

# 1 注意

今日の発表はいろいろなヒトの発表を借りて切り貼りしています

- 『フカシギの数え方』 おねえさんといっしょ！ みんなで数えてみよう！
- ZDD とフロンティア法 2017 年版 ver 0.1 奈良先端科学技術大学院大学 川原 純
- ZDD を用いたパスの列挙と索引生成 川原 純 (JST ERATO 研究員)
- ICAPS2012-Tutorial Decision Diagrams in Discrete and Continuous Planning (Scott Sanner)
- AAI2016-Tutorial Symbolic Methods for Probabilistic Inference, Optimization, and Decision-making (Scott Sanner)
- ICAPS2016-Tutorial Decision Diagrams for Discrete Optimization (John Hooker CMU)

# 2 [

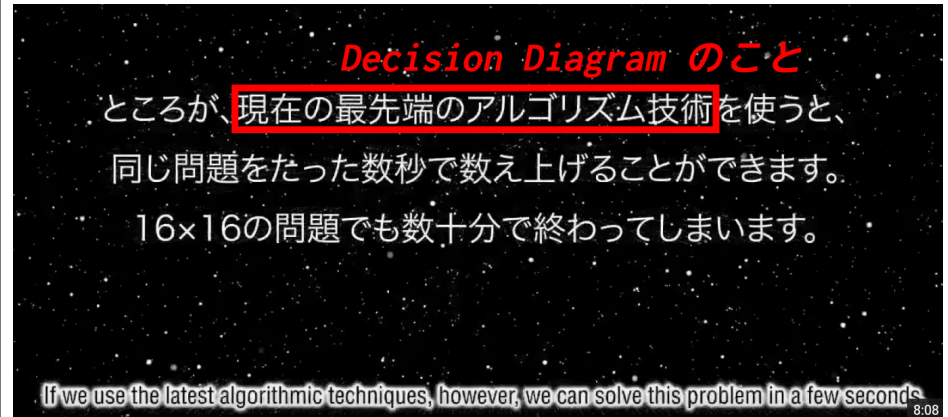
Decision Diagram とは]Decision Diagram とは

4MIN

『フカシギの数え方』 おねえさんといっしょ！ みんなで数えてみよう！」で紹介されないデータ構造

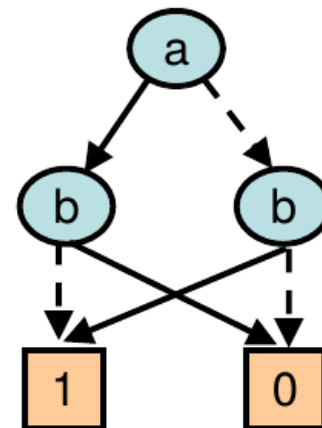
二倍速で見ましょう

# 3 今日のお話



- これを CL から扱うライブラリを紹介

# 4 Decision Diagrams (DDs)

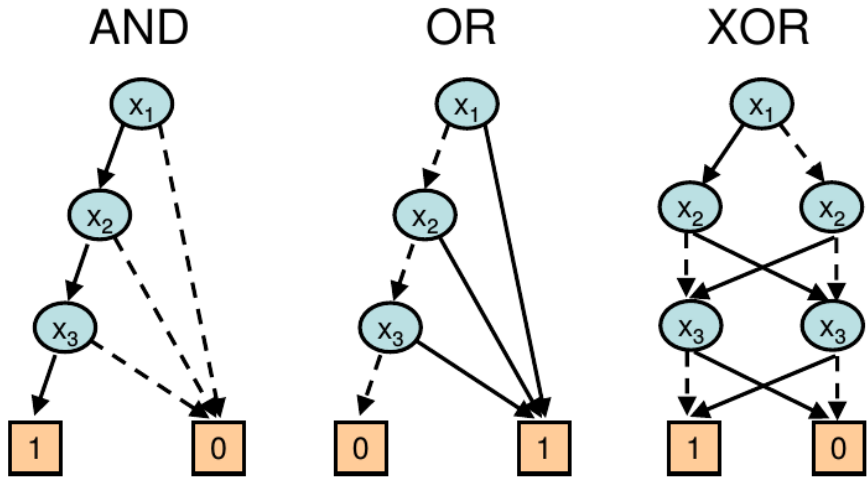


- 決定木 (Decision Tree) のグラフ版
- 関数をコンパクトに表現できる:
  - $B = \{0, 1\}$
  - $f : B^n \rightarrow B$  : BDD, ZDD
  - $f : B^n \rightarrow R$  も可能 (ADD)

### 4.1 XOR関数を線形サイズで保持できる

Tree では指数サイズのノードが必要。

AND と OR は DD でも Tree でも線形。



### 4.2 Boolean Function を表現してみる (真理値表)

$a$	$b$	$c$	$F(a, b, c)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

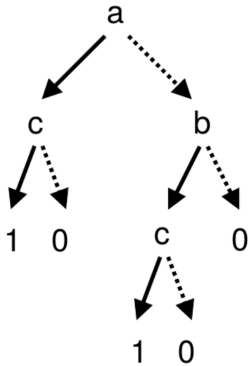
