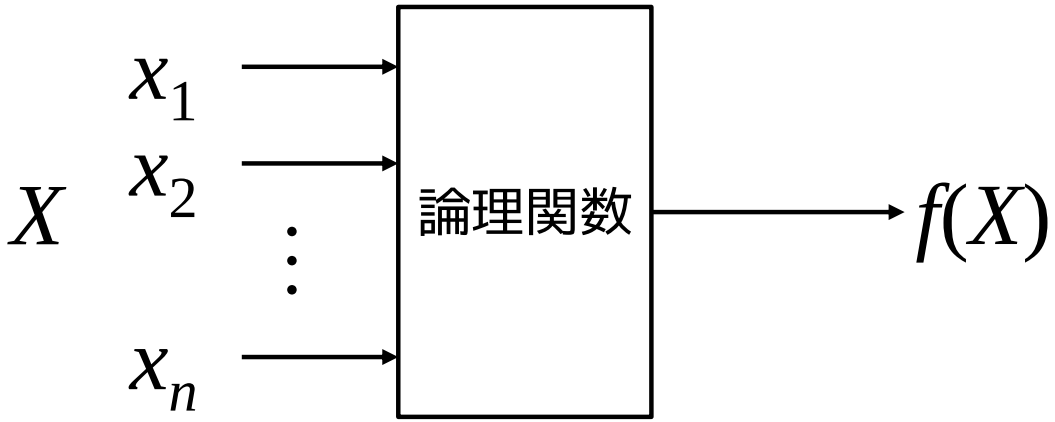
大規模論理関数処理アルゴリズムの技法

数理情報第2研究室
伝住　周平

論理関数とは

- 関数（Function）：
  - 集合 A（定義域）から集合 B（値域）への写像，
    かつ A の要素に対して B の要素が
    ただ 1 つに定まる関係のことをいう．
- 論理関数（Logic / Boolean Function）：
  - $B = \{0, 1\}$ のとき，$f: B^n \to B$ という関数 f を論理関数と呼ぶ
    （正確には二値論理関数．Switching Functionとも）．
- 一般には多値論理関数(Multi-valued Boolean Function)もある．



命題論理・述語論理

- 命題論理（Propositional Logic）:

真/偽を表す命題（proposition）変数と
それらの論理演算からなる式.
二値論理関数と等価な概念
（例） 大学生ならば人間である:

$(\mathrm{Student} \Rightarrow \mathrm{Human}) \equiv (\neg \mathrm{Student} \vee \mathrm{Human})$

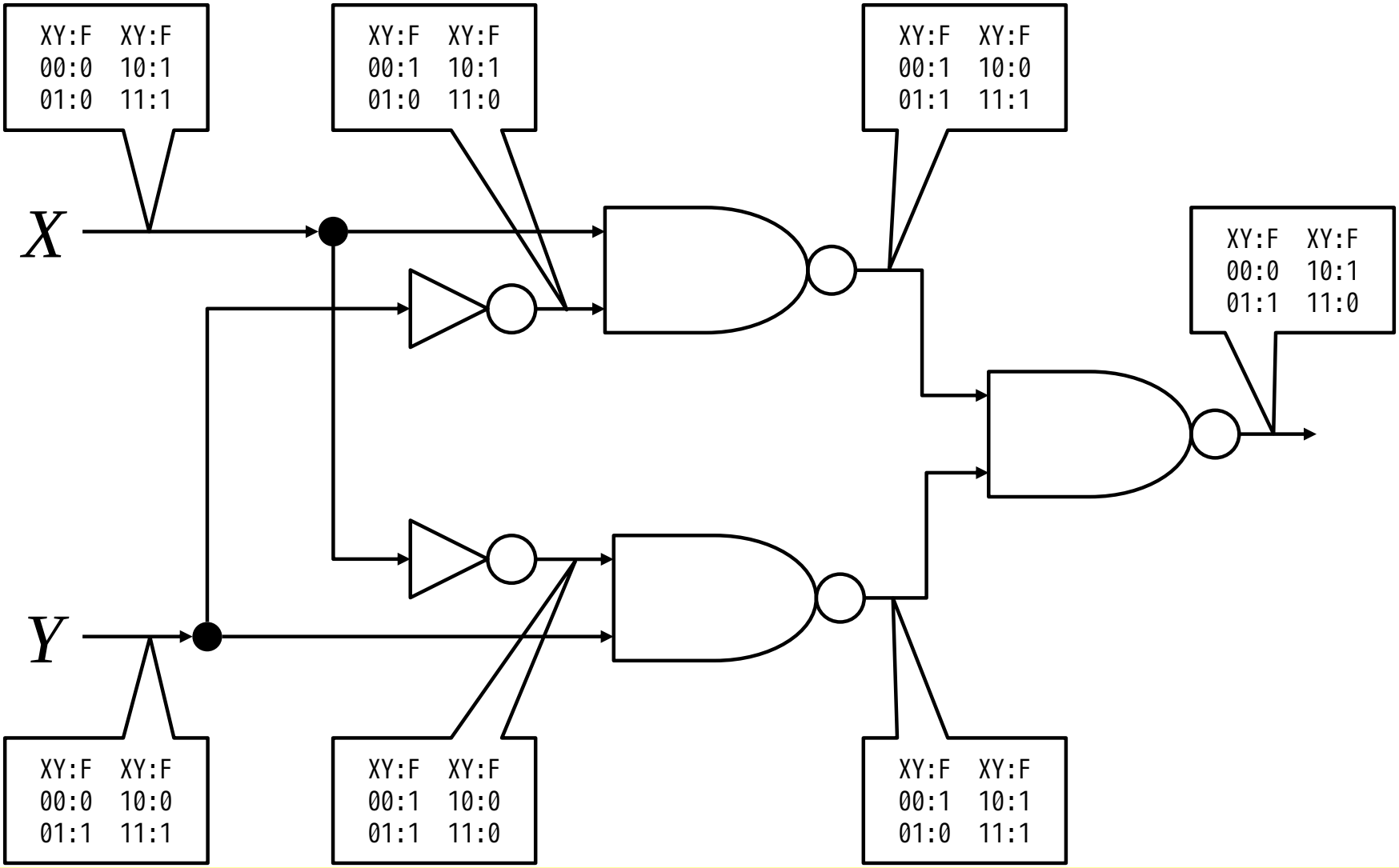
- 述語論理（Predicate Logic）:

述語（predicate）とそれらの論理演算からなる式.
述語は主語に当たる変数（二値とは限らない）を持つ.
二値論理関数よりも高度な概念.
（例） X が大学生ならば X は人間である:

$(\mathrm{Student}(X) \Rightarrow \mathrm{Human}(X)) \equiv (\neg \mathrm{Student}(X) \vee \mathrm{Human}(X))$

論理関数の処理とは(1)

・論理式/論理回路から論理関数データを作る操作



論理関数の処理とは(2)

- 恒真/恒偽判定 （→ co-NP完全問題）:

  （例） $x\,y + \sim x + \sim y\,\sim z + z$

- 等価性判定 $(F \equiv G) \Leftrightarrow (F\,G + \sim F\,\sim G \equiv 1)$:

  $x\sim y + x\,z + \sim x\,y + \sim x\sim z$

  $x\sim y + \sim x\,y + y\,z + \sim y\,\sim z$

  $x\sim y + \sim x\sim z + y\,z$

- 包含性判定 $(F \Rightarrow G) \Leftrightarrow (\sim F + G \equiv 1)$:

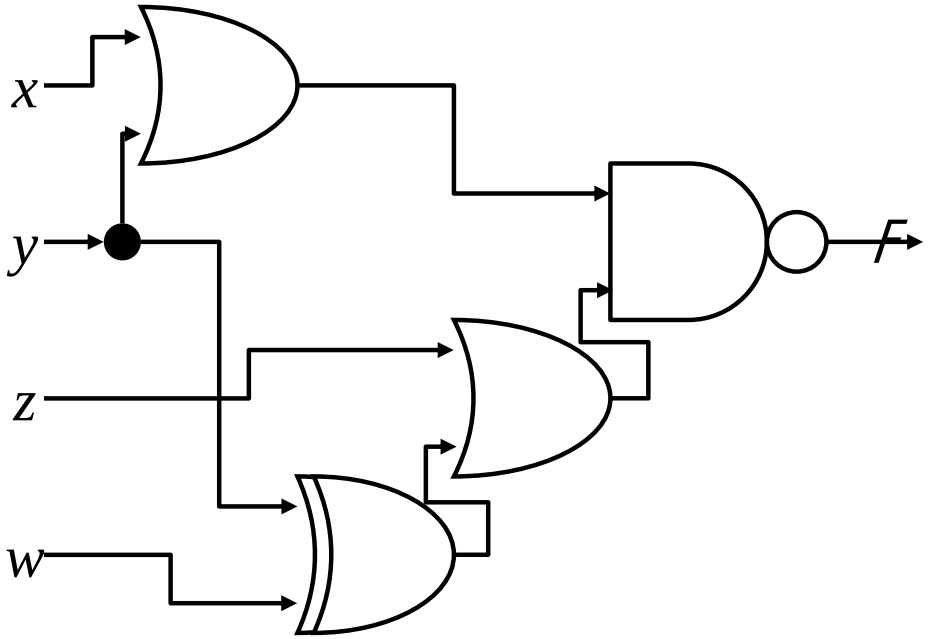
- 充足解探索:
  - 与えられた論理関数が 1 になるような
    変数値の組合せをどれか１つ求める問題（→ NP完全問題）
  - 最適化問題:
    充足解の中で最もコストが小さい（または大きい）
    解を求める問題 （→ NP困難問題）

小規模な関数の表現方法

・人間の目で見通しが良い図的表現が多く使われていた.

zw \ xy	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	1	0	0	0
10	1	0	0	0

カルノー図表現



論理回路図表現

計算機上での表現の要求

・表現がコンパクトであること

- n 入力の論理関数は $2\text{^}2^n$ 通り存在.
  → 固定長データで識別するには
  少なくとも 2^n ビットは必要. （例: 真理値表）
- 現実には全ての論理関数が均等に現れる訳ではない.
  → よく現れる関数がコンパクトに表現できる
  可変長データが使われる.

