Vorlesung
– Automatisches Beweisen –

Kap. 4.2.4: Binäre Entscheidungsdiagramme (BDD)

Prof. Dr. Wolfgang Küchlin

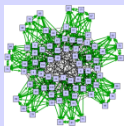
Dipl.-Inform., Dr. sc. techn. (ETH)

Arbeitsbereich Symbolisches Rechnen
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften

Universität Tübingen

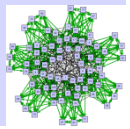
Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR 

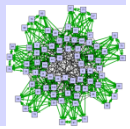
Binäre Entscheidungsdiagramme (BDDs)

- 1986 von R. Bryant vorgeschlagen
- Graphen-basierter Formalismus zur Darstellung boolescher Funktionen
- Aussagenlogische Formeln repräsentiert als gerichteter, azyklischer Graph (DAG)
- Liefern als ROBDDs eine kanonische Repräsentierung von Formeln (eindeutig in jeder Äquivalenzklasse).



ROBDDs: Canonicity

- ROBDD b zu gegebener booleschen Funktion f ist eindeutig, d.h.:
  - b ist eindeutiger Repräsentant aller zu f äquivalenten Formeln (kanonische Form).
  - Spezialfall: Unerfüllbare (allgemeingültige) Formeln werden durch 0-Knoten (1-Knoten) repräsentiert.
- Konsequenzen:
  - F ≡ G gdw. ROBDD(F) = ROBDD(G)
  - F ≡ ⊤ gdw. ROBDD(F) = ROBDD(⊤) = 1
  - F ≡ ⊥ gdw. ROBDD(F) = ROBDD(⊥) = 0
  - F erfüllbar gdw. ROBDD(F) ≠ 0
  - SAT-Problem auf ROBDDs in O(1) lösbar

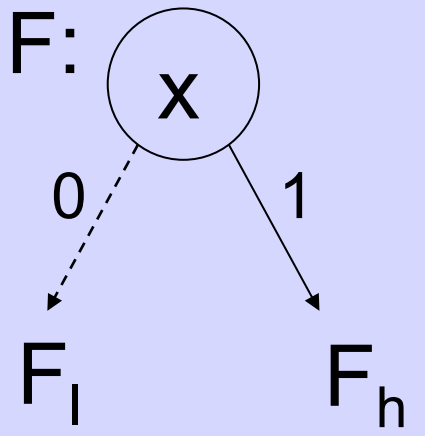


Binäre Entscheidungsdiagramme (BDDs)

- Terminalknoten 0 und 1 stellen konstante Funktion f bzw. t dar.
- Innere Knoten interpretiert als:

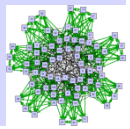
  F = if x then F_h else F_l

  $(x \Rightarrow F_h) \wedge (\neg x \Rightarrow F_l)$



BDDs: Bestimmung von F_h und F_l

- Die Restriktion $F|_{x=b}$ ($b \in \{0,1\}$) ist rekursiv definiert durch:
  - $f|_{x=b} = f$
  - $$y|_{x=b} = \begin{cases} t & \text{falls } x = y \text{ und } b = 1 \\ f & \text{falls } x = y \text{ und } b = 0 \\ y & \text{sonst} \end{cases}$$
  - $(\neg G)|_{x=b} = \neg(G|_{x=b})$
  - $(G \vee H)|_{x=b} = G|_{x=b} \vee H|_{x=b}$
  - $(G \wedge H)|_{x=b} = G|_{x=b} \wedge H|_{x=b}$

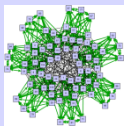


BDDs: Bestimmung von F_h und F_l (2)

- Shannon-Expansion:

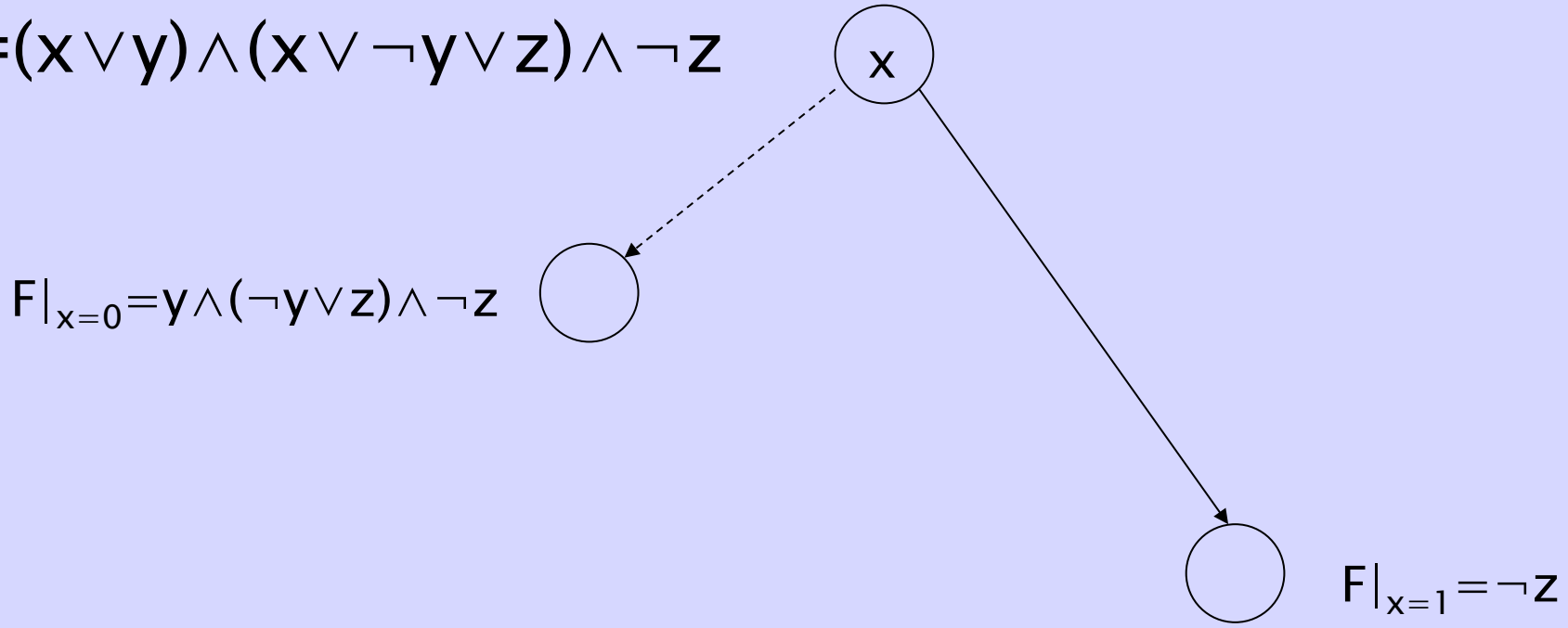
$$F \equiv (x \wedge F) \vee (\neg x \wedge F)$$
$$\equiv (x \wedge F|_{x=1}) \vee (\neg x \wedge F|_{x=0})$$
$$\equiv ((x \wedge F|_{x=1}) \vee \neg x) \wedge ((x \wedge F|_{x=1}) \vee F|_{x=0})$$
$$\equiv ((x \vee \neg x) \wedge (F|_{x=1} \vee \neg x)) \wedge ((x \vee F|_{x=0}) \wedge (F|_{x=1} \vee F|_{x=0}))$$
$$\equiv (\neg x \vee F|_{x=1}) \wedge (x \vee F|_{x=0})$$
$$\equiv (x \Rightarrow F|_{x=1}) \wedge (\neg x \Rightarrow F|_{x=0})$$