- 真理値表を使えば出来る
- 動くけど、もっとコンパクトに出来る

### 4.3 Boolean Function を表現してみる (木/Decision Tree)

ノードごとに True/False かで進む枝が決まる

表よりコンパクト

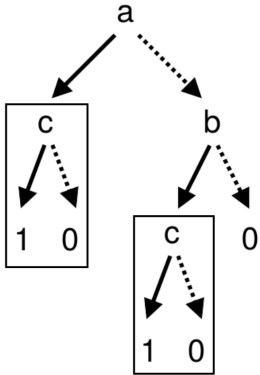
$a$	$b$	$c$	$F(a, b, c)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



### 4.4 Boolean Function を表現してみる (木/Decision Tree)

でもまだ無駄な重複がある

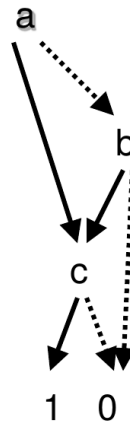
$a$	$b$	$c$	$F(a, b, c)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



## 4.5 Boolean Function を表現してみる (グラフ/Decision Diagram)

重複を共有してグラフにしよう!

$a$	$b$	$c$	$F(a, b, c)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



## 4.6 Decision Diagram 定義

node :: (index, then, else)

then 枝: index 番目の boolean 引数が true の時にたどる枝 (1-枝, true 枝)

else 枝: index 番目の boolean 引数が false の時にたどる枝 (0-枝, false 枝)

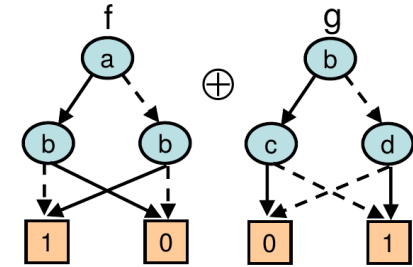
ハッシュテーブルでノードを管理

→同じ index と 子ノード を持つノードは1つしか存在しない (キャッシュされる)

かつ、グラフ上で常に index が降順で現れる (Ordered DD)

## 4.7 関数同士の演算を高速に行える

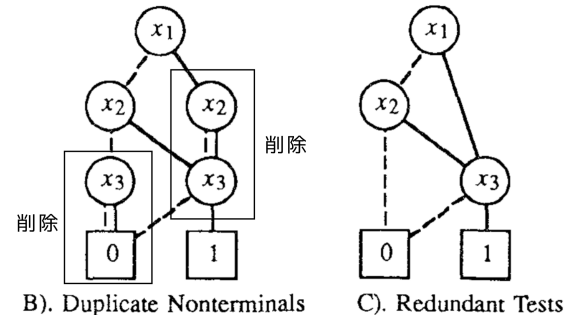
- 関数同士の演算 (代数系)
- BDD:  $\neg f, f \wedge g, f \vee g$
- ZDD:  $f \setminus g, f \cap g, f \cup g$
- ADD:  $-f, f \oplus g, f \otimes g, \max(f, g)$
- コンパクトなまま効率的に計算できる