・論理演算処理が高速に行えること

- ２つの論理関数データの等価性判定が高速に行えるか
- AND, OR, NOT等の論理演算が効率よく実行できるか

・人間が見やすいことはあまり重要でない

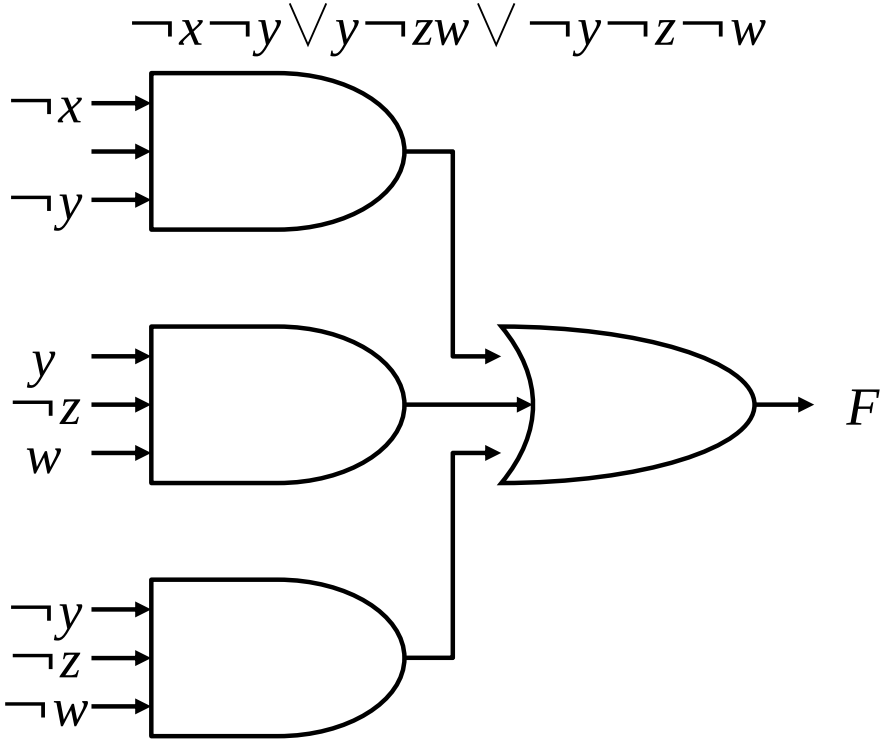
真理値表表現

- カルノー図と
  本質的には同じだが，
  見やすく並べる
  必要はない
  （単純1次元配列）
- ベクトル計算・
  並列計算に向く
- どんな簡単な
  論理関数にも，
  入力数に対して指数の
  記憶量と時間を要する
  （30変数程度が限界）
- どんな複雑な
  （ランダムな）
  論理関数でも
  同じ時間で処理できる

x_1 x_2 x_3 … x_n	F_1	F_2	F_3
0 0 0 … 0	1	0	0
1 0 0 … 0	1	0	1
0 1 0 … 0	1	1	1
1 1 0 … 0	1	1	0
0 0 1 … 0	0	0	1
1 0 1 … 0	1	1	0
0 1 1 … 0	0	0	0
1 1 1 … 0	0	0	0
・ ・ ・ ・ ・	・	・	・
・ ・ ・ ・ ・	・	・	・
・ ・ ・ ・ ・	・	・	・
1 1 1 … 1	0	1	0

積和形論理式

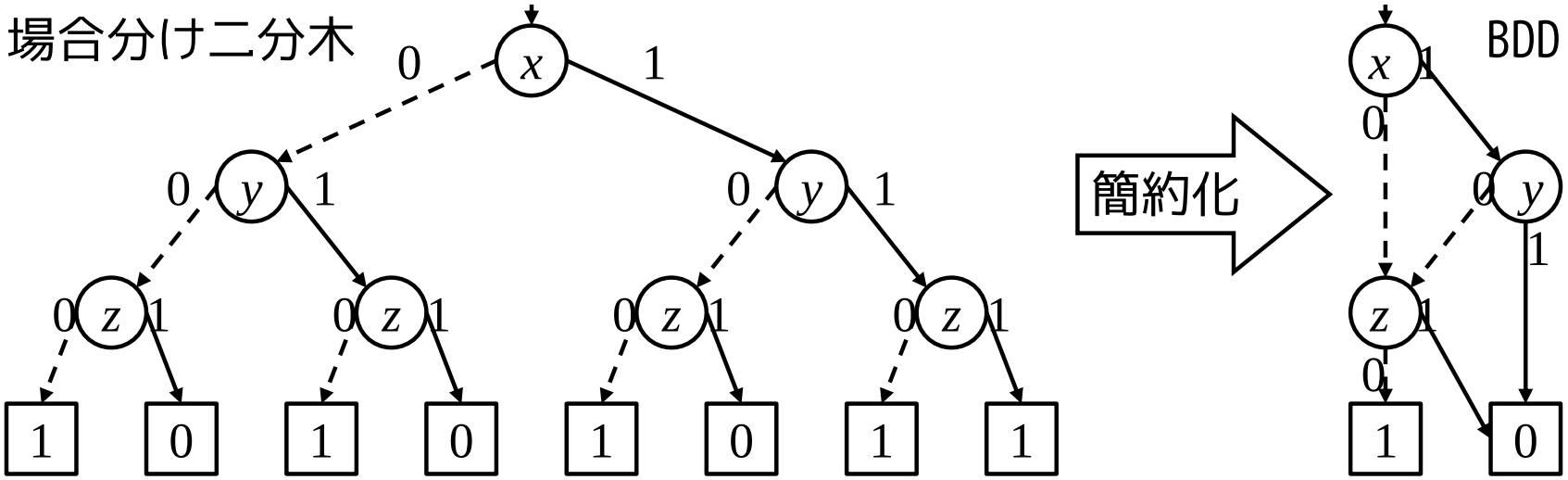
- 肯定・否定の２種類の入力変数を用いた積和２段の論理式で表す方法.
- 記憶量は論理式に現れる文字数にほぼ比例.
- 簡単な式でかける論理関数は効率よく処理できる.
- 表現が一意でなく等価性判定に時間がかかる.
- 否定演算やXOR演算が苦手.
- BDDが出現するまでは最も多く使われた（今も使われている）.



x	y	z	w	F
0	0	*	*	1
*	1	0	1	1
*	0	0	0	1

BDD（二分決定グラフ）

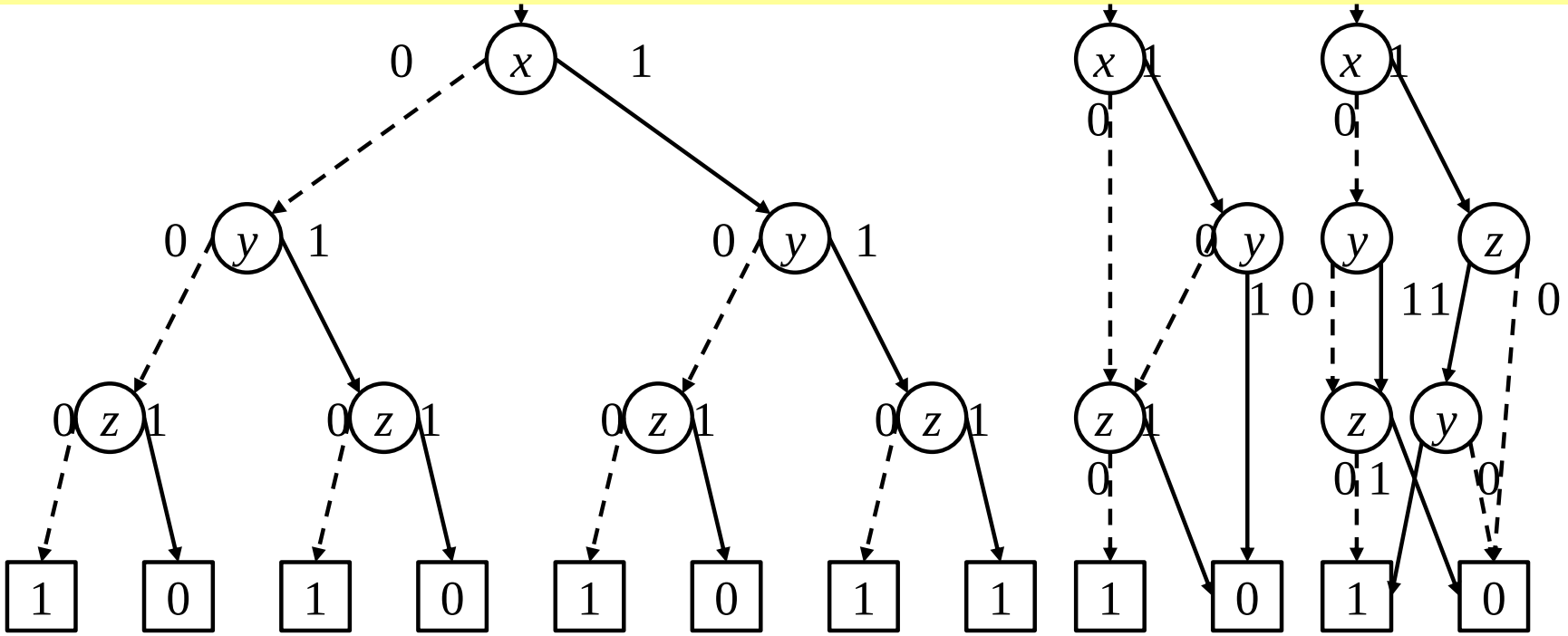
・Binary Decision Diagram



・二分木状の「場合分けグラフ」によるデータ構造
（計算機上ではポインタの配列で表現）

・1980年代後半から急速に研究が進み実用化

BDDとは



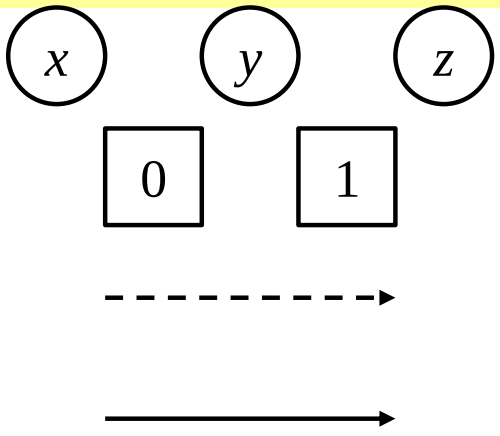
真理値表と等価なBDD
(Binary Decision Tree)

既約な順序付きBDD (Reduced Ordered BDD)

既約でも順序付きでもないBDD (Unordered BDD)

