

BDD/ZDDを基盤とする離散構造処理系の技法と最近の展開

2011年9月28日

湊 真一

北海道大学情報科学研究科 / JST ERATO

本講演の内容

- 離散構造の処理とは
- 論理関数と組合せ集合の演算処理系
 - BDD: 論理関数
 - ZDD: 組合せ集合
- 種々の離散構造と演算処理系
 - ZDD-V: 組合せ頻度表
 - SeqBDD: 系列集合
- ERATO湊離散構造処理系プロジェクト
 - プロジェクトの概要
 - 最近の研究(1) ZDDによる超高速パス列挙
 - 最近の研究(2) π DD: 順列集合の演算処理系



背景: 離散構造とその処理

■ 離散構造とは

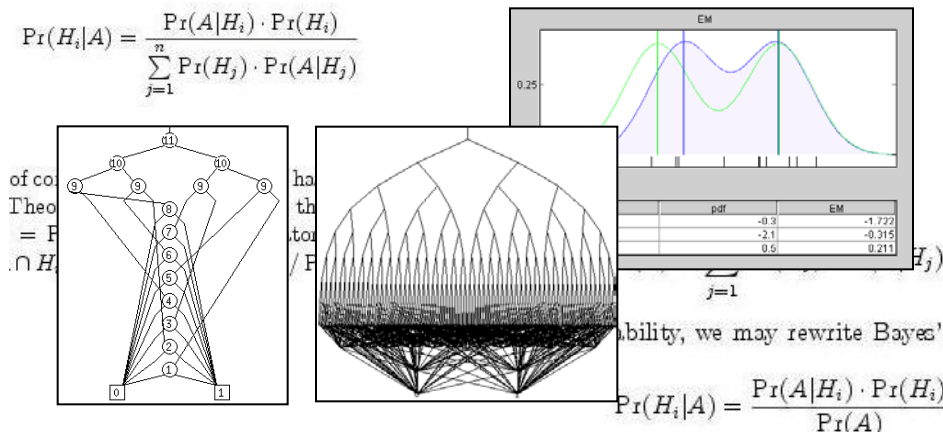
- 離散数学および計算機科学の基礎をなす数学的構造
- 集合理論、記号論理、帰納的証明、グラフ理論、組合せ論、確率論などを含む

■ およそ計算機が扱うあらゆる問題は、単純な基本演算を要素とする離散構造の処理に帰着される。

- 最終的には、膨大な数の場合分け処理を要することが多い。

■ 離散構造の処理は、計算機の様々な応用分野に共通する基盤技術

- 典型的な応用分野: システム設計自動化、大規模システム故障解析、制約充足問題、データマイニングと知識発見、機械学習と自動分類、生命情報科学、web情報解析、etc.
- 現代情報化社会に対する波及効果は極めて大。



本研究の基本構想

- 様々な工学的応用を持つ基盤技術として
「離散構造処理系」に着目し、研究開発を行う

システム設計自動化

データマイニングと知識発見

大規模システム故障解析

機械学習と自動分類

制約充足問題

生命情報科学

web情報解析

工学的応用

→ 社会への
影響大

離散構造処理系

本研究の
対象領域

性能向上
(10倍～
100倍以上)