→ $F_l \mathrel{\hat{=}} F|_{x=0}$ und $F_h \mathrel{\hat{=}} F|_{x=1}$



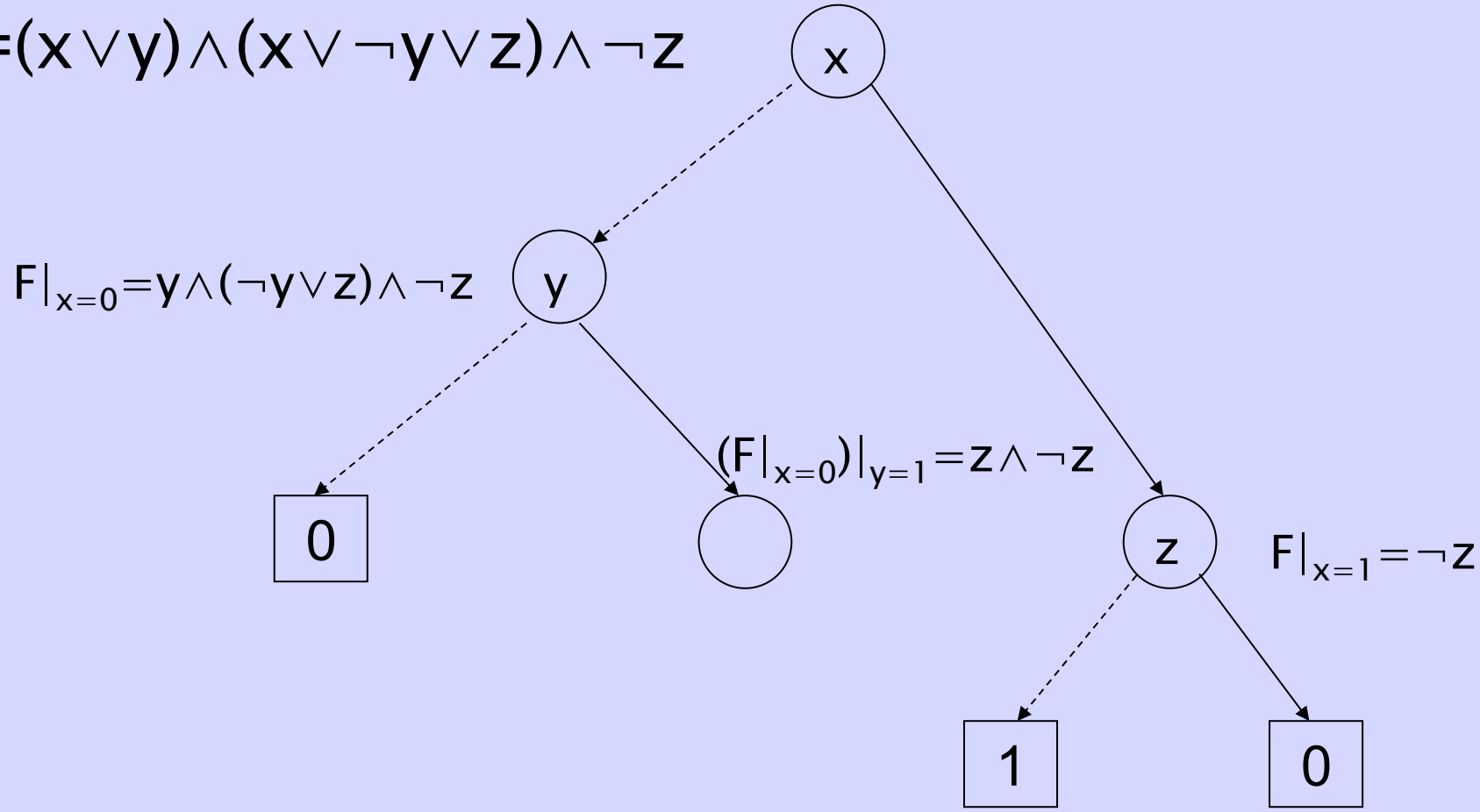
BDDs: Beispiel zur Generierung

- $F=(x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$



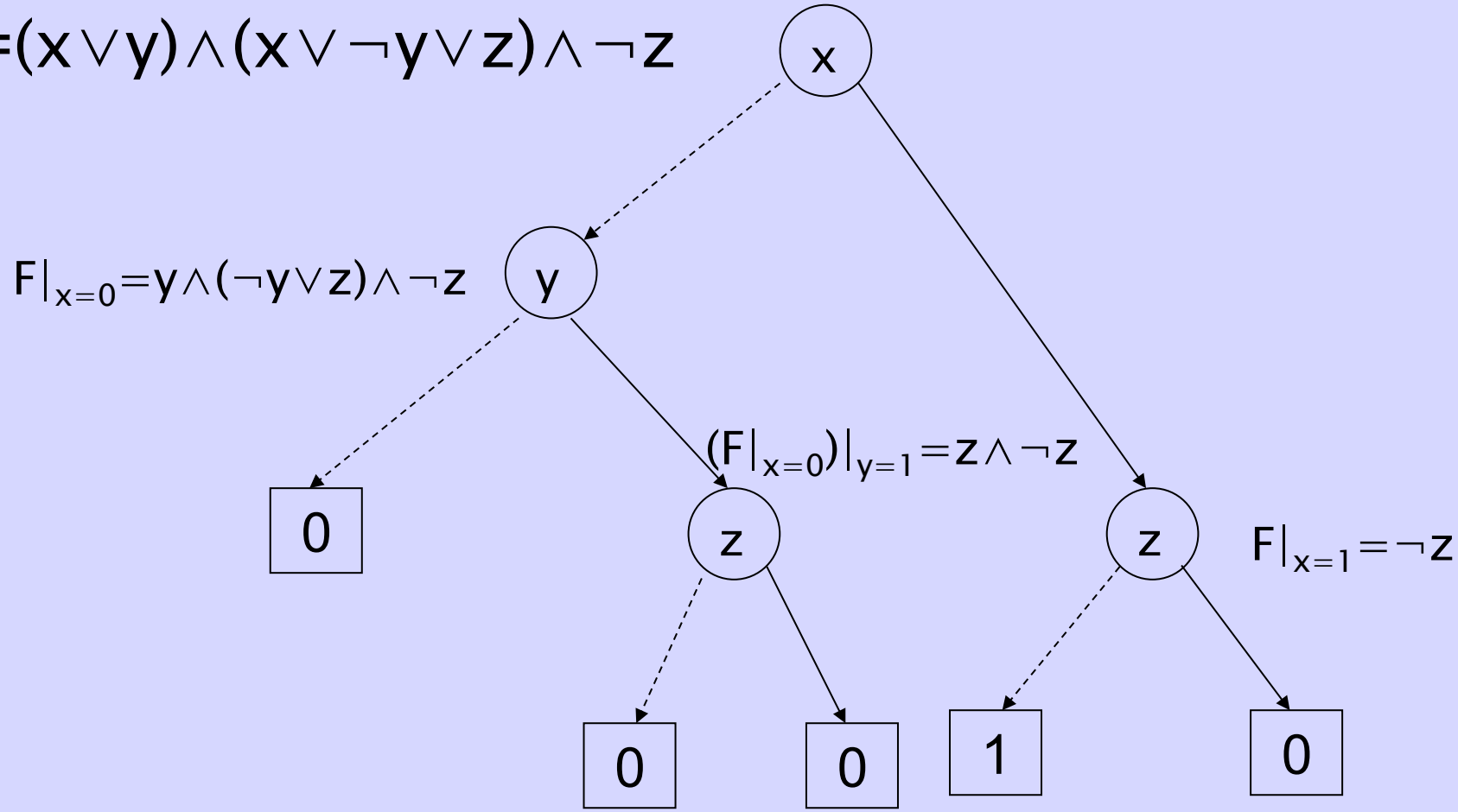
BDDs: Beispiel zur Generierung

- $F=(x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$



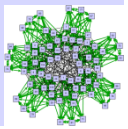
BDDs: Beispiel zur Generierung

- $F=(x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$



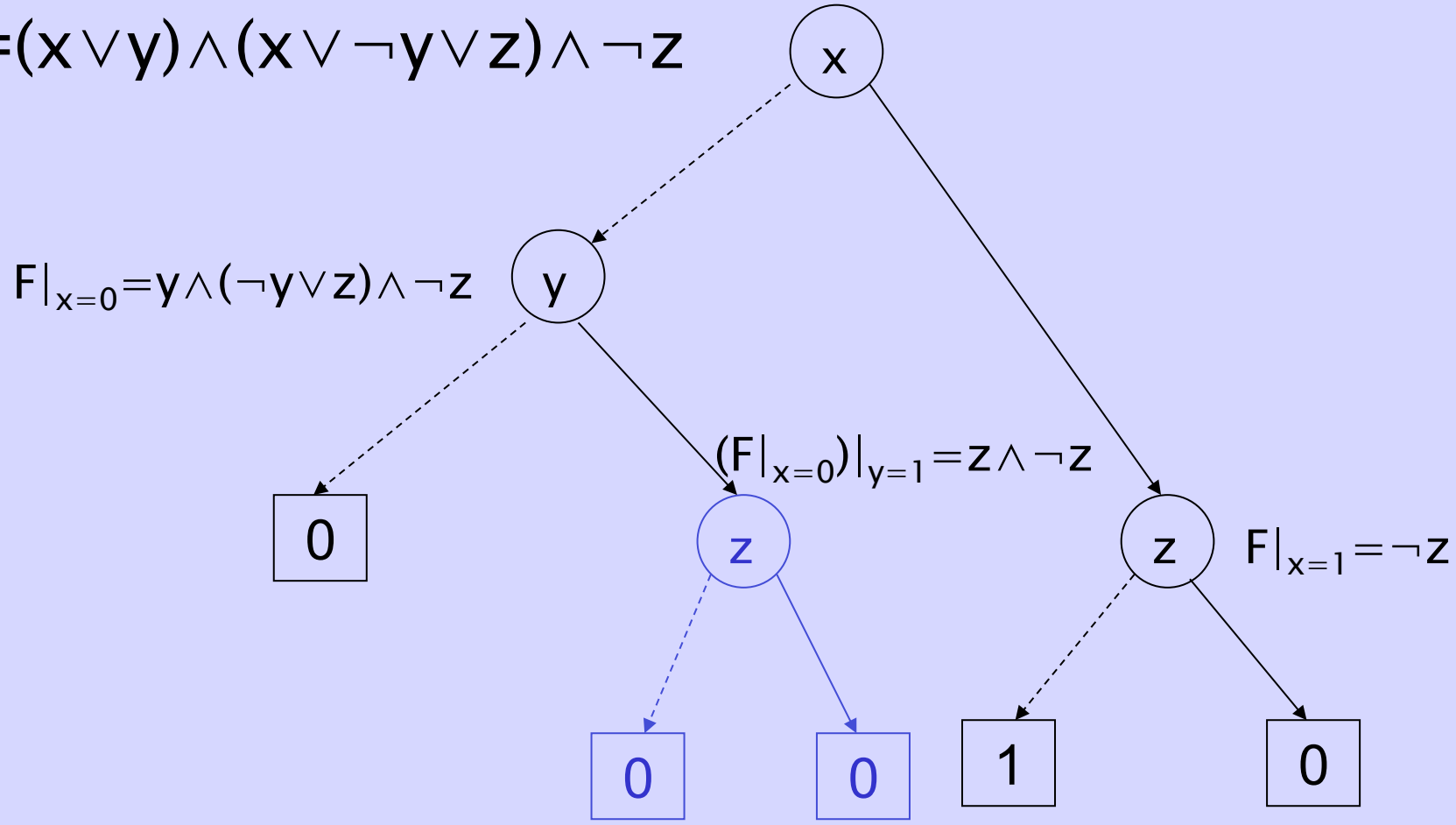
Reduced Ordered BDDs (ROBDDs)