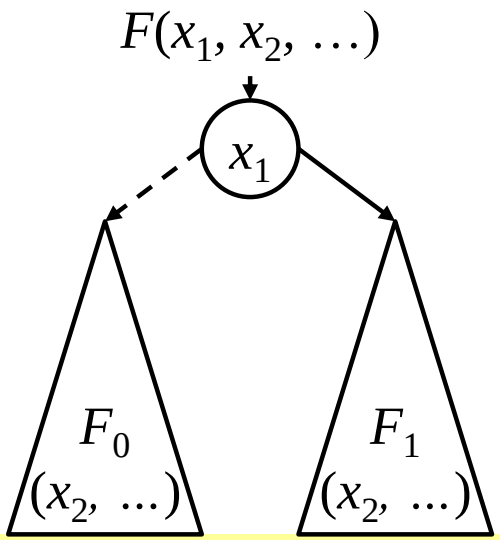
BDDの節点と枝

- 分岐節点（decision node）
- 終端節点（terminal node）
- 0-枝（0-edge）
- 1-枝（1-edge）
- 部分グラフ（sub-graph）



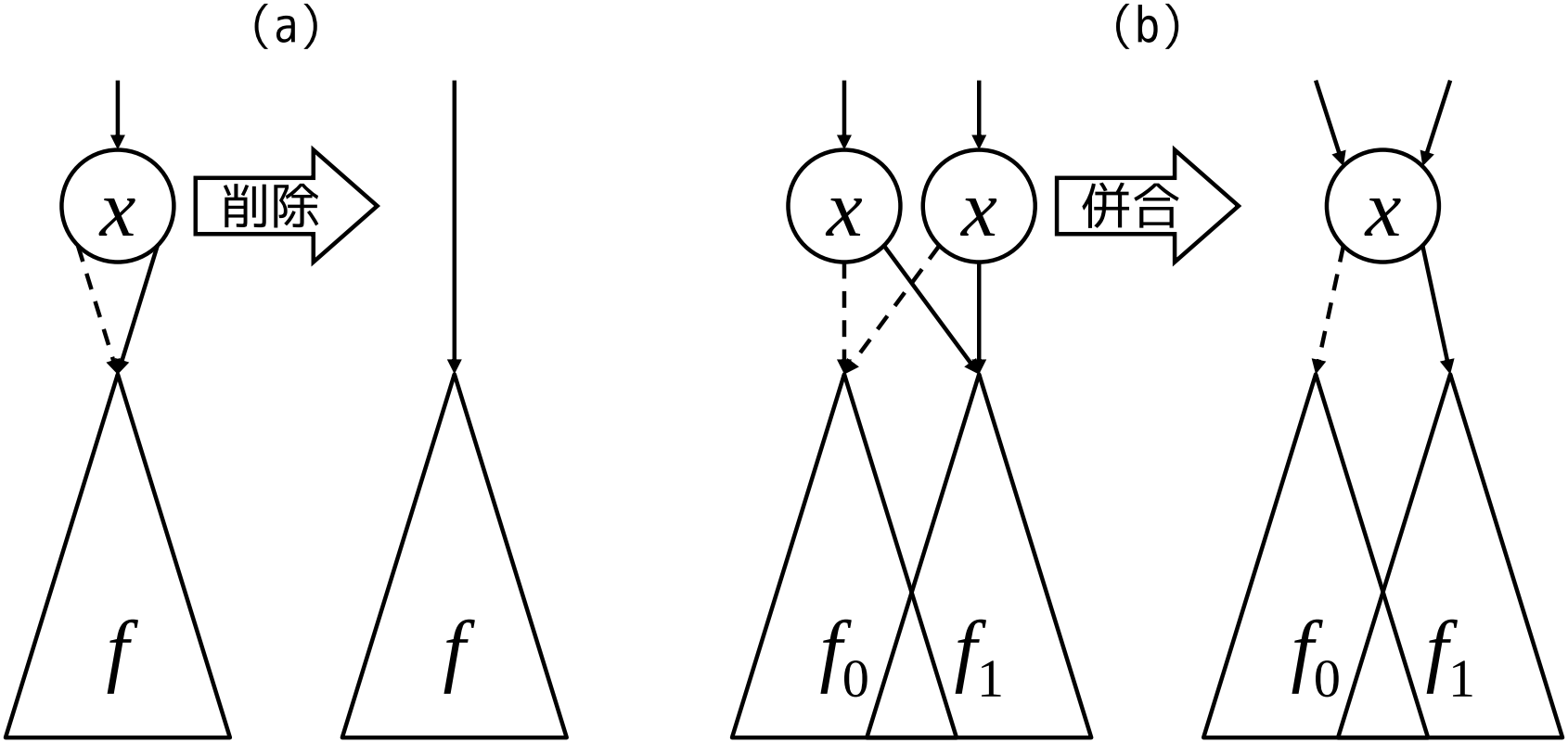
シャノンの展開
（Shannon’s expansion）

$$F(x_1,\ldots,x_n) = \neg v\, F(0, x_2, \ldots, x_n) \vee v\, F(1, x_2, \ldots, x_n)$$



BDDの簡約化規則

(a) 冗長な節点を全て削除
(b) 等価な節点を全て共有
⇒既約なBDDが得られる



参考: (b)の規則だけを可能な限り適用した形を
「準既約」(Quasi-reduced)なBDDと呼ぶこともある.

BDDの特徴

- 論理関数に対してグラフの形が一意に定まる.
  - 等価性判定が非常に容易

- 多くの実用的な論理関数がコンパクトに表現できる.
  - パリティ関数や加減算回路も効率よく表現
  - 性質の良い関数では数百入力まで扱える

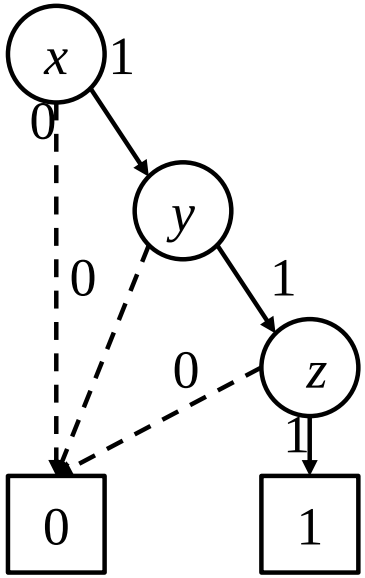
- 論理関数同士の演算が,
  グラフのサイズにほぼ比例する計算時間で実行できる.
  - 否定演算も容易

- グラフのサイズが小さくならない場合もある.
  - 乗算回路のBDDは指数サイズ

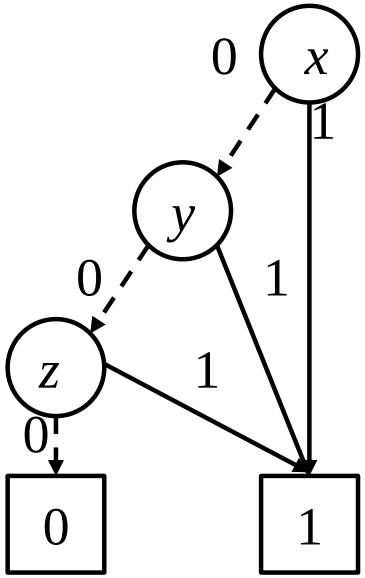
- 変数の順序づけが悪いとグラフが大きくなる.
  - 比較的良い順序づけを得る方法がいくつか実用化
  （厳密最小化はNP完全問題）

論理積・論理和・パリティ

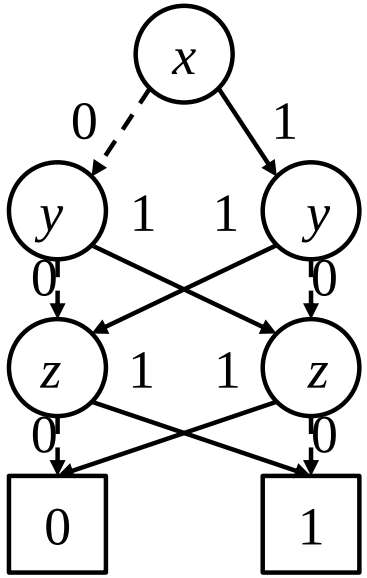
・変数は n 個
・いずれも，n に比例する節点数 $O(n)$ で表現可能
・0と1の終端節点を入れ替えるとそれぞれの否定論理になる



論理積(AND)



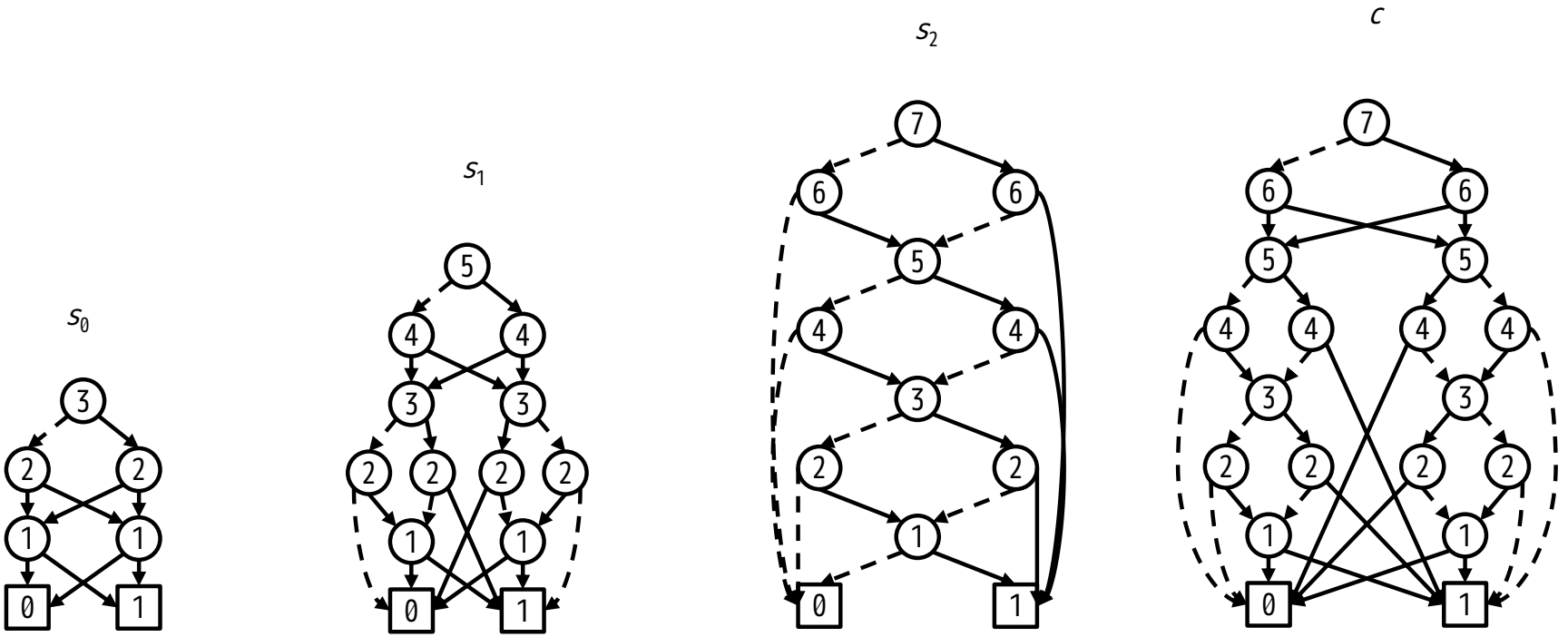
論理和(OR)