集合論

記号論理

帰納的証明

組合せ論

グラフ理論

確率論

数学的
概念構造

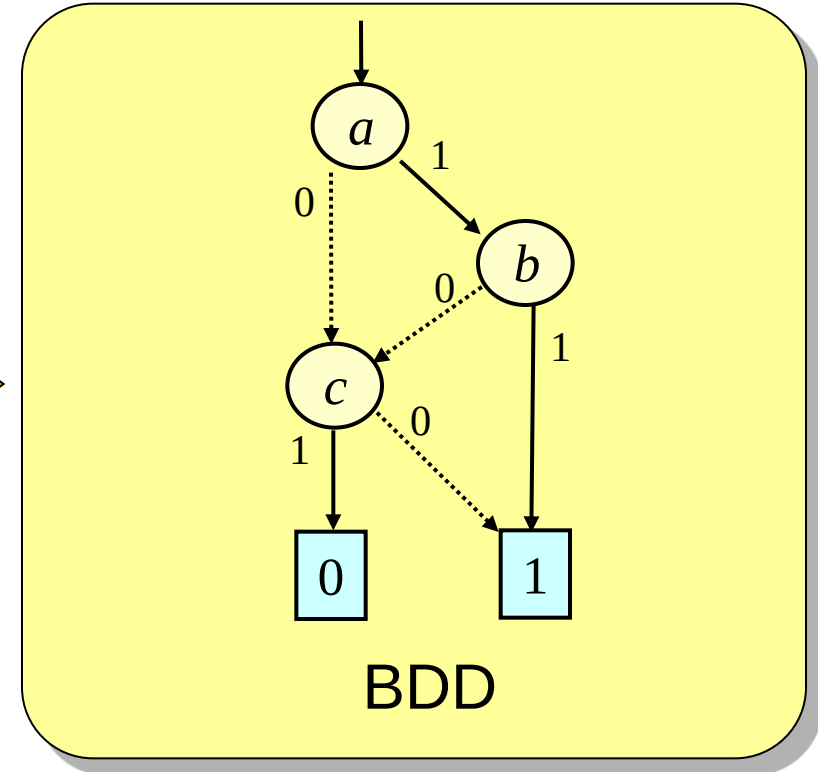
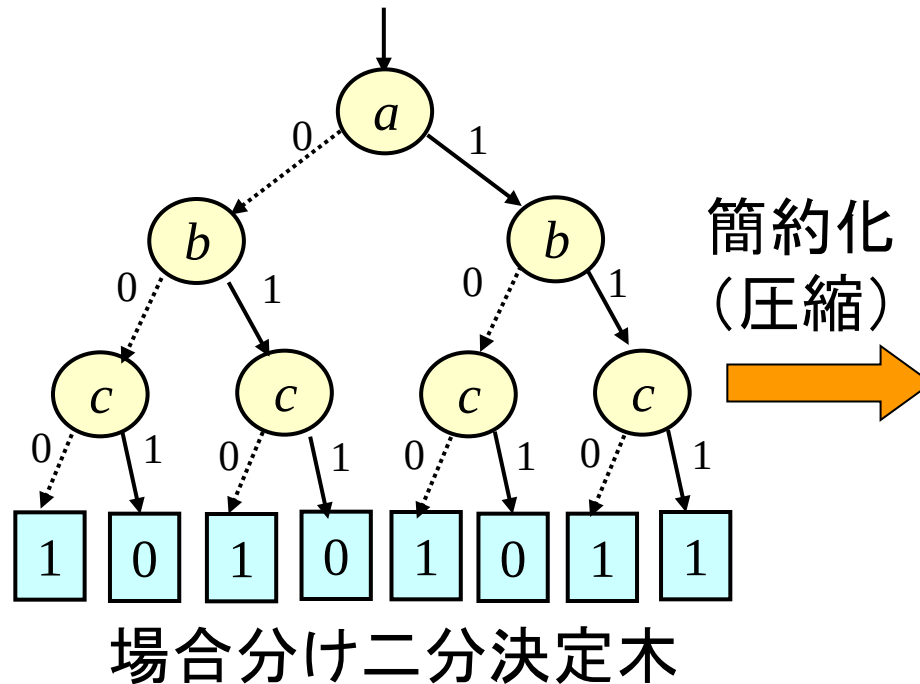
本講演の内容

- 離散構造の処理とは
- 論理関数と組合せ集合の演算処理系
 - BDD: 論理関数
 - ZDD: 組合せ集合
- 種々の離散構造と演算処理系
 - ZDD-V: 組合せ頻度表
 - SeqBDD: 系列集合
- ERATO湊離散構造処理系プロジェクト
 - プロジェクトの概要
 - 最近の研究(1) ZDDによる超高速パス列挙
 - 最近の研究(2) π DD: 順列集合の演算処理系