- strikte, totale Ordnung $\prec$ auf Variablen
  - Variable des Elternknotens muss kleiner sein als Variablen beider Kindknoten
  - Auf jedem Pfad kommt jede Variable höchstens einmal vor
- Reduktionen:
  - Knoten eindeutig dargestellt
  - Knoten k mit $k_h=k_l$ werden gelöscht und Kanten, die auf k zeigen werden auf k_h weitergezogen.



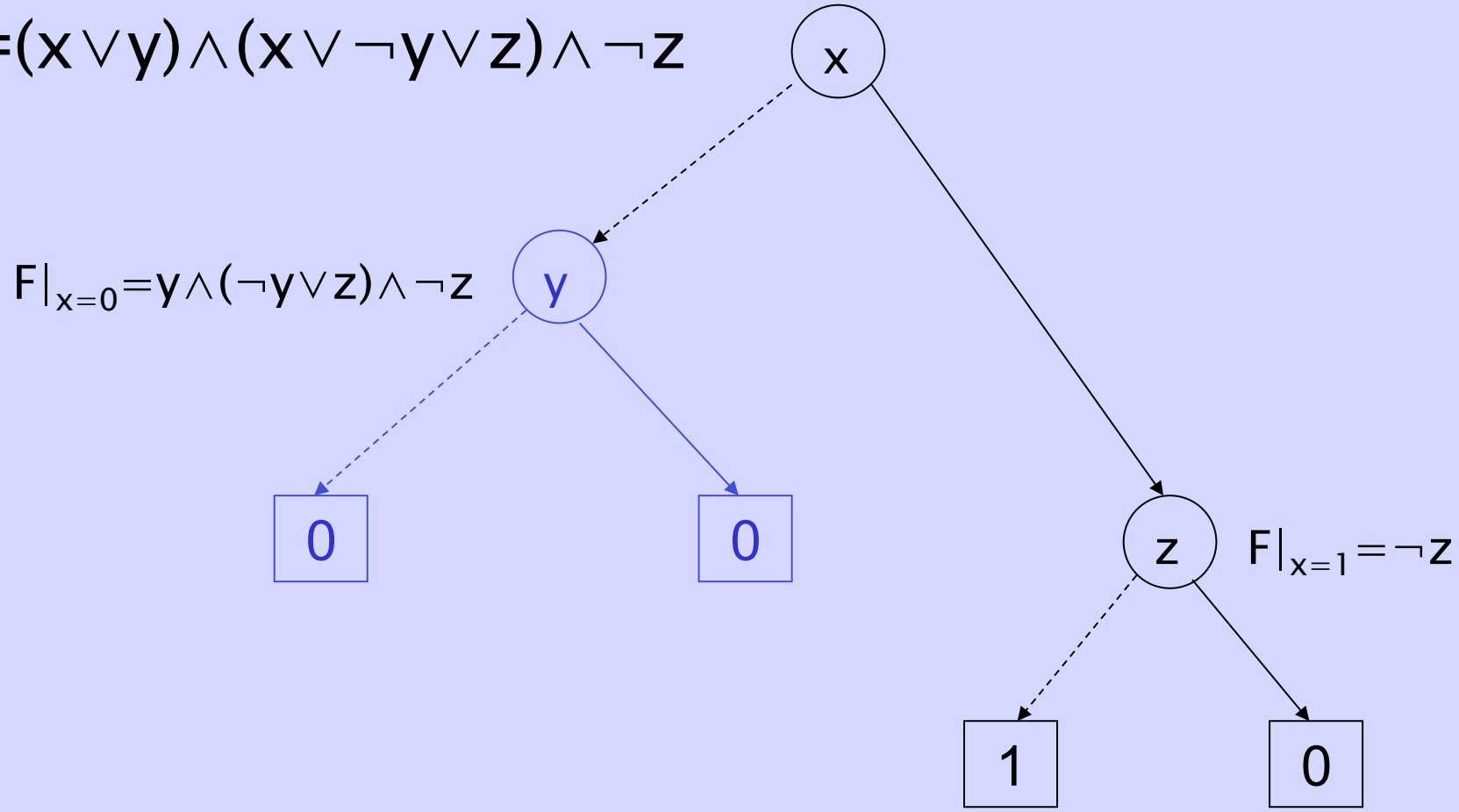
ROBDDs: Beispiel zur Reduktion eines BDDs

- $F=(x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$



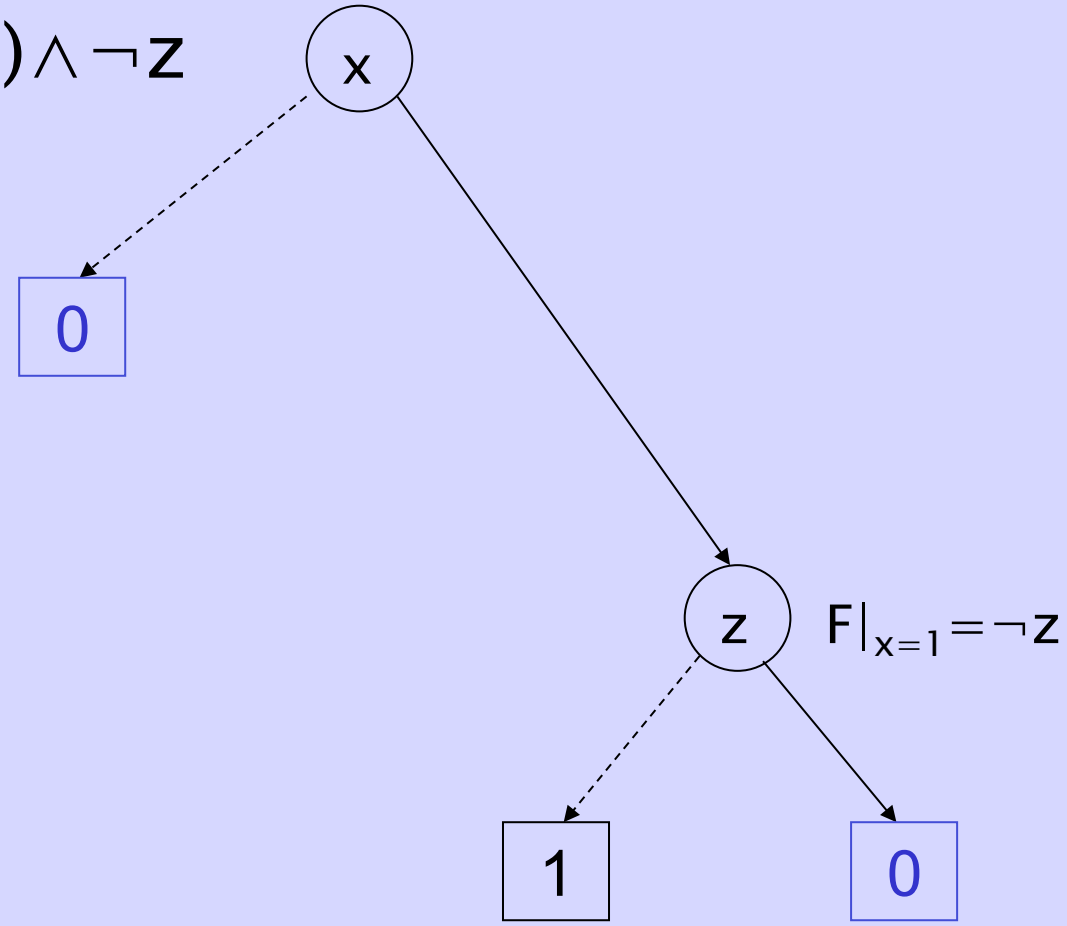
ROBDDs: Beispiel zur Reduktion eines BDDs