排他的論理和(XOR)
・パリティ

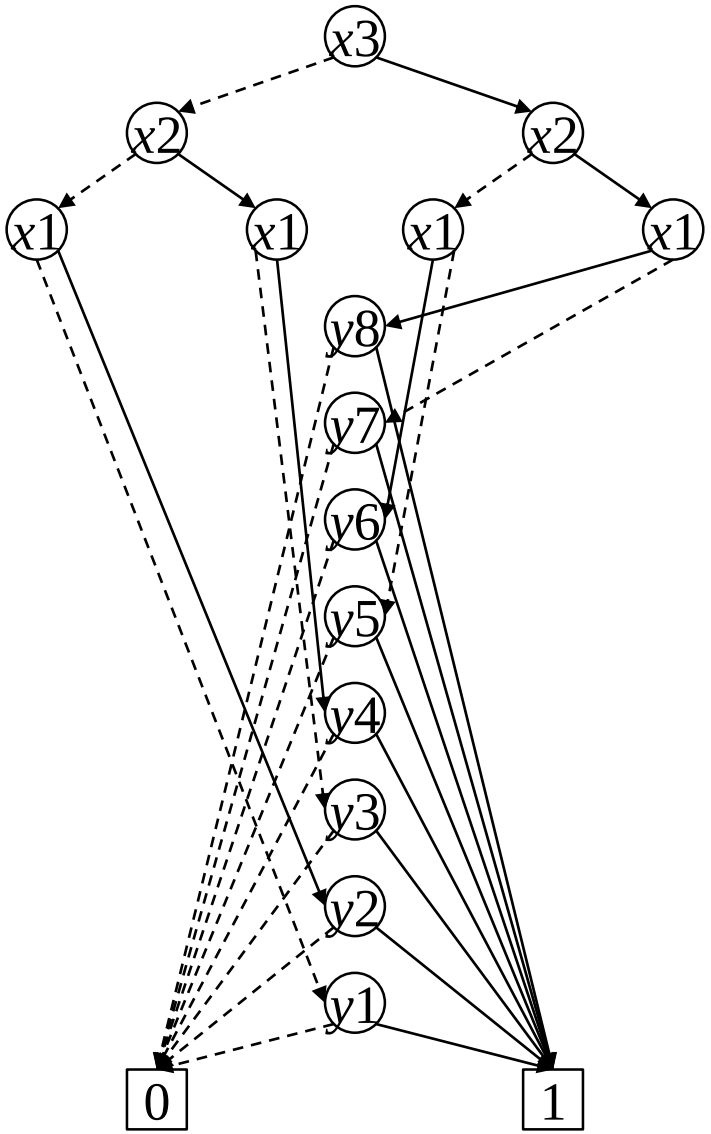
n ビット２進数の算術加算

３ビット加算器 $(a_2a_1a_0)_2+(b_2b_1b_0)_2+(c_0)_2=(s_2s_1s_0)_2$ の論理関数のBDD
（変数は $a2$，$b2$，$a1$，$b1$，$a0$，$b0$，$c0$ の順）



・n が増えてもBDDは縦方向に伸びるだけで幅は一定
→ 節点数は $O(n)$
・減算も同じく $O(n)$ となる．２進数の大小比較も $O(n)$

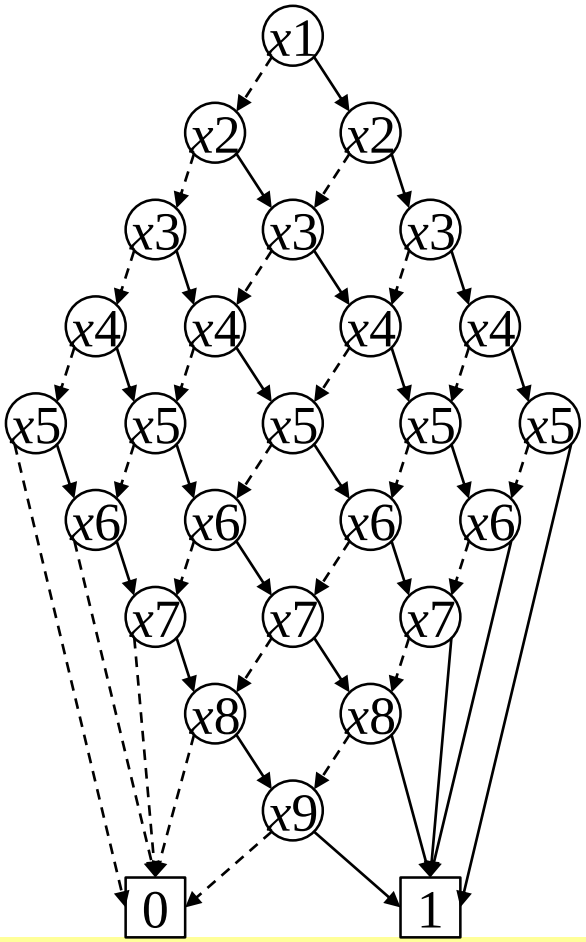
n 入力データセレクタ



- n 個のデータ入力から
  １つを選んで出力する論理関数
- Ceiling($\log_2 (n+1)$) 個の制御入力で,
  どの番号のデータを
  選ぶかを指定する.
- BDDの節点数は $O(n)$ で表現可能

対称(symmetric) 論理関数

9入力変数の
対象関数の例
（多数決関数）



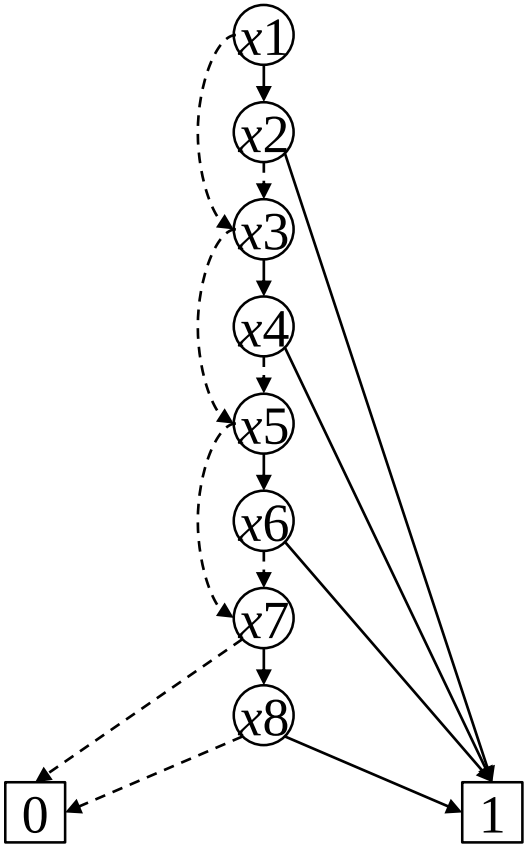
- 対称関数：変数の順序を
  入れ替えても論理に
  影響がない論理関数
  - n 個の入力変数のうち，
    1 になっている変数の
    個数によって出力の値が決まる

- BDDの場合，各段ごとに，
  最上位からその変数までの
  “1”の個数を示している．
  - BDDの幅は最大で n
  - 全体の節点数は $O(n^2)$

変数順序付けの影響(例1)

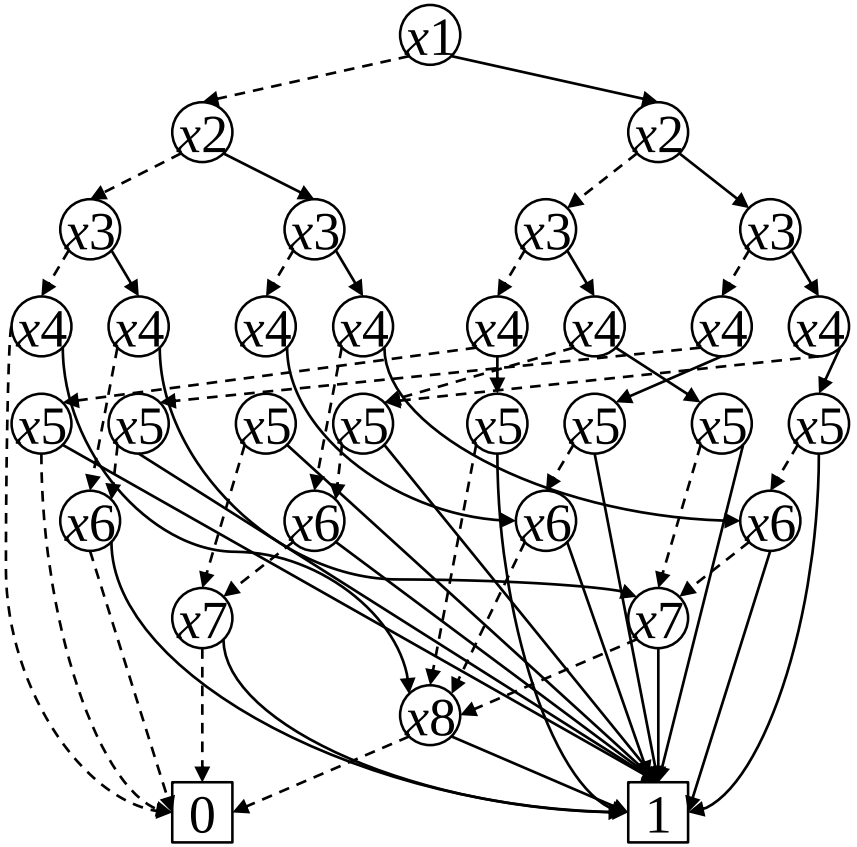
・BDDは変数の順序により節点数が著しく変化する場合がある.