BDD (Binary Decision Diagram: 二分決定グラフ)

離散構造の最も基本的なモデルである「**論理関数**」の処理技法



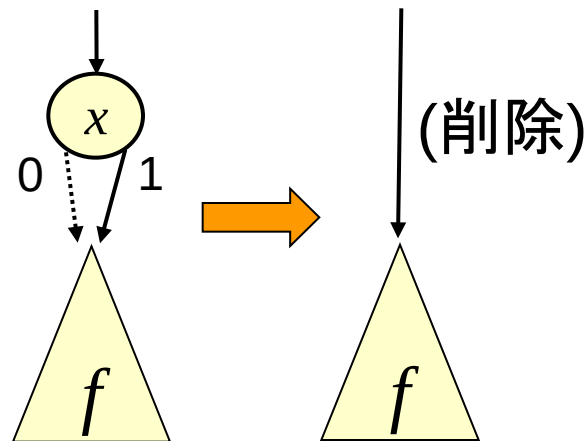
- 場合分け二分木グラフを簡約化(データ圧縮)
- 多くの実用的な論理データをコンパクトかつ一意に表現。
(数十～数百倍以上の圧縮率が得られる例も)



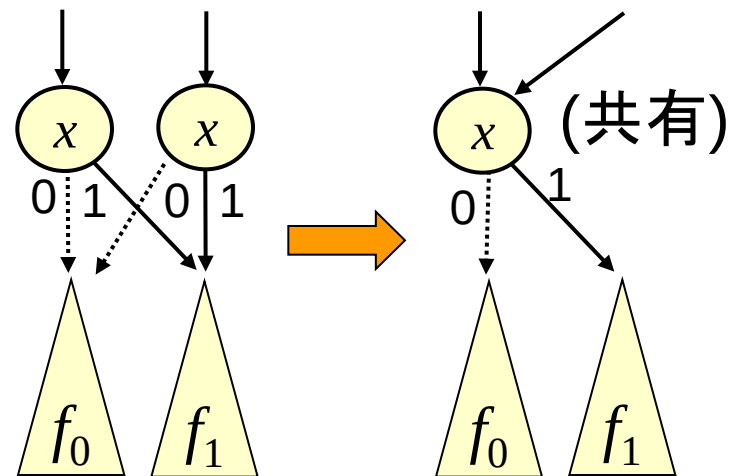
ROBDD (Reduced Ordered BDD)

- 同じ論理を表すBDDは一般には複数存在する
- 順序付きBDD:
 - 変数に**全順序関係**が定義されている
 - 根(root)から定数節点に至るすべてのパスについて変数の出現順序が、全順序関係に矛盾しない
- 既約なBDD
 - BDDの2つの簡約化規則がこれ以上適用できなくなるまで適用されている形(→ **論理関数に対して一意な表現となる**)
- 重要な性質を持つのは既約な順序付きBDD(ROBDD)
 - **以後、特に断らない限り、ROBDDのことを単にBDDと呼ぶ。**