- $F=(x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$



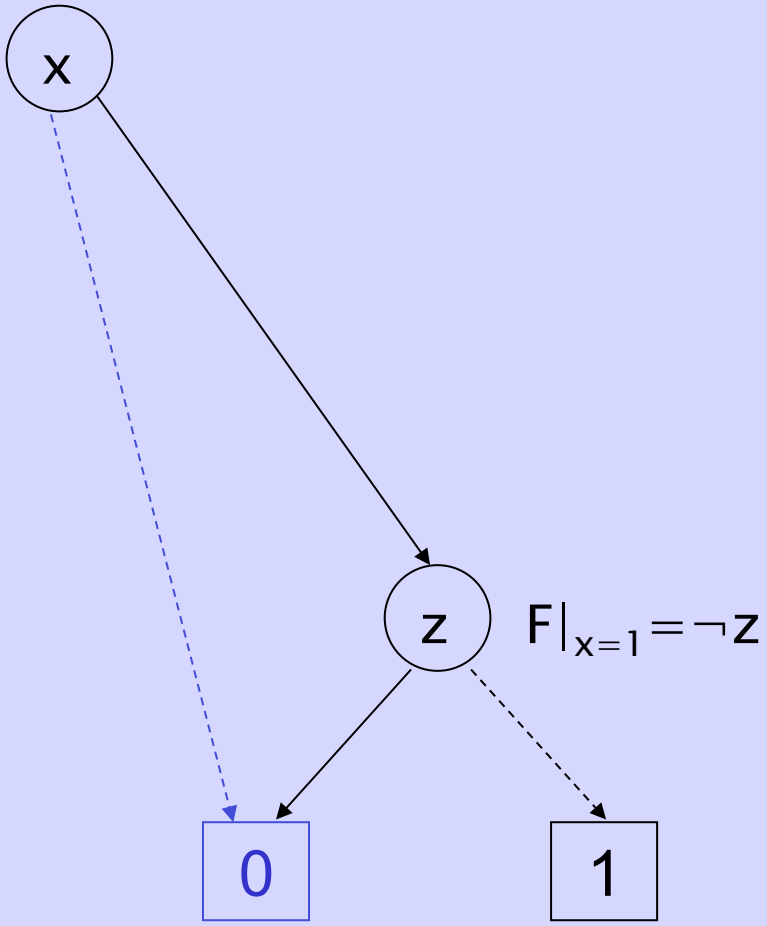
ROBDDs: Beispiel zur Reduktion eines BDDs

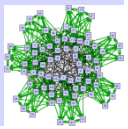
- $F=(x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$



ROBDDs: Beispiel zur Reduktion eines BDDs

- $F=(x \vee y) \wedge (x \vee \neg y \vee z) \wedge \neg z$





Vom BDD zur Booleschen Formel

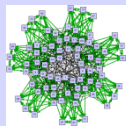
- Wie können wir aus einem BDD die zugehörige Boolesche Formel ablesen?
- Es gibt vier (äquivalente) Interpretationen für einen Knoten im BDD (vgl. Shannon-Expansion):

  $F \equiv (x \wedge F|_{x=1}) \vee (\neg x \wedge F|_{x=0})$ // → Gewinnung der DNF

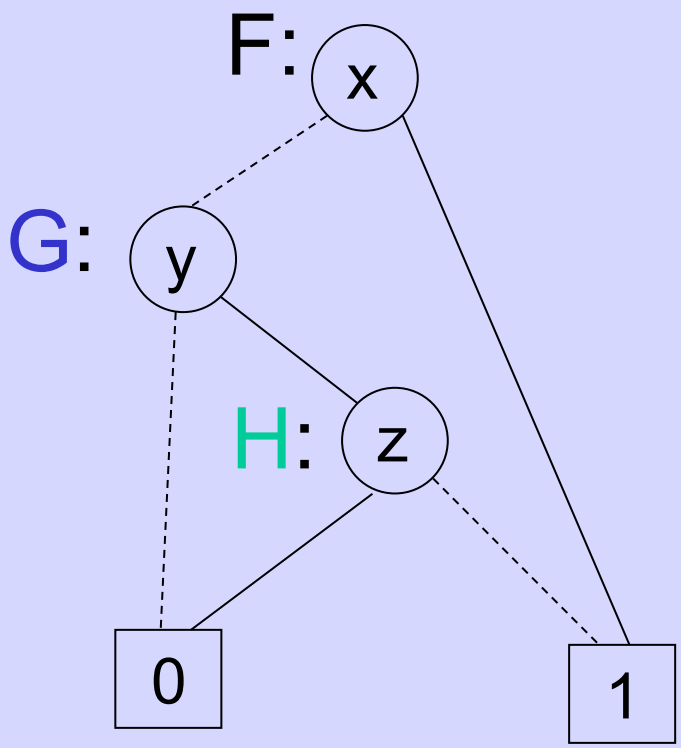
  $\equiv (\neg x \vee F|_{x=1}) \wedge (x \vee F|_{x=0})$ // → Gewinnung der CNF

  $\equiv (x \Rightarrow F|_{x=1}) \wedge (\neg x \Rightarrow F|_{x=0})$

  $\equiv$ if x then $F|_{x=1}$ else $F|_{x=0}$ fi

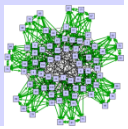


Formelgewinnung aus ROBDD

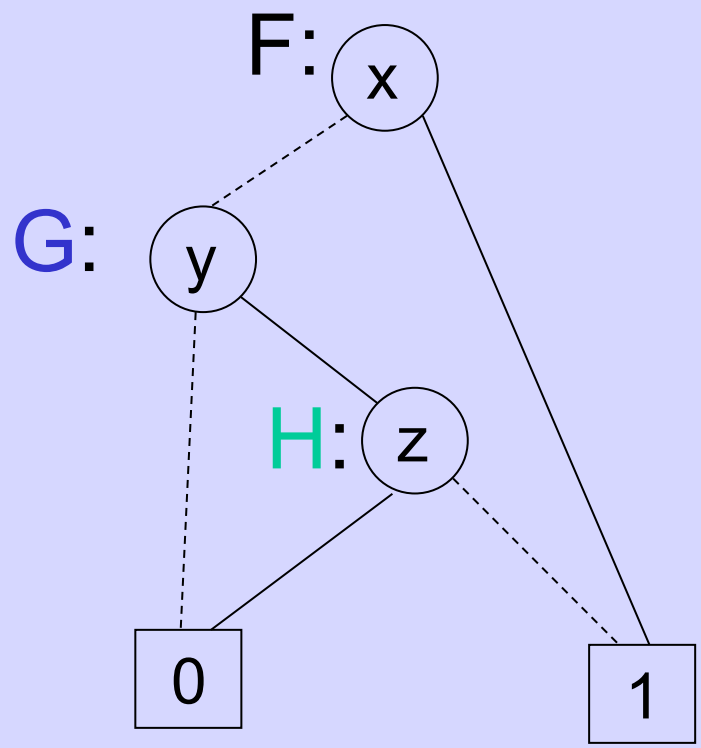


BDD-Darstellung
von $x \vee (y \wedge \neg z)$

F = if x then $\top$ else G
= if x then $\top$ else (if y then H else $\bot$)
= if x then $\top$ else
(if y then (if z then $\bot$ else $\top$) else $\bot$)
= if x then $\top$ else (if y then $\neg z$ else $\bot$)
= if x then $\top$ else $(y \Rightarrow \neg z) \wedge (\neg y \Rightarrow \bot)$
= if x then $\top$ else $(y \Rightarrow \neg z) \wedge y$
= if x then $\top$ else $\neg z \wedge y$
= $(x \Rightarrow \top) \wedge (\neg x \Rightarrow \neg z \wedge y)$
= $x \vee (\neg z \wedge y)$



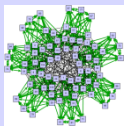
DNF Gewinnung aus ROBDD



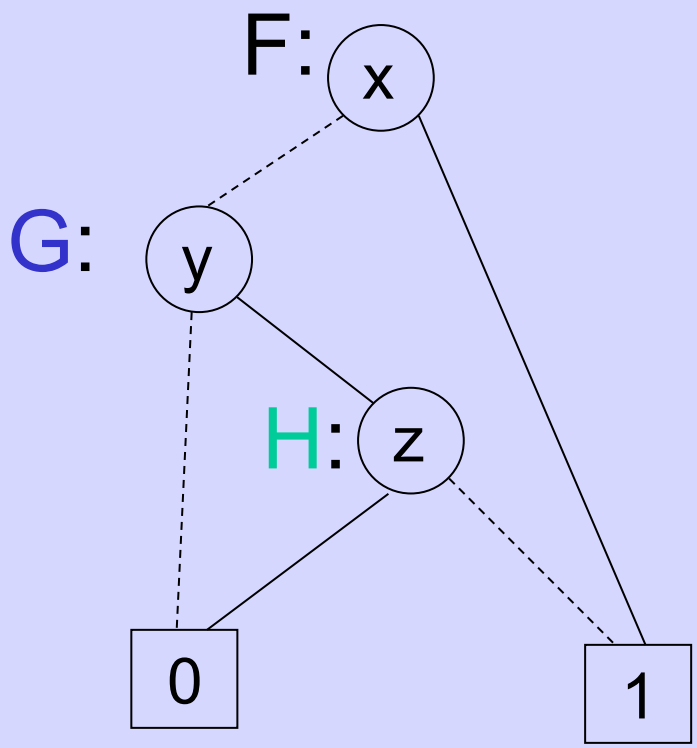
BDD-Darstellung
von $x \vee (y \wedge \neg z)$