$x1x2 \vee x3x4 \vee x5x6 \vee x7x8$



$O(n)$

$x1x5 \vee x2x6 \vee x3x7 \vee x4x8$

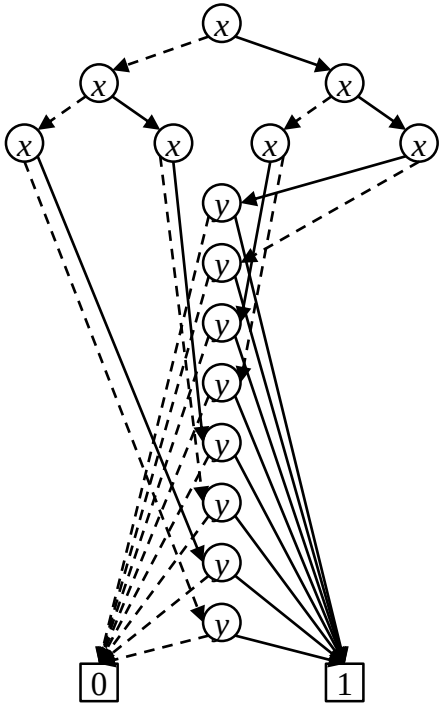


$O(2^n)$

変数順序付けの影響(例2)

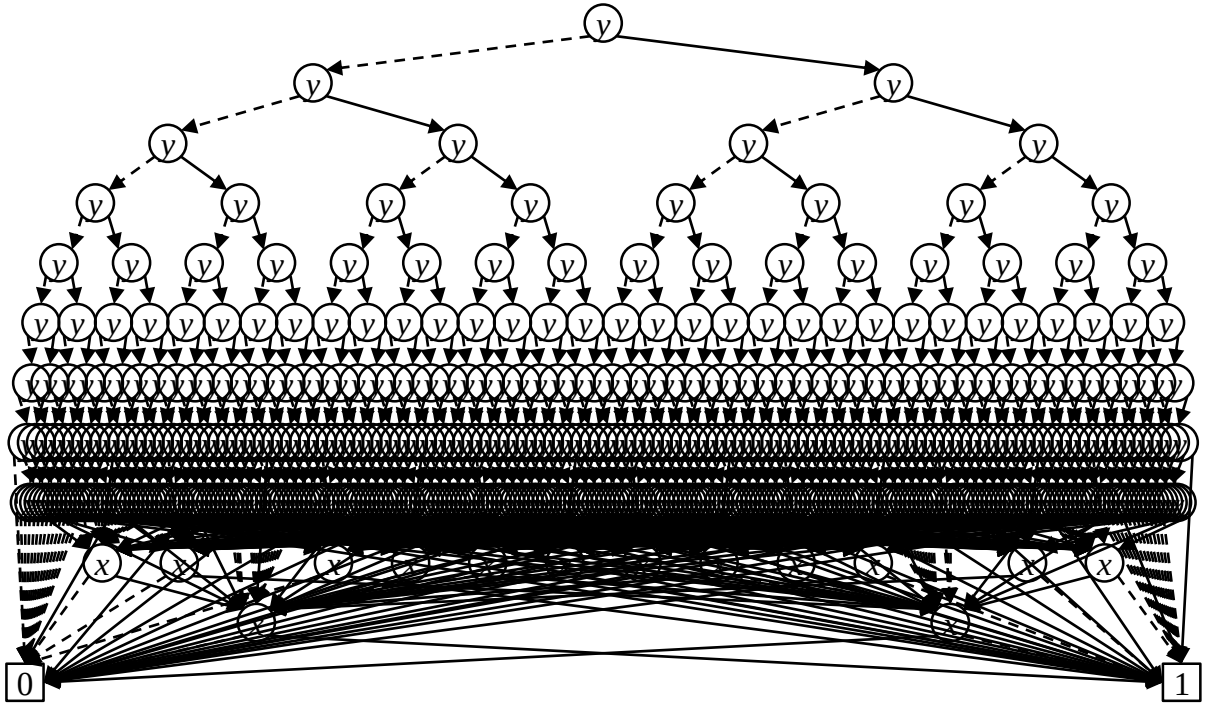
・8入力データセレクタ

制御入力が上位



$O(n)$

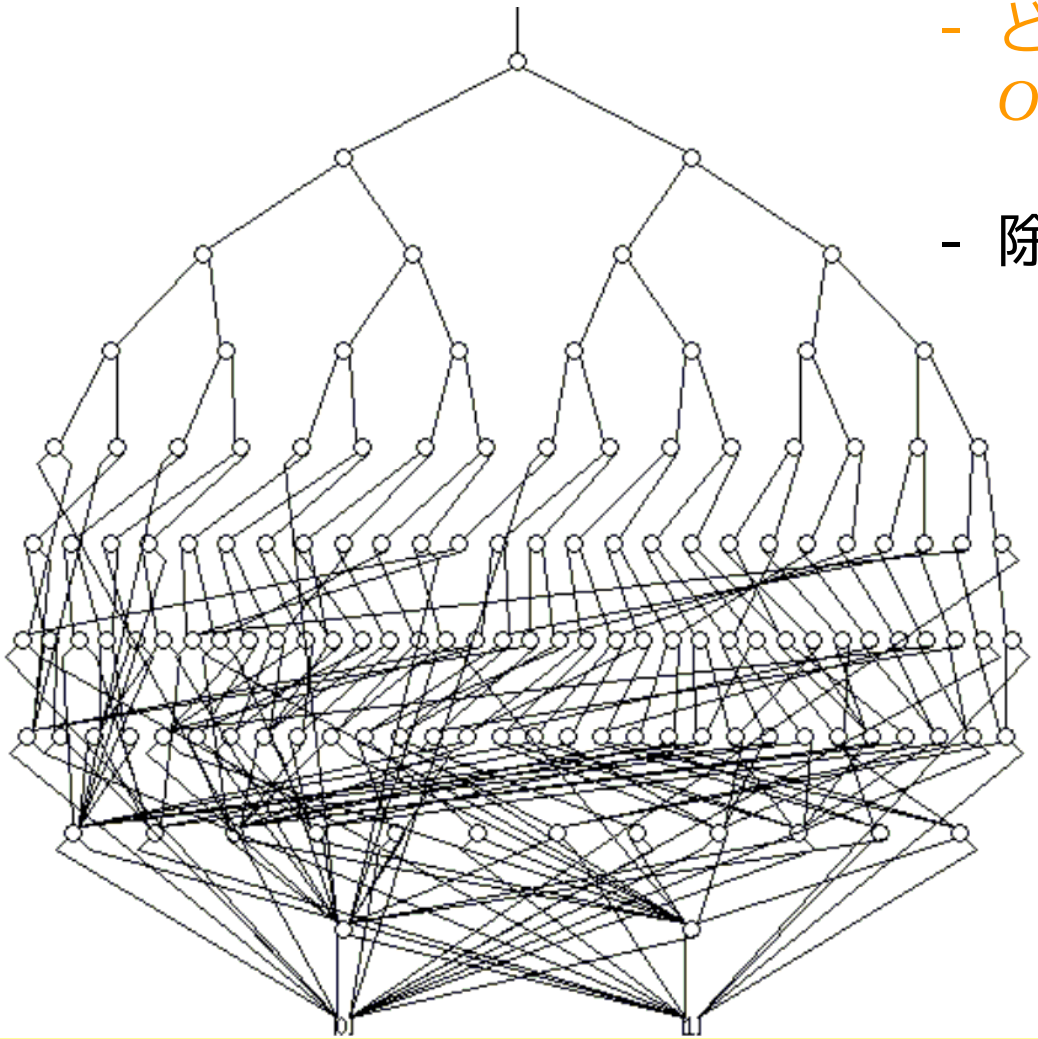
データが上位



$O(2^n)$

変数順序付けの影響(例3)

5×5ビットの乗算器の出力
(第5ビット)