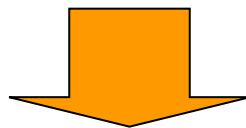
BDD の簡約化規則



冗長な節点を削除



等価な節点を共有

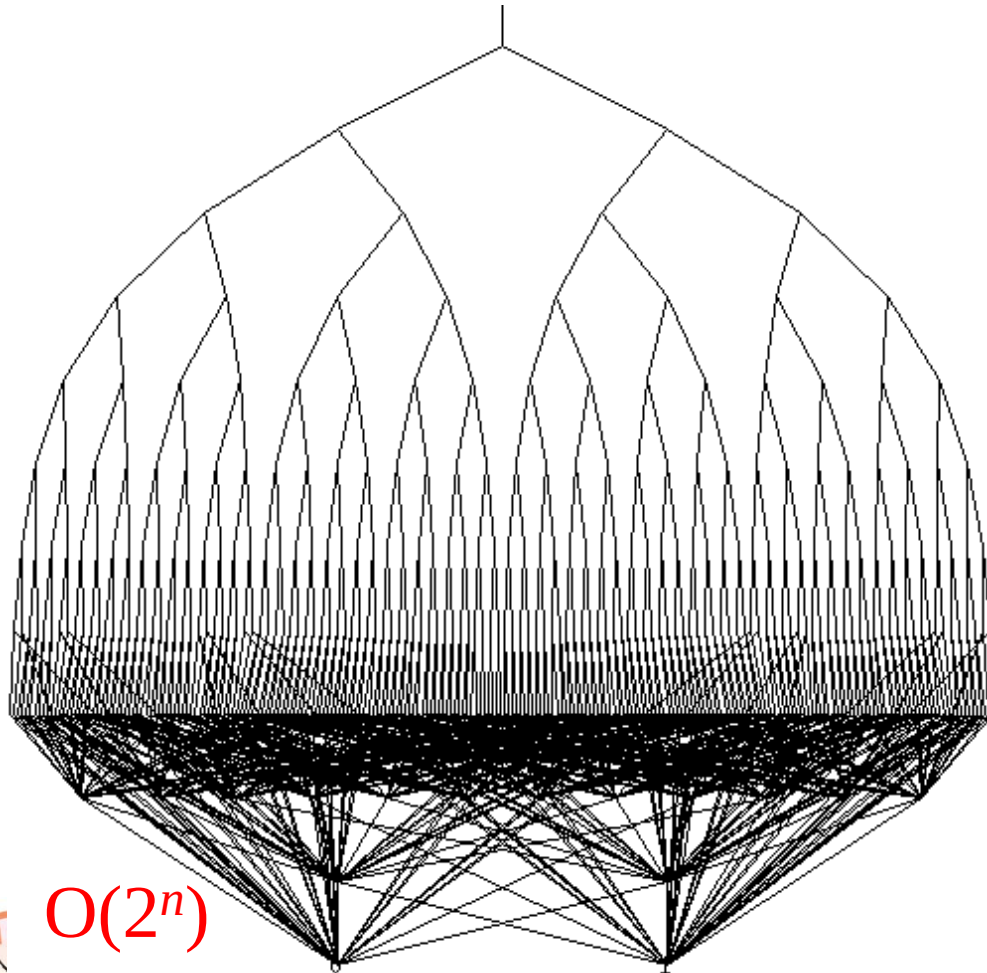


既約なBDDが得られる。

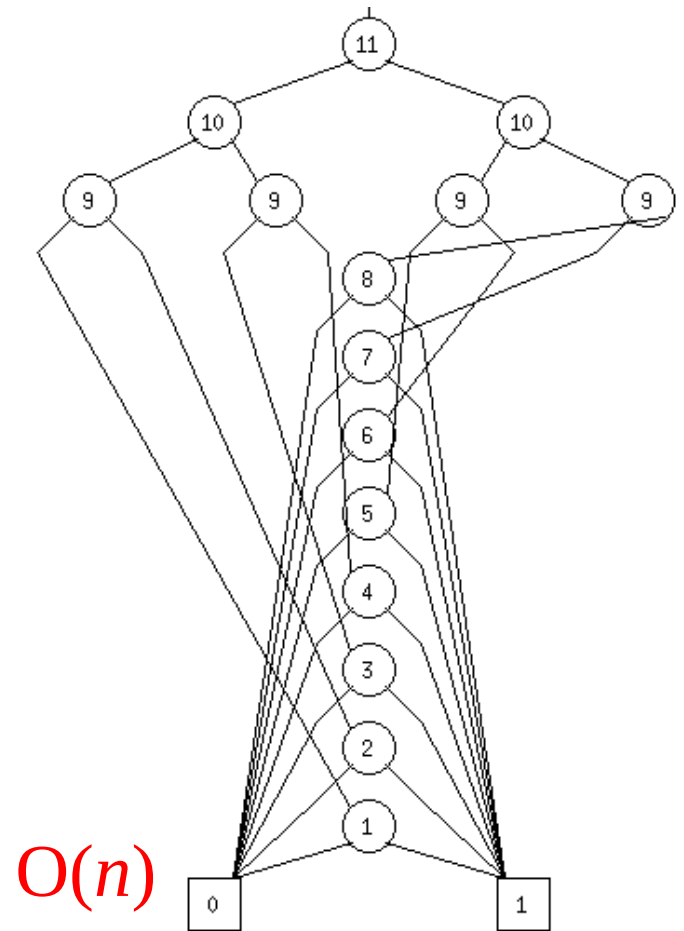
変数の展開順序を固定すれば論理関数を一意に表現できる

BDD簡約化の効果

- 特定の問題では、指数関数的な圧縮効果が得られる。
 - 例題に依存するが、多くの実用的な問題で効果がある。



$O(2^n)$



$O(n)$

BDDの節点数(理論的上限)

- n 入力の論理関数は 2^{2^n} 種類存在する。