## 4.8 縮約規則

DD に縮約規則をつけることでさらにコンパクトに出来る

1-枝と 0-枝が同じノード を削除 ← 出力に影響を与えない 無駄なノードだから



正式には **Ordered BDD (OBDD) == DD + 縮約規則 + 変数順序**

Ordered でない BDD を使うことはまれなので, BDD といえば普通 OBDD

## 4.9 BDD 同士の演算: Apply

2つの Boolean 関数  $f$  と  $g$  の論理和/論理積などをとることができる

$(f R g)(x) = f(x) R g(x)$ ,  $x = (x_0, x_1, \dots)$  のとき ( $R = \wedge, \vee, \dots$ ),

$$f R g = \text{BDD}(i, \quad f_{x_i=1} R g_{x_i=1}, \quad f_{x_i=0} R g_{x_i=0})$$

ただし  $i$  は  $f, g$  のルートノードの index

$f R g = \text{Apply}(f, g, R)$  と書くと、Apply は再帰的に定義可能。

```
(defun apply (f g op)
  (match* (f g)
    ((bdd index1      then1 else1)
     (bdd (= index1) then2 else2))
    (make-bdd :index index1
              :then (apply then1 then2 op)
              :else (apply else1 else2 op)))
    ... (index が違う場合など) ...
    ... (leaf node の場合など) ...
  ))
```

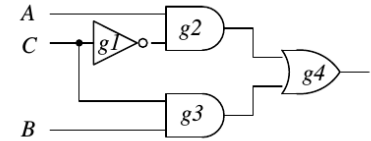
## 4.10 用途: 自動定理証明、回路の検証、自動プログラム検証 (Formal Methods, Verification)

指数的に多い要素を「並列に」操作できる

→ 全ケースを余すこと無く検査できる

注: ここでいう「並列」は、「多数の要素をまとめて処理」ぐらいの意味

並列計算機を走らせることは関連は無い(が、その意味の並列化も可能)



	Level-Based	Fanout-Based	
		C0	C1
A	0	0	0
B	0	0	0
C	0	1	1
g1	1	1	1
g2	2	0	1
g3	1	0	1
g4	3	0	1

Figure 4: Example circuit

## 5 BDD の問題点

BDD は論理関数を表現するのには良いが、ある種類の関数が得意でない

集合族:  $F = \{\{a, b\}, \{a, c\}, \{c\}\}$  — を BDD で表してみる

関数  $f(x_0, x_1, x_2)$ :

例:  $S = \{a, b\}$  は  $F$  に含まれているか?

集合  $S$  が 集合族  $F$  に含まれれば  $f = 1$ , 含まれなければ  $f = 0$

引数  $x_0, x_1, x_2$ : 各要素  $a, b, c$  が  $S$  に入力に含まれていれば 1, 含まれていなければ 0.