・２進数の算術乗算は苦手
- どんな変数順序でも，
  $O(2^n)$ の節点数になる

- 除算も同様に常に指数オーダ

変数順序の影響パタン

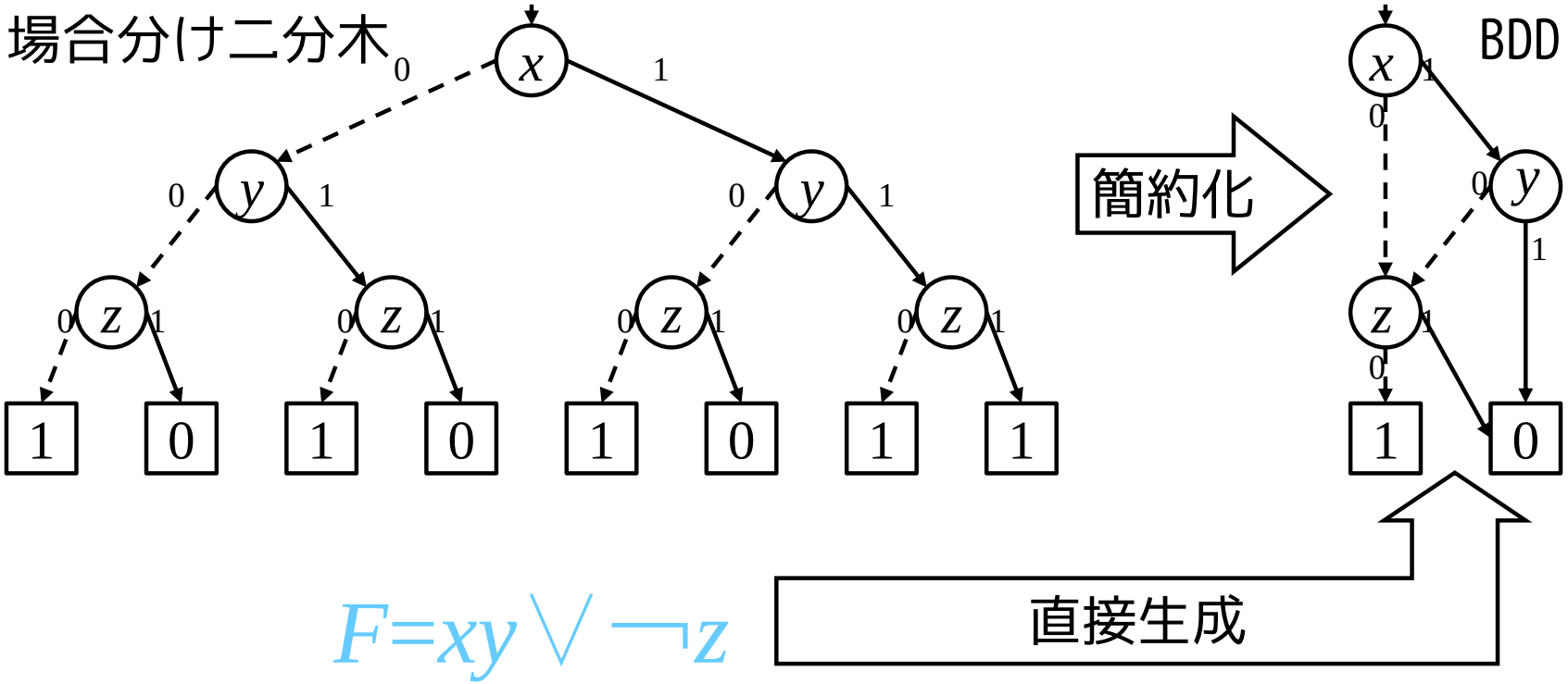
(a) どんな順序でも簡単

(b) 順序付けにより簡単
だったり複雑だったり

(c) どんな順序でも複雑

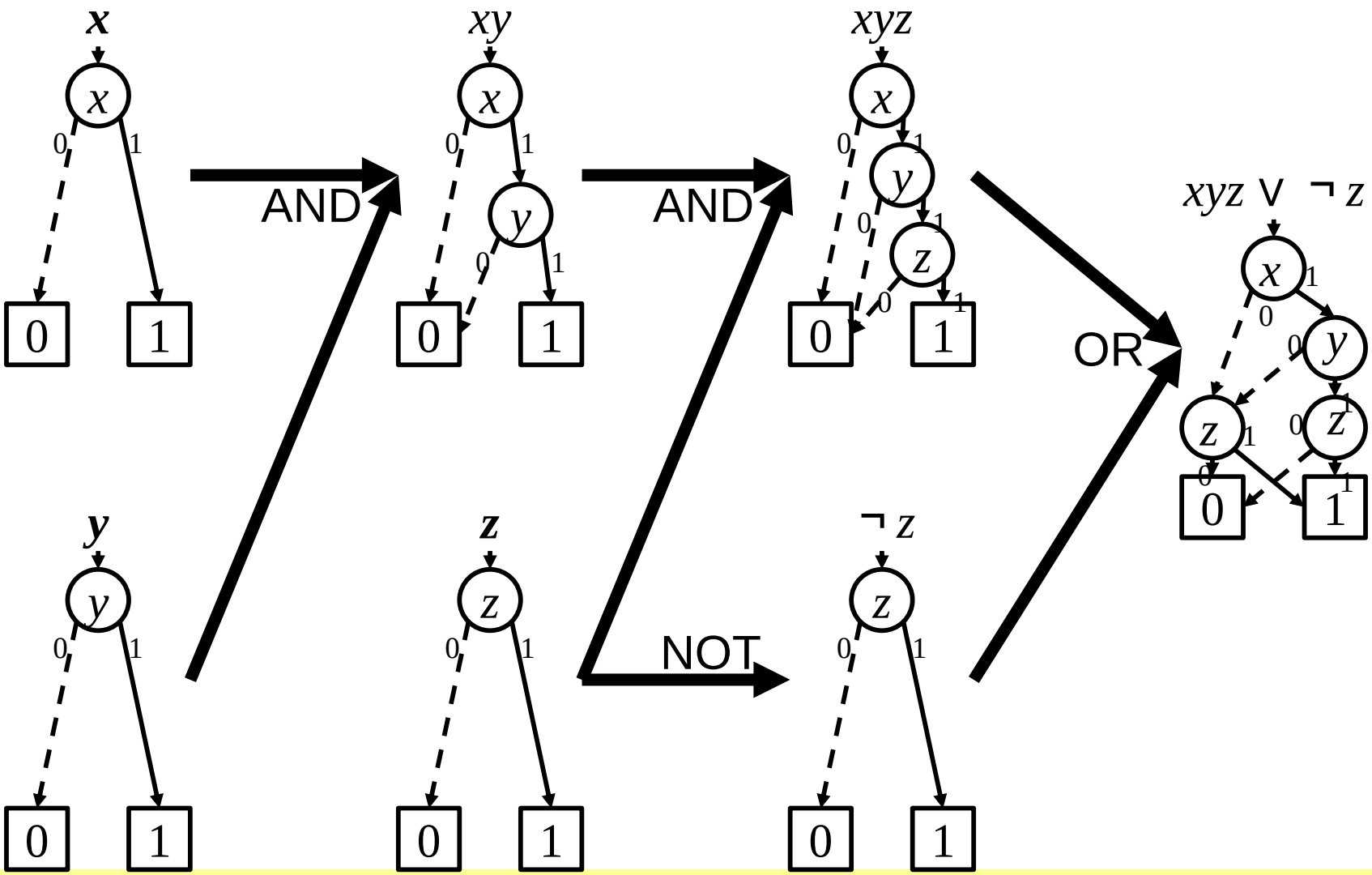
BDDの生成アルゴリズム

・真理値表に対応する二分木を簡約化する方法では，
常に指数オーダの記憶量と処理時間がかかってしまう．
・実用的には，論理式からBDDを直接生成
するアルゴリズム[Bryant86]を用いる

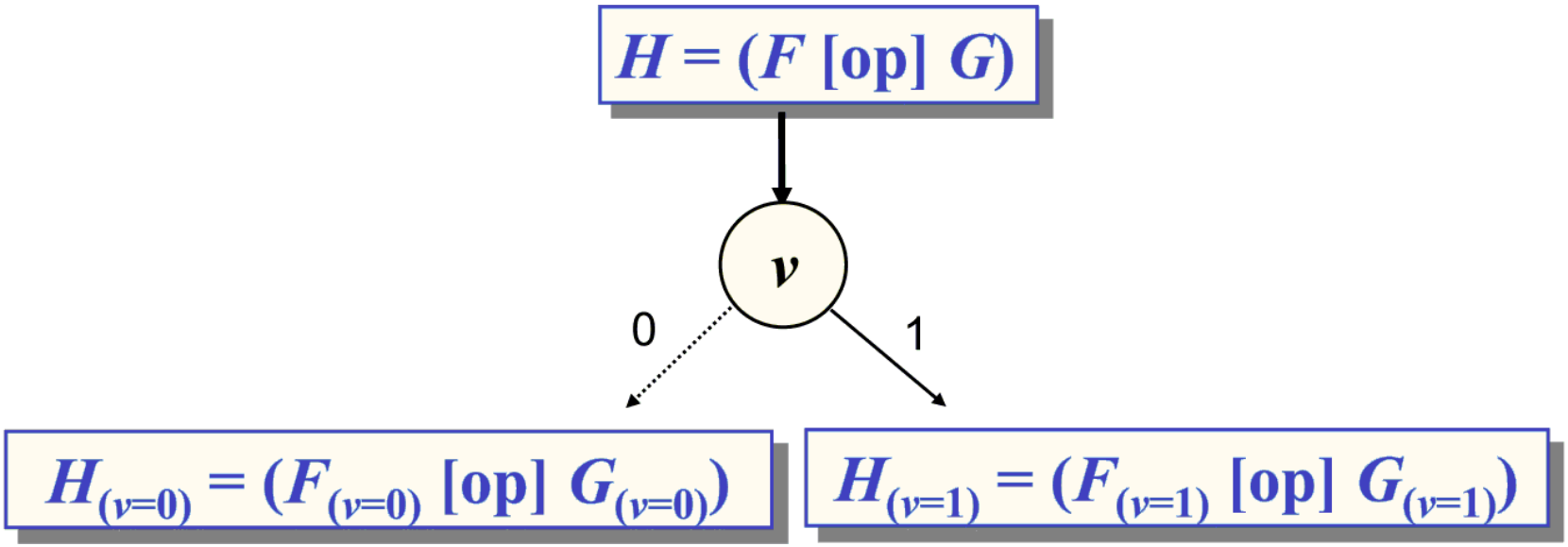


論理式からのBDD生成

・２つのBDDの間の二項論理演算を繰り返して目的のBDDを生成

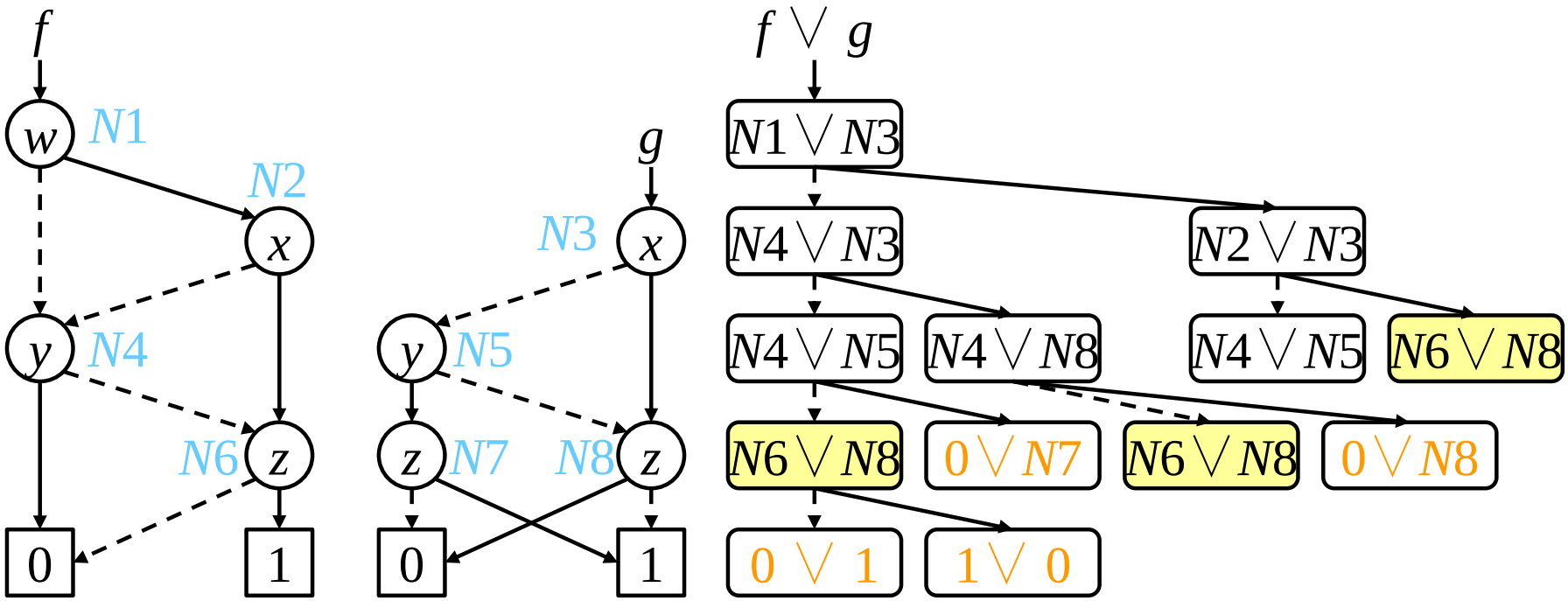


論理演算アルゴリズム



- ある変数 x に $0, 1$ を代入して再帰的に展開
- 全ての変数を展開すると自明な演算になる
  (OR演算の例) $F \vee 1=1, F \vee 0=F, F \vee F=F \ldots$
- 各演算結果をBDDに組み上げる