これを識別するには少なくとも 2^n ビットの記憶量が必要。
 - 固定長データでは真理値表が最小
- BDDの場合、 n 変数論理関数を表現するための最悪の場合の節点数は $O(2^n/n)$ となる。
 - 1個の節点を記憶するのに $O(n)$ ビット必要なので、全体としては $O(2^n)$ ビット必要。
- ただし、実用上よく用いられる多くの論理関数について、 n の線形や多項式の節点数になることが知られている。
 - 可変長データ(圧縮データ構造)

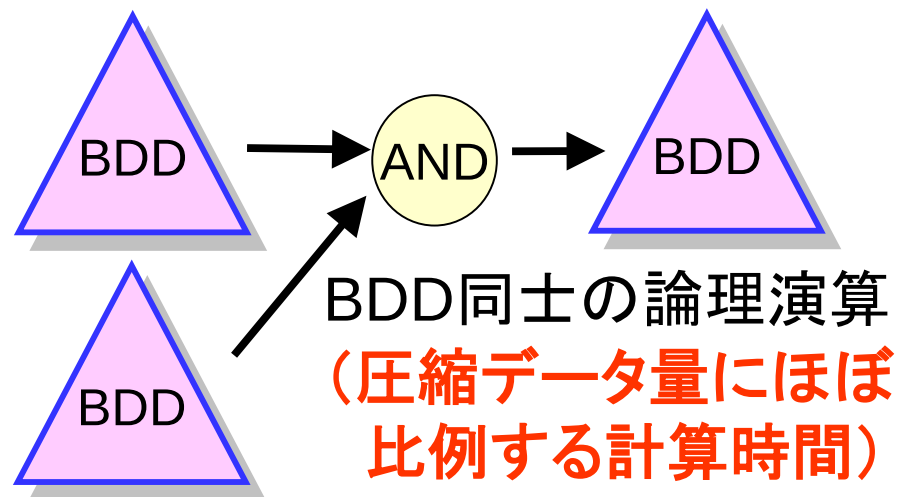
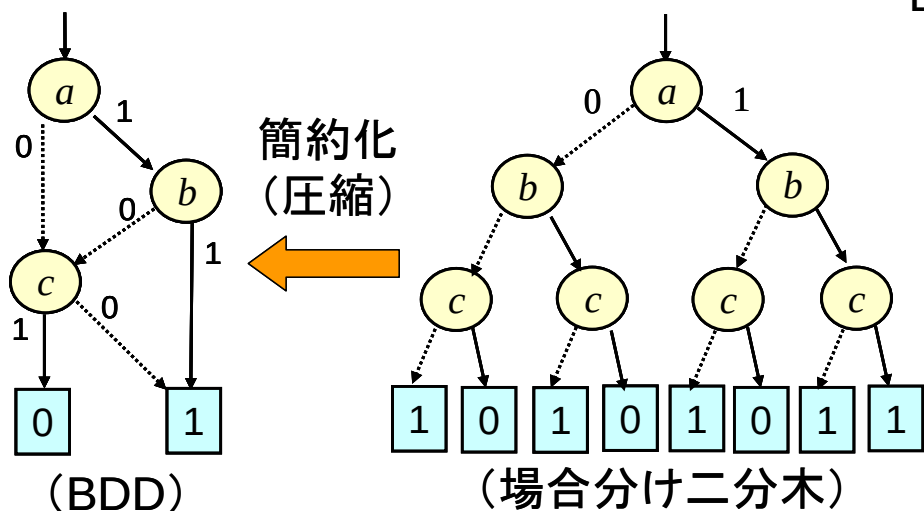
論理演算によるBDDの生成法