$f(1, 1, 0) = 1$ , 従って  $S \in F$

## 5.1 実際にやってみると...

左はいまいち小さくならない。

- これを改良したのが右の **ZDD**

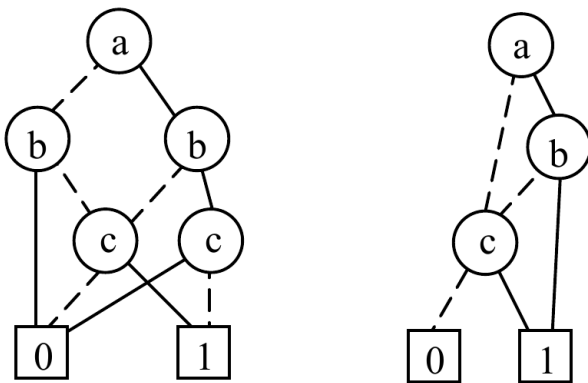
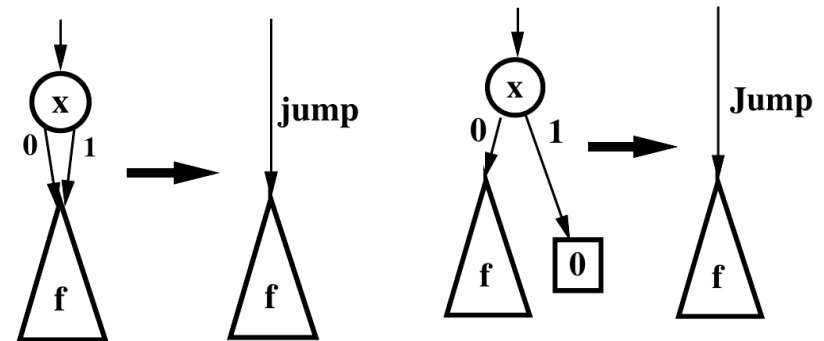


Figure 2. The BDD and the ZDD for the set of subsets  $\{\{a,b\}, \{a,c\}, \{c\}\}$ .

## 5.2 別の縮約規則: Zero-suppressed Decision Diagram (ZDD)

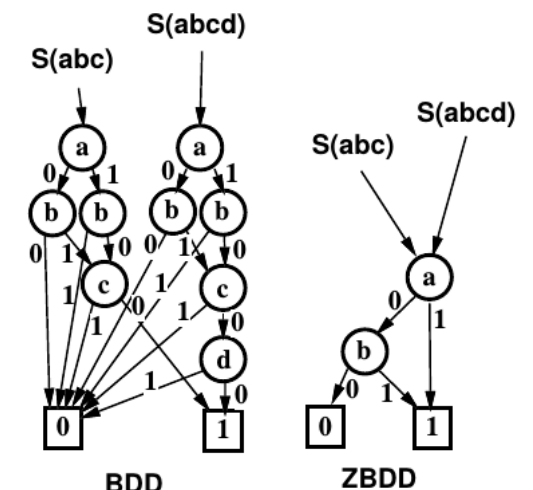
BDD: 同じなら削除

ZDD: 1-枝が0なら削除



## 5.3 同じ関数でも zdd のほうが小さいのは...

$S(abc):$		$S(abcd):$	
abc	S	abcd	S
000	0	0000	0
100	1	1000	1
010	1	0100	1
110	0	1100	0
001	0	0010	0
101	0	1010	0
011	0	0110	0
111	0	1110	0
		0001	0
		1001	0
		0101	0
		1101	0
		0011	0
		1011	0
		0111	0
		1111	0



- 理由: 殆どの場合で関数の値が **0** だから

- $\rightarrow 0$  と  $1$  の割合が同じぐらいの時は bdd,  $0$  が多い場合には zdd が良い

## 6 ZDD は使える

$0$  が多い場合には zdd が良い

- 最悪指数時間の問題を動的計画法で解く場合…
  - 空間全体のうち使う空間はほんの少しなので…
  - 保持すべきデータを zdd に貯めれば殆ど  $0$
  - **BDD** より **ZDD** が速いはず!
  - 指数的に速いアルゴリズムを書くのに役立つはず  $\neq$  定数倍の高速化