二項論理演算の実行例


f
w
N1
N2
x
y
N4
N6
z
0
1
g
N3
x
y
N5
z
N7
N8
z
0
1
f ∨ g
N1∨N3
N4∨N3
N2∨N3
N4∨N5
N4∨N8
N4∨N5
N6∨N8
N6∨N8
0∨N7
N6∨N8
0∨N8
0 ∨ 1
1∨ 0

充足解探索・最適解探索

- 既約なBDDが与えられたとき，
  その論理関数が充足可能かどうか（恒偽関数でないかどうか）
  の判定は定数時間で可能

- 恒偽関数でない場合に，
  充足解（論理を 1 にするための各入力変数の 0,1 の割当）
  を求めることも容易
  - 0終端節点を指さない枝を辿れば必ず1終端節点に到達可能
  - グラフの変数の数に比例する時間

- 入力変数に 1 を代入するときのコストが与えられているとき，
  コスト最小の充足解（最適解）を求められる
  - グラフの節点数に比例する時間

- 論理関数が 1 になる割合（真理値表密度）も求められる
  - グラフの節点数に比例する時間

BDDの改良技術

・BDDの演算処理を効率化するための様々な改良技術が研究開発されている

・実用上，重要な技法として以下の２つがある

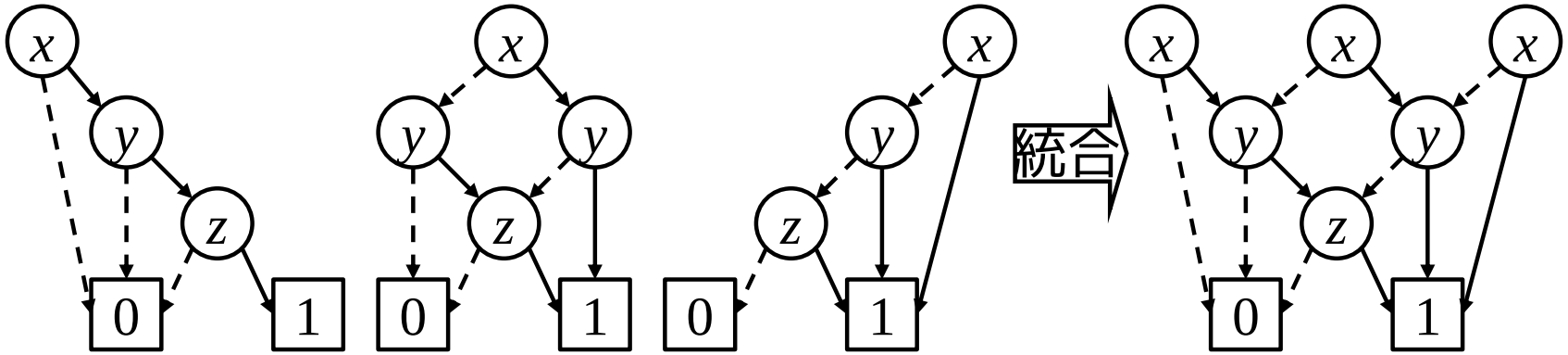
1．複数のBDDを統一的に共有して扱う技法

2．否定枝の技法

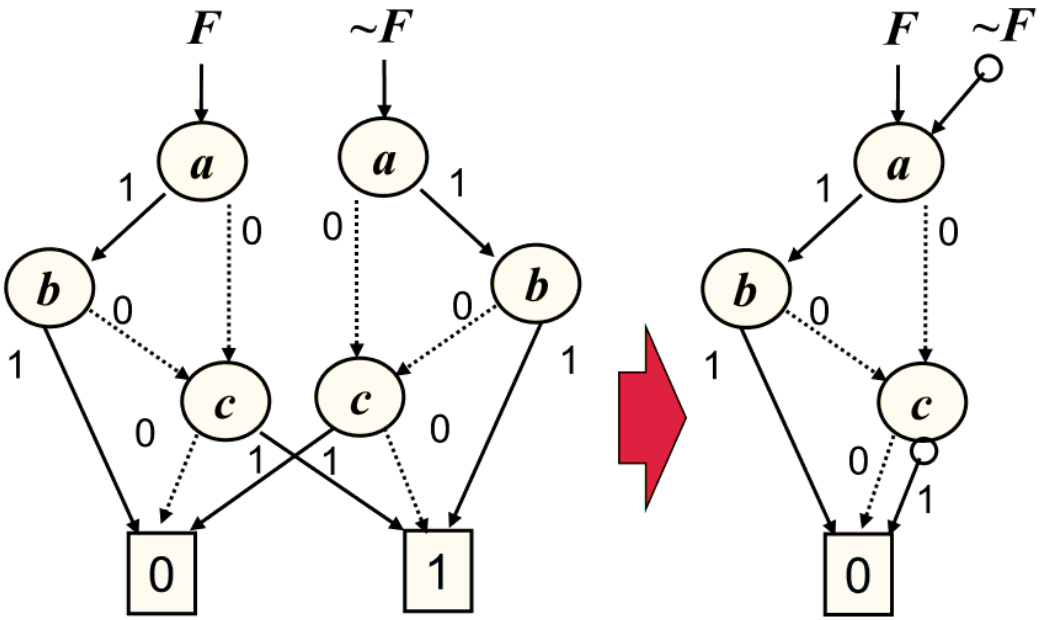
この２つの技法は現在のほとんどのBDD処理系で用いられている

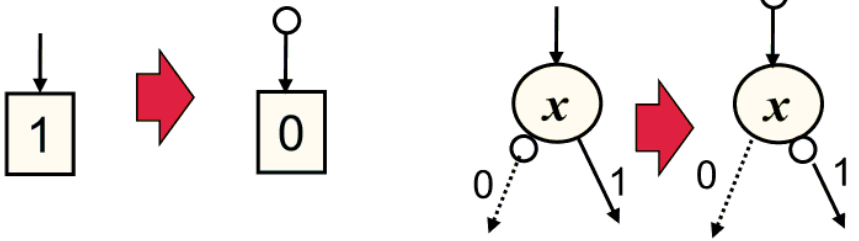
複数のBDDの共有化

- 変数の順序を揃える
- 全てのBDDを１つのグラフに共有化
  - 等価な部分グラフをまとめる



否定枝(Negative edge)





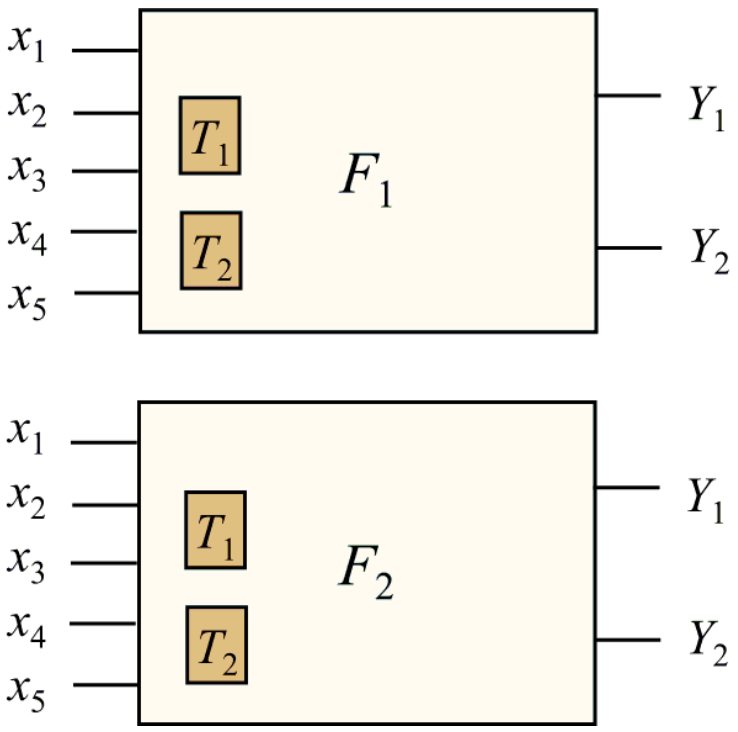
- 否定演算を表すマークを枝につけることで，否定同士の関係にあるBDDを共有化する技法
- 否定演算が定数時間で実行可能に
  - グラフの根の枝の否定枝をon/offするだけ
- グラフの一意性を保つため２つの制約を導入
  - 1終端節点は使用しない
  - 0枝は否定枝にできない

LSI設計技術への応用

- BDDは元々は計算機ハードウエア設計のために研究・開発された
  - 論理シミュレーション・論理検証
  - 論理最適化・論理合成
  - テストパタン生成
- LSIの論理設計は非常に大規模な論理関数データを扱う困難な問題
  - 人手では手に負えない規模
  - 誤りが許されない（Pentiumのバグの例）
  - 設計期間の短縮の要求（企業間の開発競争）
  - 巨大なマーケット

順序回路の論理検証

・レジスタ（記憶素子）を含む順序回路の等価性判定問題



すべての入力の系列に対して，
同じ出力系列となるかを
チェックする問題
（組合せ回路よりはるかに難しい）
レジスタの値によって回路の
「状態」が決まる．

２つの回路が取りうる
状態の対応関係が分かれば，
対応する各状態において
組合せ回路と同様に等価性判定

論理合成・最適化へ応用

- 与えられた仕様を満たす品質の良い
  （素子数や段数が小さい）回路を自動生成したい

- 適当な初期回路を生成しこれを最適化する

- 論理回路の部分的変形を繰り返して品質を改善する方法
  - 部分的変形を行うときに
    回路全体の論理に影響を与えないことが条件
  - 条件判定にBDDを用いることで，それ以前に比べて
    数十～数百倍も大規模な論理回路を最適化可能に

- 適当な初期回路として積和形論理式を作り，
  これを因数分解してコンパクトな回路を生成する方法もある
  - BDDを用いて，積和形論理式の生成と因数分解を
    高速化する技法も開発されている

テストパタン生成

- LSI製造時に小さなチリが混入すると故障が発生
  - 何%かの確率で発生
  - 出荷時に検査して取り除く

- 出荷時にLSI内部を直接見ることは不可能
  - ある入力パタンを与えたときに，期待値と異なる出力が観測されたときに初めて不良品と確認できる
  - 故障を発見するための入力パタン（＝テストパタン）をすべての故障箇所について求めておく
  - 出荷時の手間を減らすため，テストパタンは少なくしたい（少ないパタンでなるべく多くの故障をみつけたい）