- 場合分け二分木を圧縮してBDDを作る方法では、常に指数オーダーの記憶量と処理時間がかかってしまう。

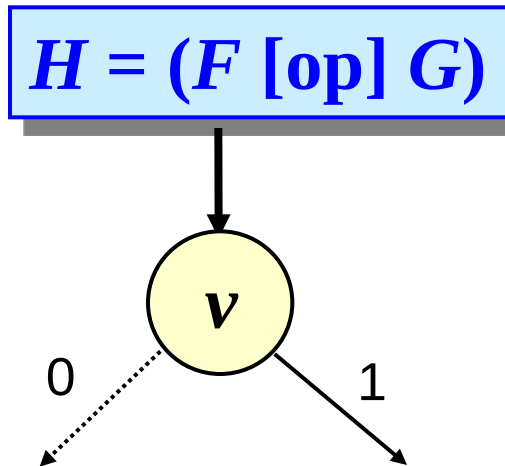


Bryant (CMU)

1986年に画期的なBDD演算アルゴリズムを提案。以後急速にBDD技術が発達。
(長期間、情報科学の全分野での最多引用文献となった)



二項論理演算アルゴリズム

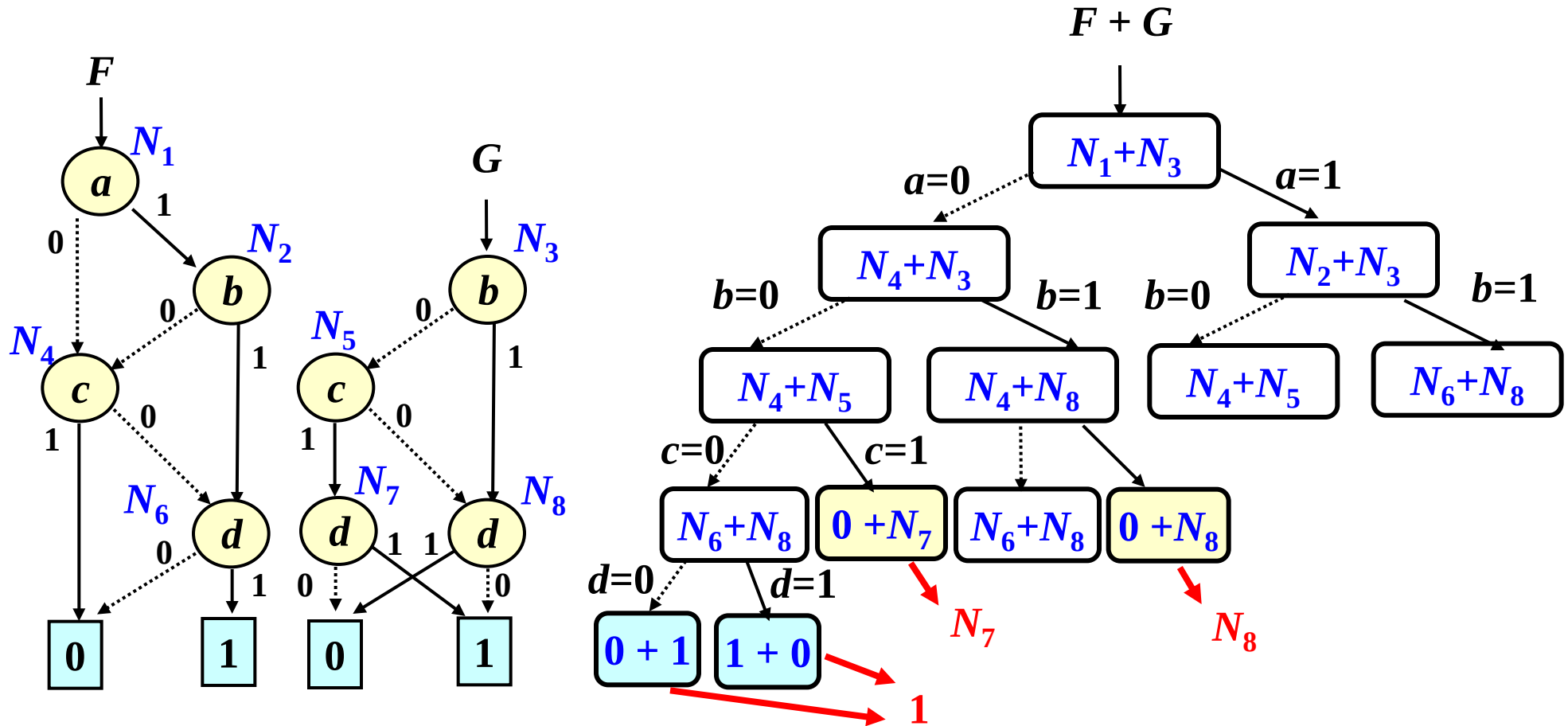


$$H_{(v=0)} = (F_{(v=0)} \text{ [op] } G_{(v=0)})$$

$$H_{(v=1)} = (F_{(v=1)} \text{ [op] } G_{(v=1)})$$