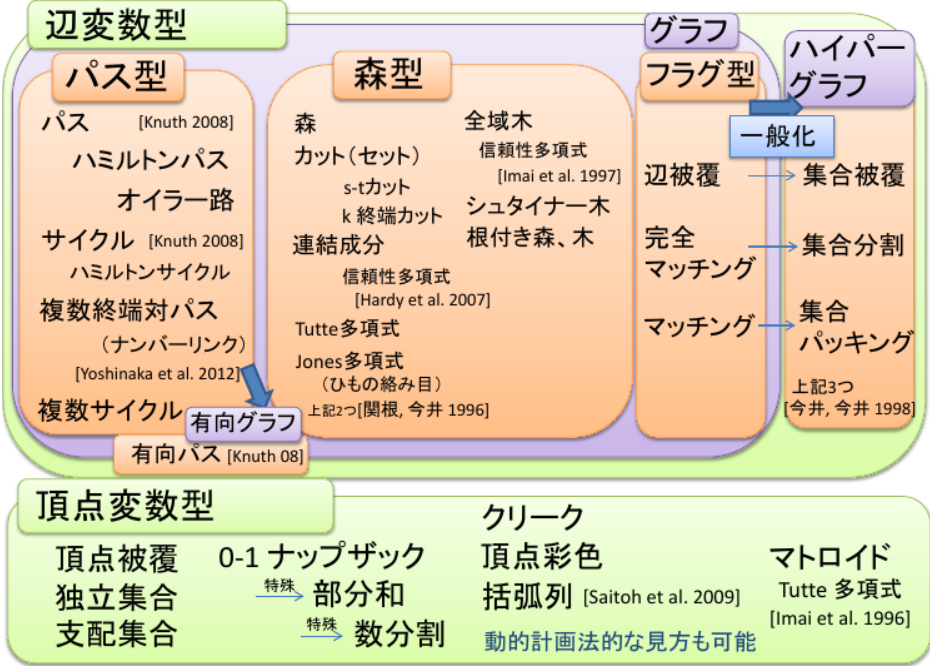
### 6.1 例えば... (ERATO のスライドを借用)

#### 様々なグラフ構造

- パス以外にも様々なグラフ構造に対して ZDD 構築可能
  - ハミルトンパス
  - 複数終端対パス
  - 森、全域木
  - マッチング
  - 集合被覆
  - グラフ分割
  - etc.

川原 純, 湊 真一,  
"組合せ問題の解を列挙索引化する ZDD 構築アルゴリズムの汎用化,"  
電子情報通信学会コンピュータ研究会, 信学技報, vol. 112, No. 93,  
COMP2012-12, pp. 1-7, June 2012.

6.2 例えば... (ERATO のスライドを借用)



6.3 例えば... (ERATO のスライドを借用)

ZDD に関するその他の研究成果

- グラフ彩色問題 [Morrison et al. 2016]
- 多次元ナップザック問題 [安田ら 2016]
- 制約を追加したナップザック問題 [Nishino et al. 2015]
- Web の影響拡散の厳密計算 [Maehara et al. 2017]
- 自然言語処理における最適化 [西野ら 2015]
- 系統樹復元問題に対する列挙アルゴリズム [Kiyomi et al. 2012]

## 6.4 例えば... (ERATO のスライドを借用)

### ZDD に関するその他の研究成果

- 最長路問題の求解 [Kawahara et al. 2016]
- 頂点故障も考慮した信頼性評価 [園田ら 2016]
- 避難計画作成 [Takizawa et al. 2013]
- AND/OR 演算に関する計算量の証明 [Yoshinaka et al. 2012]
- 簡潔データ構造による ZDD のインデックス作成 [Denzumi et al. 2014]
- プリミティブソーティングネットワークの数え上げ [Kawahara et al. 2011]