- Wir benutzen $(x \wedge F|_{x=1}) \vee (\neg x \wedge F|_{x=0})$
- $DNF(F) = (x \wedge DNF(F_{hi})) \vee (\neg x \wedge DNF(F_{lo}))$

$F = (x \wedge \top) \vee (\neg x \wedge DNF(G))$
$= x \vee (\neg x \wedge ((y \wedge DNF(H)) \vee (\neg y \wedge \bot)))$
$= x \vee (\neg x \wedge y \wedge DNF(H) \vee \neg x \wedge \neg y \wedge \bot)$
$= x \vee (\neg x \wedge y \wedge DNF(H))$
$= x \vee (\neg x \wedge y \wedge ((z \wedge \bot) \vee (\neg z \wedge \top)))$
$= x \vee (\neg x \wedge y \wedge (z \wedge \bot) \vee \neg x \wedge y \wedge (\neg z \wedge \top))$
$= x \vee (\neg x \wedge y \wedge \neg z)$
$= (x \vee \neg x) \wedge (x \vee (y \wedge \neg z))$
$= x \vee (\neg z \wedge y)$



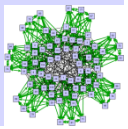
DNF Gewinnung aus ROBDD (2)



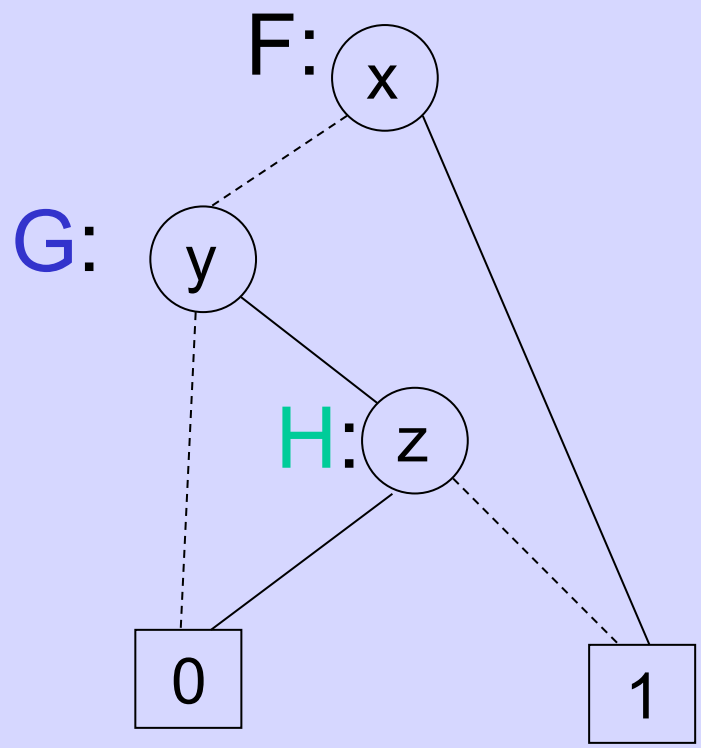
BDD-Darstellung
von x∨(y∧¬z)

Tiefensuche, left most – inner most,
Backtracking:

F = (¬x ∧ y ∧ ¬z) ∨ x
= (x ∨ ¬x) ∧ (x ∨ (y ∧ ¬z))
= x ∨ (y ∧¬z)



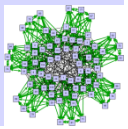
CNF Gewinnung aus ROBDD



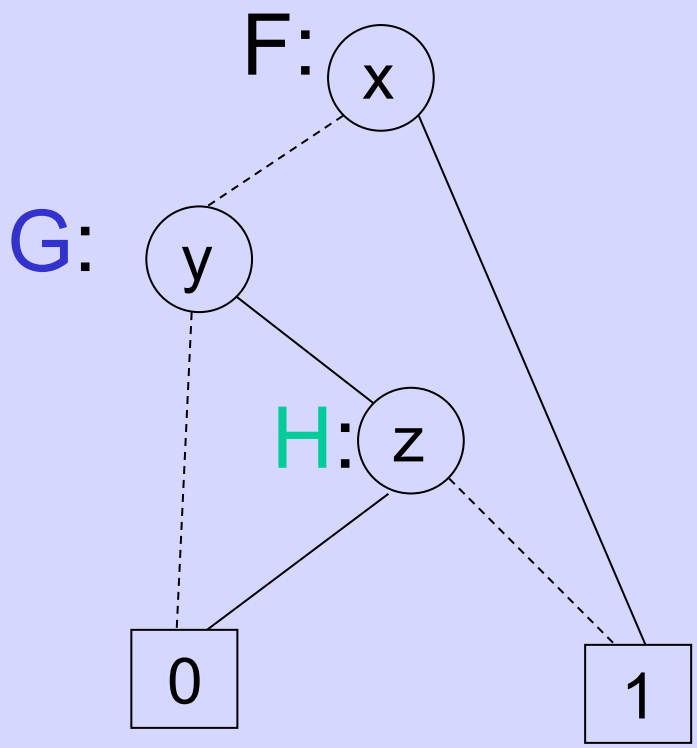
BDD-Darstellung
von $x \vee (y \wedge \neg z)$

- Wir benutzen $(\neg x \vee F|_{x=1}) \wedge (x \vee F|_{x=0})$
- $CNF(F) = (\neg x \vee CNF(F_{hi})) \wedge (x \vee CNF(F_{lo}))$

$F = (\neg x \vee \top) \wedge (x \vee CNF(G))$
$= x \vee ((\neg y \vee CNF(H)) \wedge (y \vee \bot))$
$= (x \vee \neg y \vee CNF(H)) \wedge (x \vee (y \vee \bot))$
$= (x \vee \neg y \vee CNF(H)) \wedge (x \vee y)$
$= (x \vee \neg y \vee ((\neg z \vee \bot) \wedge (z \vee \top))) \wedge (x \vee y)$
$= (x \vee \neg y \vee \neg z \vee \bot) \wedge (x \vee \neg y \vee z \vee \top) \wedge (x \vee y)$
$= (x \vee \neg y \vee \neg z) \wedge (\top) \wedge (x \vee y)$
$= (x \vee \neg y \vee \neg z) \wedge (x \vee y)$
$= x \vee ((\neg y \vee \neg z) \wedge (y))$
$= x \vee ((\neg y \wedge y) \vee (\neg z \wedge y))$
$= x \vee (\neg z \wedge y)$



CNF Gewinnung aus ROBDD (2)



BDD-Darstellung
von $x \vee (y \wedge \neg z)$

Idee: CNF ist die Konjunktion aus Allem, was nicht passieren darf.

Also: Simultaner Ausschluss der Wege zur 0 zu kommen.

$F = \neg(\neg x \wedge \neg y) \wedge \neg(\neg x \wedge y \wedge z)$

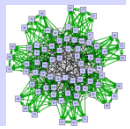
$\equiv (x \vee y) \wedge (x \vee \neg y \vee \neg z)$

$\equiv x \vee (y \wedge (\neg y \vee \neg z))$

$\equiv x \vee (y \wedge \neg z)$

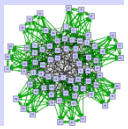
ROBDDs: Größe

- Die Größe des ROBDDs kann stark von der gewählten Variablenordnung abhängen:
  - Bryant: Das ROBDD für die Formel $(p_1 \wedge p_2) \vee \ldots \vee (p_{2n-1} \wedge p_{2n})$ hat 2n+2 Knoten unter der Ordnung $p_1, p_2, \ldots, p_{2n}$ und 2^{n+1} Knoten unter der Ordnung $p_1, p_{n+1}, p_2, p_{n+2}, \ldots, p_n, p_{2n}$.
  - Bryant: Es gibt F mit n Variablen, so dass ROBDD zu F für jede Ordnung mindestens 2^{cn} Knoten hat (c>0).



Generierung von ROBDDs

- „Top-Down“-Ansatz
  - Zerlegung der Eingabeformel F unter gleichzeitiger Generierung des BDDs, d.h
    - Selektion der größten in F vorkommenden Variablen
    - Anlegen eines Knotens und Berechnung von F_l und F_h
    - Rekursive BDD-Transformation von F_l und F_h
    - Einbau der Kind-BDDs in den Knoten und Reduktion
- „Bottom-Up“-Ansatz
  - BDDs für atomare Formeln (Variablen, $\bot$, $\top$)
  - Operationen zur Generierung komplexer BDDs aus einfacheren (z.B. BDD-Disjunktion, -Konjunktion, -Negation)



Effiziente Konstruktion von ROBDDs

- Aufbau eines ODT (ordered decision tree)

  Sei a ein boolescher Ausdruck in n Variablen.

