket \phi \wedge \psi \rrbracket_k^i := \llbracket \phi \rrbracket_k^i \wedge \llbracket \psi \rrbracket_k^i \quad \llbracket \neg \phi \rrbracket_k^i := \neg \llbracket \phi \rrbracket_k^i$$

$$\llbracket G \phi \rrbracket_k^i := \text{false}$$

$$\llbracket F \phi \rrbracket_k^i := \bigvee_{j=i}^k \llbracket \phi \rrbracket_k^j$$

$$\llbracket X \phi \rrbracket_k^i := \text{falls } i < k, \text{ dann } \llbracket \phi \rrbracket_k^{i+1} \text{ sonst false}$$

$$\llbracket \phi \cup \psi \rrbracket_k^i := \bigvee_{j=i}^k ( \llbracket \psi \rrbracket_k^j \wedge \bigwedge_{n=j}^k \llbracket \phi \rrbracket_k^n )$$

# Übersetzung der Semantik

$$I(x^{(0)}) \wedge T(x^{(0)}, x^{(1)}) \wedge \dots \wedge T(x^{(k-1)}, x^{(k)}) \wedge T(x^{(k)}, x^{(l)}) \wedge \bigvee_{l=0}^k \llbracket \phi \rrbracket_k^o$$

Lasso:

$$\llbracket p \rrbracket_k^i := p(x^{(i)})$$

$$\llbracket \phi \wedge \psi \rrbracket_k^i := \llbracket \phi \rrbracket_k^i \wedge \llbracket \psi \rrbracket_k^i \quad \llbracket \neg \phi \rrbracket_k^i := \neg \llbracket \phi \rrbracket_k^i$$

$$\llbracket G \phi \rrbracket_k^i := \bigwedge_{n=\min(i,l)}^k \llbracket \phi \rrbracket_k^n$$

$$\llbracket F \phi \rrbracket_k^i := \bigvee_{j=\min(i,l)}^k \llbracket \phi \rrbracket_k^j$$

$$\llbracket X \phi \rrbracket_k^i := \llbracket \phi \rrbracket_k^{\text{succ}(i)} \quad \text{succ}(i) = i+1 \text{ falls } i < k, \text{ sonst } l$$

$$\llbracket \phi \cup \psi \rrbracket_k^i := \bigvee_{j=i}^k (\llbracket \psi \rrbracket_k^j \wedge \bigwedge_{n=j}^{j-1} \llbracket \phi \rrbracket_k^n) \vee \bigvee_{j=i-1}^k (\llbracket \psi \rrbracket_k^j \wedge \bigwedge_{n=j}^k \llbracket \phi \rrbracket_k^n \wedge \bigwedge_{n=j}^{j-1} \llbracket \phi \rrbracket_k^n)$$

# Zusammenfassung SAT

Es gibt inzwischen auch zustandsbasierte SAT-Model Checker

Man kann auf weitere Zuwächse im SAT-Solving vertrauen

Formeln für Bestimmung des max.  $k$  gelten als schwer

- meist wird auf Bestimmung oberer Schranken für  $k$  verzichtet
- unvollständige Methode

# Zusammenfassung

## symbolisches Model Checking

Ideen für andere geeignete Datenstrukturen?

Symbolische Model Checker arbeiten gelegentlich mit Überapproximationen, d.h. einer Obermenge der erreichbaren Zustände, die Datenstruktur verkleinert

Industrielle Model Checker sind meist symbolisch

BDD und SAT-basierte Methode mischen sich –  
“Boolean Expression Diagrams”