## 6.5 ZDD と BDD では扱う Apply 操作が違う

BDD: 論理操作 —  $\neg f, f \wedge g, f \vee g$

ZDD: 集合操作 —  $f \setminus g, f \cap g, f \cup g$

互換性はそこまでではない アルゴリズムも別に作らないといけない

## 7 本題: BDD/ZDD を Lisp から使うためのライブラリ

BDD は沢山のライブラリがある



### 7.1 そのうち、ZDD に対応するのはあまり多くない

- CUDD: 大御所, C のライブラリ
- SapporoBDD, TdZdd (Graphillion の内部): ERATO が作った C/Python ライブラリ,
- ほかに数件だけ、つまり ZDD はあまりまだ注目されていない
- CUDD への CFFI バインディング: CL-CUDD が存在
- → zdd 対応なし, CLOS(遅い), Quicklisp 登録なし, テスト無し, Tutorial 無し, CUDD を自分でビルドしてインストールする必要あり, 最新版の CUDD に対応せず。
- 自分がやったこと: ココらへんを整備して、これで何かを作れ



ることを実証すること。

## 8 Mate-ZDD

おねえさんのパス数え上げ問題を解くためのアルゴリズム

Don Knuth の Simpath アルゴリズムをピュア ZDD で書き直した、シンプルだが遅いバージョン

これを CL-CUDD を使って実装した

<https://github.com/guicho271828/simpath>

デモ

## 9 Future Work

自分の専門である 自動行動計画ソルバ(プランナ)を ZDD で作る

- 配列ベースの普通の手法に比した利点:
  - よりスケールする (大きな問題が解ける)
    - \* メモリ使用量が削減できる
    - \* 探索の情報を圧縮して保持できるから
  - 細かなループの中のチューニングを気にしなくて良い

川原 et al. 数理解析研究所講究録 第 1744 巻 2011 年 35-41 ZDD によるパスの  
列挙

\* ZDD が複数の状態をまとめて処理するから

### 9.1 BDD ベースのプランニングアルゴリズムは存在する:

**procedure** *BDDA\**

$Open(f, x) \leftarrow h(f, x) \wedge \Phi_{S0}(x)$

**while** ( $Open \neq \emptyset$ )

$(f_{\min}, Min(x), Open'(f, x)) \leftarrow goLeft(Open)$

**if** ( $\exists x (Min(x) \wedge \Phi_G(x))$ ) **return**  $f_{\min}$

$VarTrans_{x,x'}(Min(x))$

$Open''(f, x) \leftarrow \exists x' Min(x') \wedge T(x', x) \wedge$

$\exists e' h(e', x') \wedge \exists e h(e, x) \wedge (f = f_{\min} + e - e' + 1)$

$Open(f, x) \leftarrow Open'(f, x) \vee Open''(f, x)$

**Table 2.** The  $A^*$  algorithm using *OBDDs*.

BDD より ZDD のほうがよいはず

← 指数爆発している全空間に比べ、実際に使われる空間サイズは小さいから

## 9.2 International Planning Competition 2014 で優勝

コレに勝つ!

### Sequential Optimal track: Results

17 planners submitted. Showing the top FIVE

SymBA*-2	151/280	1st
SymBA*-1	143/280	1st
cGamer	120/280	2nd
SPM&S	114/280	3rd
RIDA	113/280	4th
Dynamic-Gamer	99/280	5th

## 9.3 強い

### BDDs Strike Back (in AI Planning)

**Stefan Edelkamp**  
Institute of Artificial Intelligence  
University of Bremen  
edelkamp@tzi.de

**Peter Kissmann and Álvaro Torralba**  
Foundations of Artificial Intelligence  
Saarland University, Saarbrücken  
{kissmann,torralba}@cs.uni-saarland.de

### Symbolic Pattern Databases in Heuristic Search Planning

#### Stefan Edelkamp

Institut für Informatik,  
Albert-Ludwigs-Universität,  
Georges-Köhler-Allee, D-79110 Freiburg  
eMail: edelkamp@informatik.uni-freiburg.de

## 10 まとめ

BDD, ZDD を概説

CL-CUDD をまともに使える状態にした

ZDD でお姉さん問題を解くソルバを作った

プランニングに応用したい