  Build(a,n) :≡ Build´(a,1,n)

  Build´(a,i,n) :≡

  if (i>n) then // all variables are substituted, a evaluates to true or false

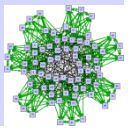
  if a then return 1 else return 0 fi

  else // build BDDs for low(x_i) and hi(x_i) and build node x_i

  MkNode(i, Build´(a[x_i/0],i+1,n), Build´(a[x_i/1],i+1,n))

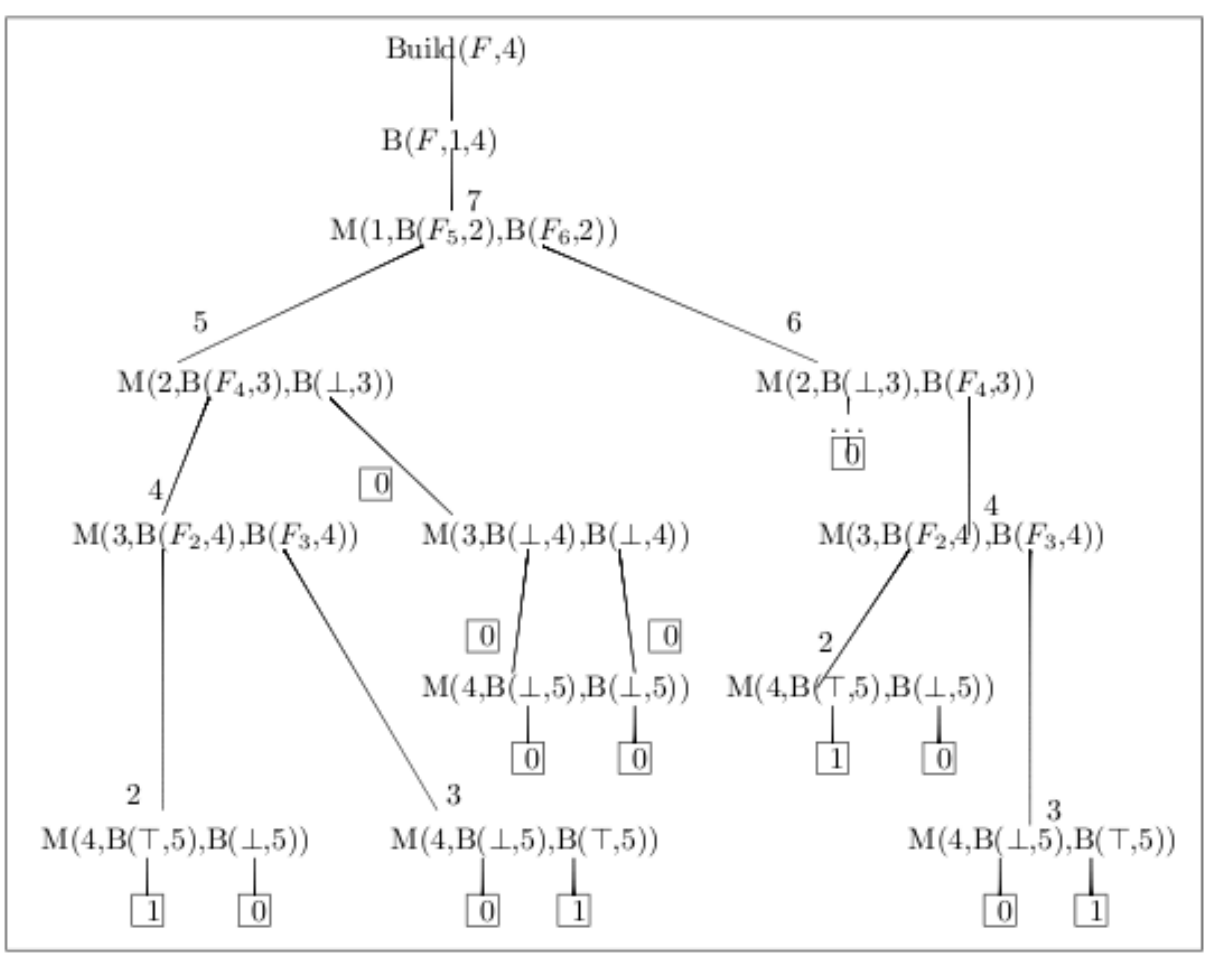
  fi

- Bei naiver Implementierung von MkNode wird ein Baum aufgebaut mit 2^n Knoten
  - Gleiche Knoten werden mehrfach erzeugt



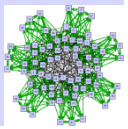
Beispiel zum Aufbau eines ROBDDs

- F = $(\neg x \wedge \neg y \wedge (z \leftrightarrow u)) \vee (x \wedge y \wedge (z \leftrightarrow u))$



- F_5= F[x=0] = $(\neg y \wedge (z \leftrightarrow u))$
- F_6 = F[x=1] = $(y \wedge (z \leftrightarrow u))$
- F_4 = F_5[y=0] = $(z \leftrightarrow u)$
- F_5[y=1] = $\bot$
- F_2 = F_4[z=0] = $(\bot \leftrightarrow u)$
- F_2[u=0] = $\top$
- F_2[u=1] = $\bot$
- F_3 = F_4[z=1] = $(\top \leftrightarrow u)$
- F_3[u=0] = $\bot$
- F_3[u=1] = $\top$
- F_6[y=0] = $\bot$
- F_6[y=1] = $(z \leftrightarrow u)$ = F_4

B steht für Build´ und M für MkNode.

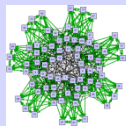


Konstruktion von ROBDDs (2)

- Effizienter Aufbau von ROBDDs
  - Kein unnötiger rek. Aufruf Build´(a,i,n) falls a = $\top$ oder a = $\bot$
  - Jeder Knoten darf nur einmal konstruiert werden
  - Tabellenrepräsentation, um existierende Knoten schnell zu finden.

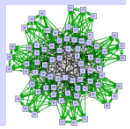
Node #	Var #	l	h
0	n+1		
1	n+1		
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6

- Operationen auf Tabelle T:
  - init(T): füge 0 und 1 in leeres T ein
  - u←add(i,l,h): neuer Knoten
  - var(u), low(u), high(u): accessors



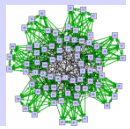
Konstruktion von ROBDDs (3)

- Zum Auffinden existierender Knoten inverse Tabelle H:
  - init(H): leeres H anlegen
  - b←member(i,l,h): ist (i,l,h) in H vorhanden?
  - u←lookup(i,l,h): finde H(i,l,h)
  - insert(i,l,h,u): repräsentiere Abb. (i,l,h) ≡ u durch Eintrag in H
- H muss effizient als Hashmap realisiert werden
  - sonst könnte man auch in T suchen …