- テストパタン生成には大規模な論理演算処理が必要
  - BDDを用いた高速化が有効

組合せ問題・最適化応用

- グラフ理論で扱われる基本的な問題にBDDを応用
  - （例） 最大クリーク問題，最大独立集合問題，等

- 与えられたグラフの枝（または節点）の集合の中から，
  ある条件を満たす部分集合を見つけ出す問題が得意

- グラフの枝（または節点）に論理変数を割り当て，
  変数の組合せの集合をBDDで表現

- 満たすべき条件を論理式で表し，そのBDDを生成できれば，
  コスト最小となる変数組合せを即座に求められる．
  - ただし，大規模な例題に対してはBDDのサイズが
    爆発的に増大して主記憶に入りきらない場合がある．
    常に簡単に解けるわけではない．

組合せ最適化問題の例

・ナップザック問題

・商品が n 個，それぞれ値段と重さが決まっている．
手持ちのナップザックの許容重量の範囲で，
合計金額が最大となるように商品を組合せたい．
- 買うかどうかを変数 x_i の $1,0$ で表現する．
- 各商品の重さを a_i，値段を c_i，
ナップザックの許容重量を b とすると，
$\Sigma a_i x_i \leqq b$ の条件下で，$\Sigma c_i x_i$ が最大になるような
x_i の $0,1$ の組合せを求めればよい．

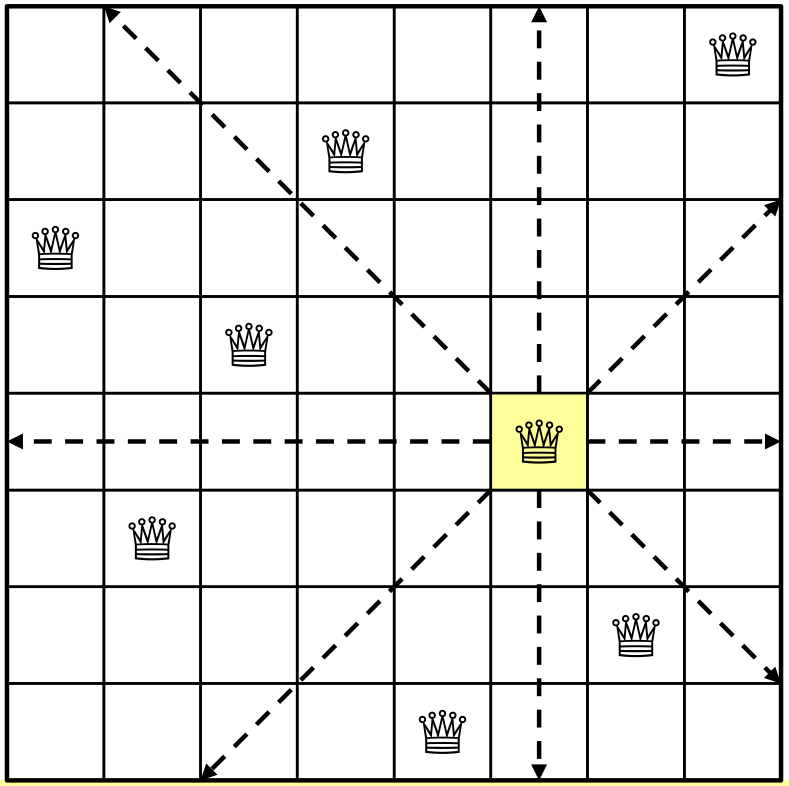
・制約条件 $\Sigma a_i x_i \leqq b$ は，$x_1, \ldots, x_n$ を入力とする論理関数となる．
- BDDで表すと，根から1終端節点に至る経路が，
ナップザックの許容範囲の商品の組合せを示している．
- 全ての経路の中で，
コストの合計が最大になるような経路を選べばよい．
これはBDDの節点数に比例する時間で求められる．

組合せ問題の例

・８クイーン問題

・８×８のチェス盤上にクイーン（縦横斜めに利く）を８個，お互いに取り合わないように配置する問題

・論理変数を64個用意する
  - 各論理変数は，各マス目のクイーンの有無を 0,1 で表現
・クイーンの条件を論理式で記述する
  - 各列にクイーンは１つだけ
  - 各行にクイーンは１つだけ
  - 各斜め筋にクイーンは0または1個
・全ての条件の論理積(AND)を表すBDDを生成すれば解の個数が直ちにわかる



その他の組合せ問題

- ナイト巡回問題:
  - チェス盤上をナイト（８方向の桂馬）が，各マス目１回ずつ通って元の位置に戻ってくる一種の一筆書きパズル
  - BDDで解いたという論文[Lobbing et al. 96]がある.
- 巡回セールスマン問題
  - N 個の都市をなるべく短い経路で１回ずつ巡回する問題.
  - BDDで解く方法の一例: 2つの都市を結ぶ枝は $N(N\text{-}1)/2$ 個存在. 各枝に論理変数を割り当てる．通る通らないを 1,0 で表現. 一巡して元に戻る経路（ハミルトン路）の全てを表すBDDを作り，その中でコスト（距離）最小の組合せを選ぶ.

N	論理変数	節点数	解の個数	計算時間
8	28	2,054	2,520	8.7s
9	36	6,472	20,160	28.8s
10	45	19,972	181,440	216.5s

グラフ理論の種々の問題

・その他にBDDで扱えるグラフ理論の問題の例:

- 最大クリーク問題 (Maximal cliques)
- 最大kカバー問題 (Maximum k-cover)
- 最小α被覆問題 (Minimum α-covering)
- 最大独立集合 (Maximum independent set)
- 最小節点カバー (Minimum vertex cover)
- 最小彩色問題 (Minimum coloring)
- 最小クリーク分割 (Minimum clique partition)
- 最小クリークカバー (Minimum clique cover)
- 最小支配集合 (Minimum dominant set)

・BDDでない方が効率が良い場合もあり
問題の性質に依存するので要注意

専用の解法との比較

- BDDを用いた解法では，
  制約条件を論理式で記述できれば後は機械的に解ける
  - 与えられた問題に対して解法を素早く実装できる
  - 多数の解をBDDで共有して全部列挙できる

- 一方，有名な問題に対しては
  専用の効率的なアルゴリズムが開発されていることが多い
  - 問題固有の性質を利用しているので
    BDDを用いた場合よりも高速に解けることが多い
  - 特に，
    解を１つだけ見つける問題に対しては専用の解法が強い．
    例えば，巡回セールスマン問題にはある条件のもとで，
    1000都市でも最適解を見つける方法が知られている．
    BDDで全解を列挙する方法では15都市程度が限界．

論理関数と組合せ集合

論理関数: $f = (\neg x\, y) \vee (x\, z)$

組合せ集合: $F = \{\{x, y, z\}, \{x, z\}, \{y\}, \{y, z\}\}$

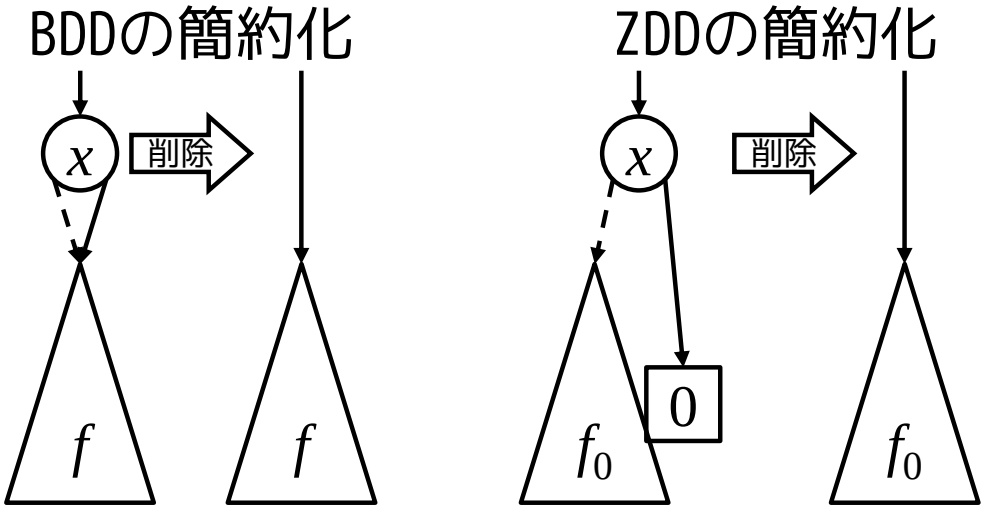
x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
f	0	0	1	1	0	1	0	1
			y	yz		xz		xyz

- 組合せ集合と論理関数の演算は対応関係がある
  - 和集合(Union) → 論理和(OR)
  - 積集合(Intersection) → 論理積(AND)
  - 補集合(Complement) → 論理否定(NOT)

ZDD(ゼロサプレス型BDD)

・**「組合せ集合」**を効率的に表現するためのBDDの改良[湊93]

- 通常と異なる
  簡約化規則を考案.
- 疎な集合の族を扱う時
  著しい効果が得られる.
  例： 店の商品数に
  比べ１顧客の
  購入数は
  極めて少ない.



・ZDDはBDDの改良技術として現在，世界的に広く使われている.

- 最近では，データマイニング分野に応用されて，
  画期的な有効性が示されている.
  （数百倍のデータ圧縮率・数十倍の処理高速化）
- 他にも応用例は増えつつある.

ZDDの応用

- 大規模な組合せアイテム集合データ

データベース解析（データマイニング・知識発見）
- ZDD上の集合演算を組合せることにより，
データ内に頻出するパタンを高速に抽出するアルゴリズム

- 人工知能システムにおける
知識インデックス表現
  - ロボットが自分で推論しながら次の動作を決めるような
複雑な演算処理を，ZDDで効率よく実現できる可能性がある.
  - 機械学習システムにおいて，学習状況をZDDを用いて
メモリ上に保存して，学習を高速化する.

- グラフ理論の様々な問題
  - 特に疎なグラフを表現する場合にZDDは効果的

演習問題

1. n 変数論理関数 f を表すBDDが与えられる.
   f を 1 にする解のうち，入力変数中の 1 の数が
   最も少なくなるようなものを一つ求めたい.
   これをBDDの節点数に対して線形時間で計算する方法を示せ.

2. 2^n 入力データセレクタ $f(x_1,\ldots x_n, y_1,\ldots, y_{2^n}) = y_{\sum_{i=1}^{n} x_i 2^{i-1}+1}$

   を表すBDDの節点数を最小/最大にする変数順序を1つ求めよ.
   また，そのときのBDDの内部節点数はそれぞれいくつか.

(発展)
   求めた節点数が最小/最大であることを証明せよ.

提出期限: 2017年7月19日(水) AoE
提出方法: ①Email: denzumi@mist.i.u-tokyo.ac.jp
(PDF or Wordファイル，件名は「演習第二Aレポート」)
②数理２研のポスト