- ある変数 v 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演算の場合) $F+1=1$, $F+0=F$, $F+F=F$...
- 各演算結果をBDDに組み上げる

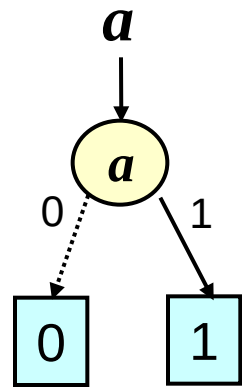
二項論理演算の実行例



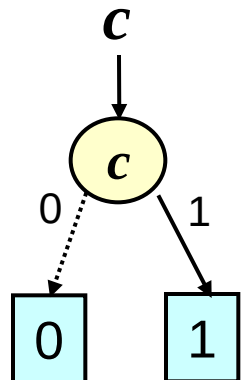
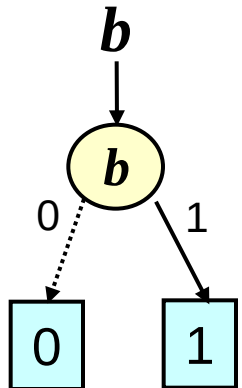
演算キャッシュ ($F [op] G \rightarrow H$) と呼ぶテーブルに途中結果を保存し、冗長な再帰呼び出しを防ぐ