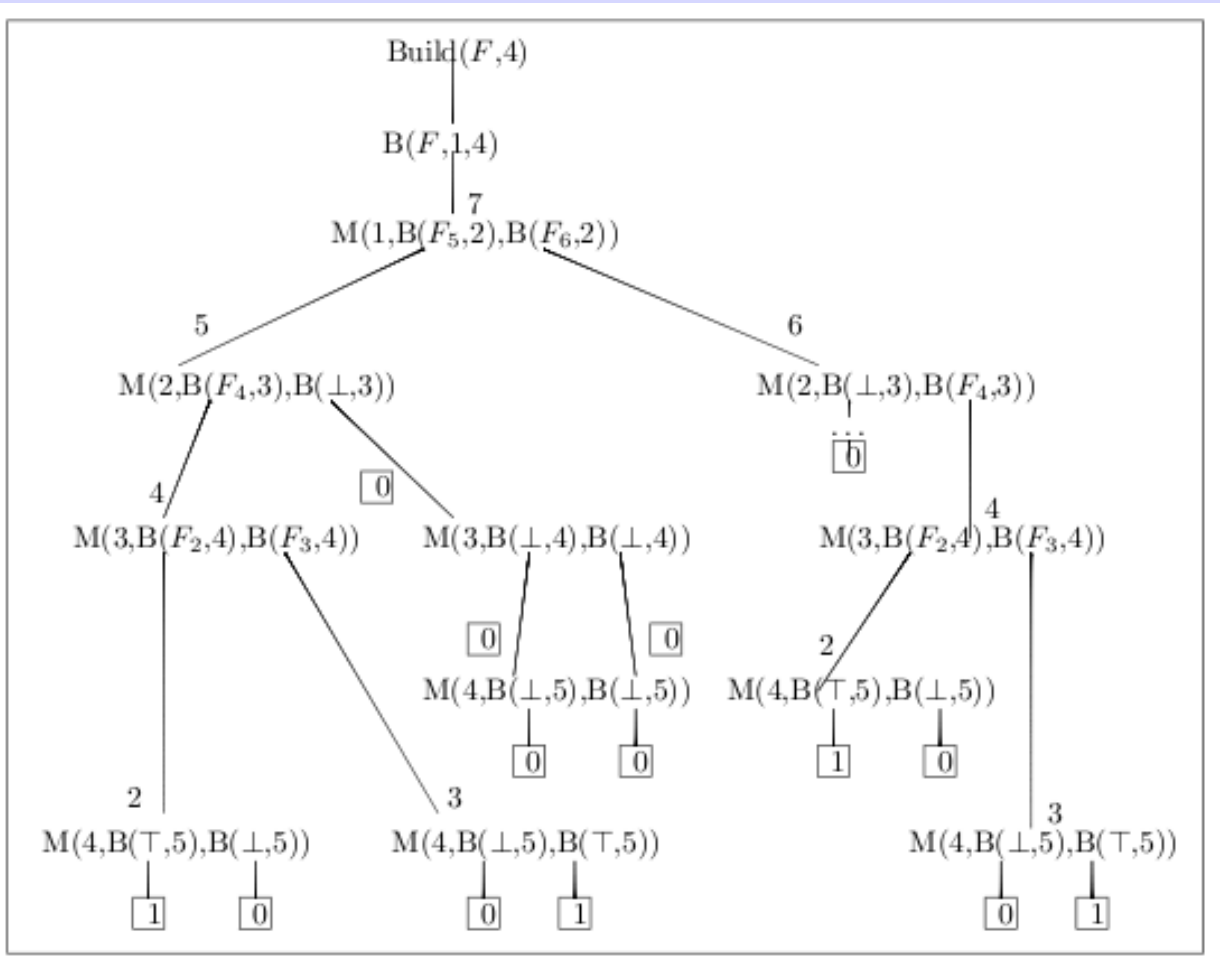
Intelligentes MakeNode für ROBDDs

```
MkNode[T,H](i,l,h) : ≡
        if (l=h) then return h
        else if member[H](i,l,h) then return lookup[H](i,l,h)
                else u = add[T](i,l,h);
                        insert[H](i,l,h,u);
                        return u;
                fi
        fi
```



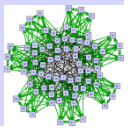
Beispiel zum Aufbau eines ROBDDs

- $F = (\neg x \wedge \neg y \wedge (z \leftrightarrow u)) \vee (x \wedge y \wedge (z \leftrightarrow u))$



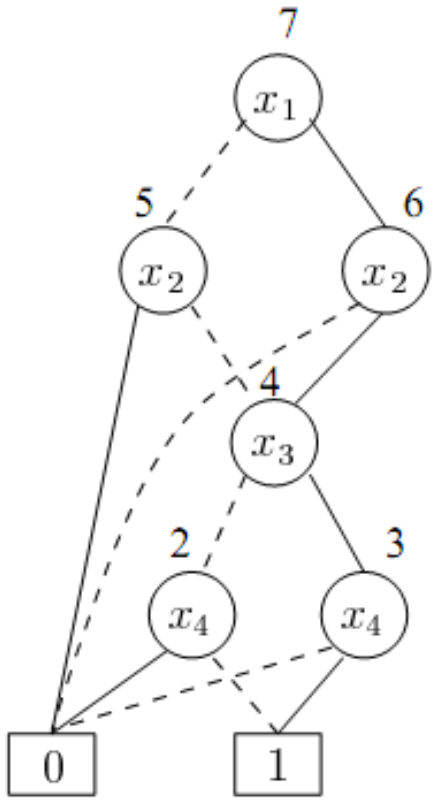
Node #	Var #	l	h
0	n+1		
1	n+1		
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6

steht für Build´ und M für MkNode.



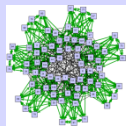
Beispiel zum Aufbau eines ROBDDs

- F = (¬x∧¬y∧(z↔u))∨(x∧y∧(z↔u))



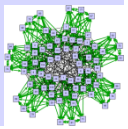
$T : u \rightarrow (i, l, h)$

u	var	low	$high$
0	5		
1	5		
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6



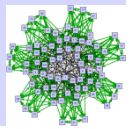
Boolesche Operationen auf ROBDDs

- Apply(op, u1, u2)
  - Wende boolesche Operation op auf BDD u1 und BDD u2 an
  - Apply(op, BDD(t1), BDD(t2)) = BDD(t1 op t2)
- Grundlagen:
  - (if x then t1 else t2 fi) op (if x then t ‘1 else t ‘2 fi)
    ≡ if x then (t1 op t ‘1) else (t2 op t ‘2) fi
  - (if x then t1 else t2 fi) op t
    ≡ if x then t1 op t else t2 op t fi
- Praxis:
  - Verwende (Hash-)Tabelle G zur Speicherung von Zwischenergebnissen (dyn. Programmieren)
  - Nicht alle Variablen sind in beiden BDDs gleichzeitig vorhanden

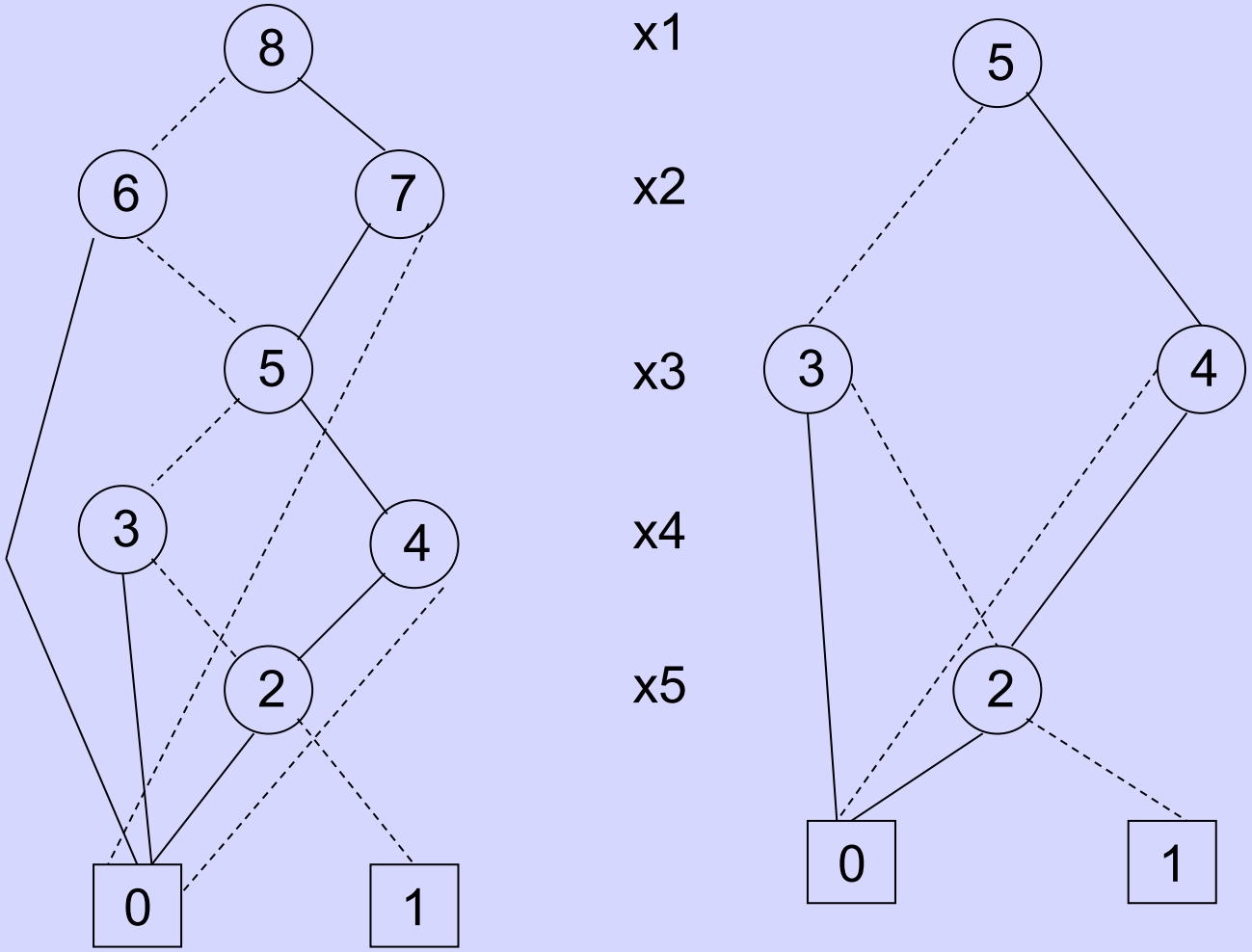


Apply für ROBDDs

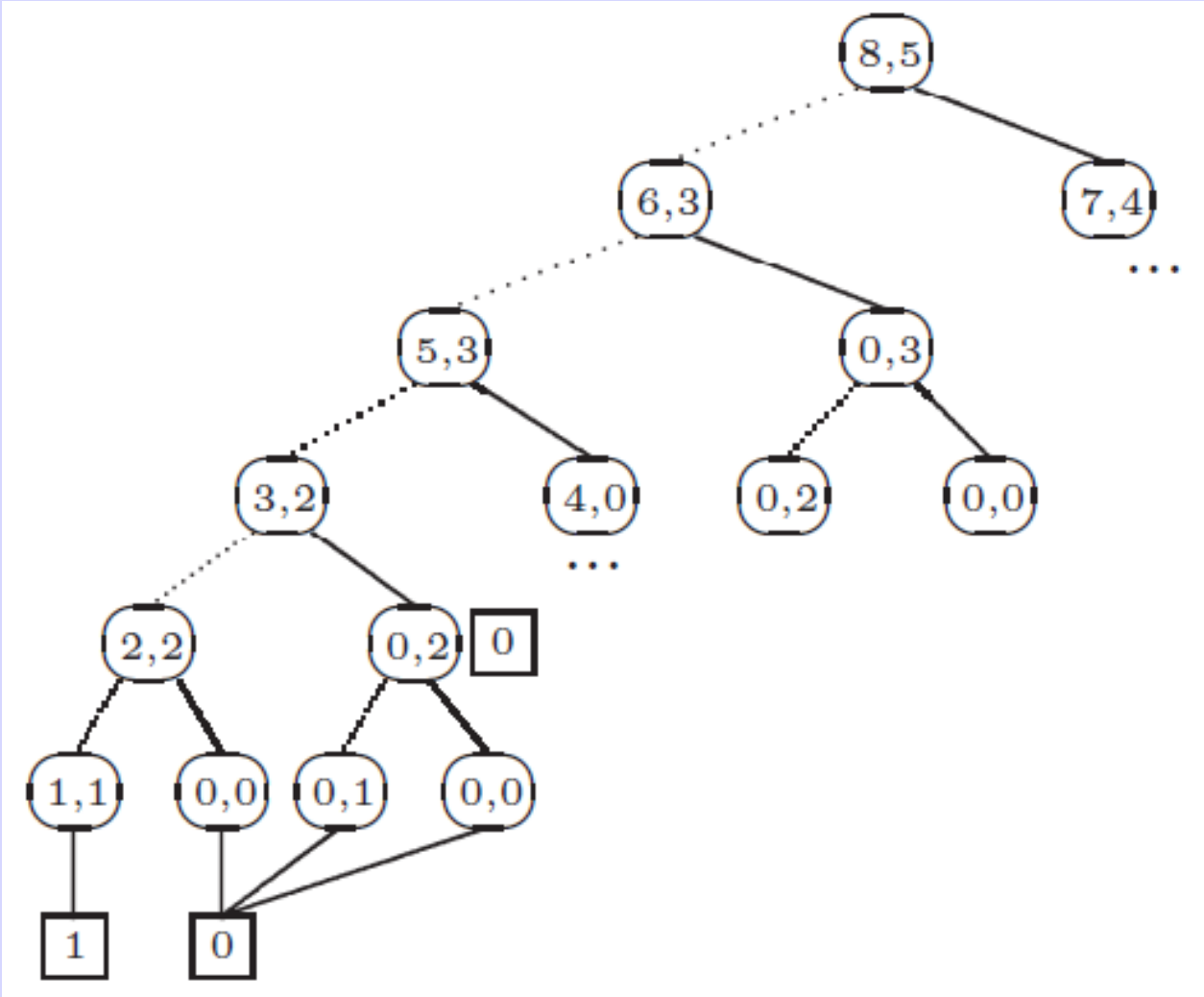
```
Apply[T,H](op,u1,u2) ≡ init(G); App[T,H](op,u1,u2)

App[T,H](op, u1, u2) ≡
    if G(u1, u2) ≠ empty then return G(u1, u2)
    else if u1 ∈ {0,1} and u2 ∈ {0,1} then u ← op(u1,u2)
            else if var(u1) = var(u2) then
                        u ← MkNode(var(u1), App(op, lo(u1), lo(u2)),
                                            App(op, hi(u1), hi(u2)))
                    else if var(u1) < var(u2) then
                                u ← MkNode(var(u1), App(op, lo(u1),u2),
                                                    App(op, hi(u1), u2))
                            else u ← MkNode(var(u2), App(op, u1, lo(u2)),
                                                    App(op, u1, hi(u2)))
                            fi
                    fi
            fi
    fi
    G(u1, u2) ← u
    return(u)
```



Beispiel zu Apply: ∧ –Verknüpfung



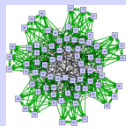
Baum aller Aufrufe im obigen Beispiel



Restriktion auf BDDs

- Algorithmus *Restrict*(u,j,b), erzeugt zu einem OBDD u einen OBDD für $u|_{j=b}$.

```
Restrict[T,H](u, j, b) :=
if var(u) > j then return u  // j not in u
else if var(u) < j then   // build new OBDD from restricted parts
        return MkNode(var(u), Restrict(lo(u),j,b), Restrict(hi(u), j,b)
else if b=0 then return lo(u)   // var(u) = j, j:=0
else if b=1 then return hi(u)   // var(u) = j, j:= 1
fi fi fi fi
```



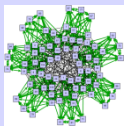
Quantifizierung

- existentielle: $\exists x\, F := F|_{x=0} \vee F|_{x=1}$

  OBDD($\exists x$ F)=Apply(or, Restrict(F,x,0), Restrict(F,x,1))

- universelle: $\forall x\, F := F|_{x=0} \wedge F|_{x=1}$

  OBDD($\forall x$ F)=Apply(and, Restrict(F,x,0), Restrict(F,x,1))



Anzahl der erfüllenden Belegungen

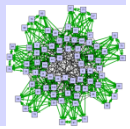
SatCount(u) =

$$2^{var(low(u))-var(u)-1} * SatCount(low(u)) +$$

$$2^{var(high(u))-var(u)-1} * SatCount(high(u))$$

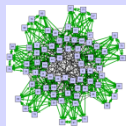
- var(low(u))-var(u)-1 zählt die übersprungenen Variablen beim Übergang von u zu low(u)
- Ohne übersprungene Variablen ergibt sich:

  SatCount(u) = SatCount(low(u)) + SatCount(high(u))



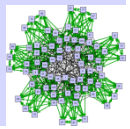
Konsistenztest auf ROBDDs

- Sei F eine boolesche Formel. G ist konsistent mit F, falls G $\wedge$ F erfüllbar ist.
- Falls F und G als BDD u1, u2 vorliegen, haben wir
  - Consistent(F, G) = SAT(u1 $\wedge$ u2) = SAT(APPLY(and, u1, u2)).
  - Der BDD APPLY(and, u1, u2) wird aber nicht wirklich gebraucht
- AndSat(u1, u2)
  - SAT((if x then t1 else t2 fi) $\wedge$ (if x then t‘1 else t‘2 fi))
    = AndSat((if x then t1 else t2 fi), (if x then t‘1 else t‘2 fi))
    = AndSat(t1, t‘1) OR AndSat(t2, t‘2)
  - SAT((if x then t1 else t2 fi) $\wedge$ t)
    = AndSat((if x then t1 else t2 fi), t)
    = AndSat(t1, t) OR AndSat(t2, t)



Konsistenztest für ROBDDs

```
init(G);

AndSAT[G](u1, u2) ≡
      if G(u1, u2) ≠ empty then return G(u1, u2) else
      if u1 ∈ {0,1} and u2 ∈ {0,1} then return (u1 ∧ u2)
                else if var(u1) = var(u2) then
                                  b ← (AndSAT(lo(u1), lo(u2)) ∨ AndSAT (hi(u1), hi(u2)))
                        else if var(u1) < var(u2) then
                                  b ← (AndSAT(lo(u1), u2) ∨ AndSAT (hi(u1), u2))
                                  else b ← (AndSAT(u1, lo(u2)) ∨ AndSAT (u1, hi(u2)))
                                fi
                          fi
                G(u1, u2) ← b;
                return b;
     fi fi
```



Bemerkungen zu AndSAT

- Der Test lässt sich wie folgt optimieren

  if u1 = 0 or u2 = 0 then return false
  else if u1 = 1 or u2 = 1 then return true
  else …

- Der Test lässt sich auf beliebige Mengen von BDDs verallgemeinern
- Der Test lässt sich verallgemeinern auf den Fall, dass u2 nicht als BDD vorliegt, aber die Variablen von u2 eine Teilmenge der Variablen von u1 sind

  (Ersetze AndSAT(lo(u1), lo(u2)) $\vee$ AndSAT (hi(u1), hi(u2))

  durch AndSAT(lo(u1), $u2|_{var(u1)=0}$) $\vee$ AndSAT (hi(u1), $u2|_{var(u1)=1}$))
- Der Test lässt sich so abändern, dass die DNF von u1 $\wedge$ u2 erzeugt wird (gebe statt „true“ jeweils die Variablenbelegung aus).