論理式からのBDD生成

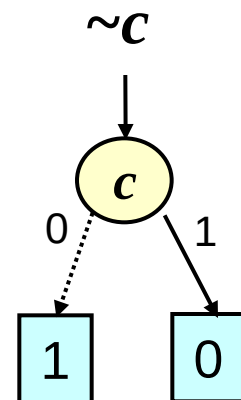
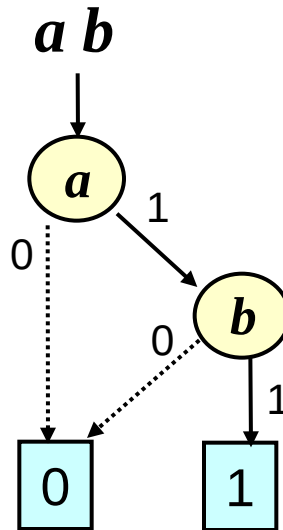
- 2つのBDDの間の二項論理演算を繰り返して任意のBDDを生成



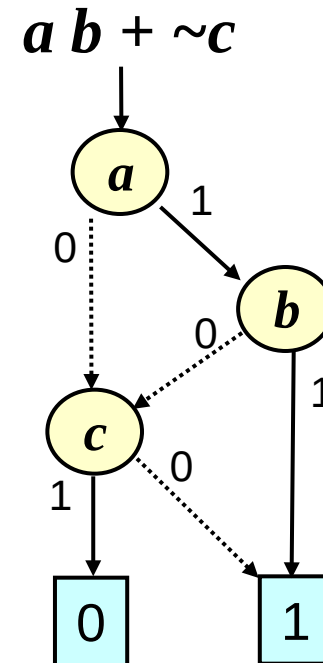
AND
演算



NOT
演算

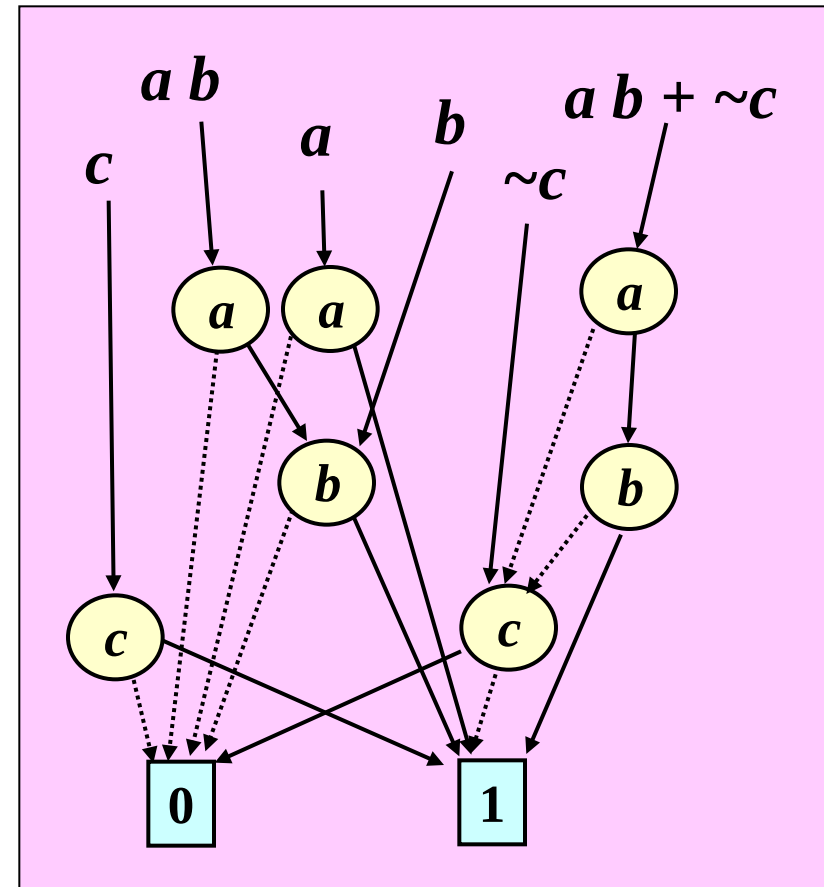
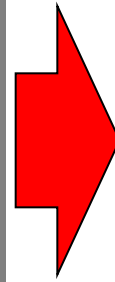
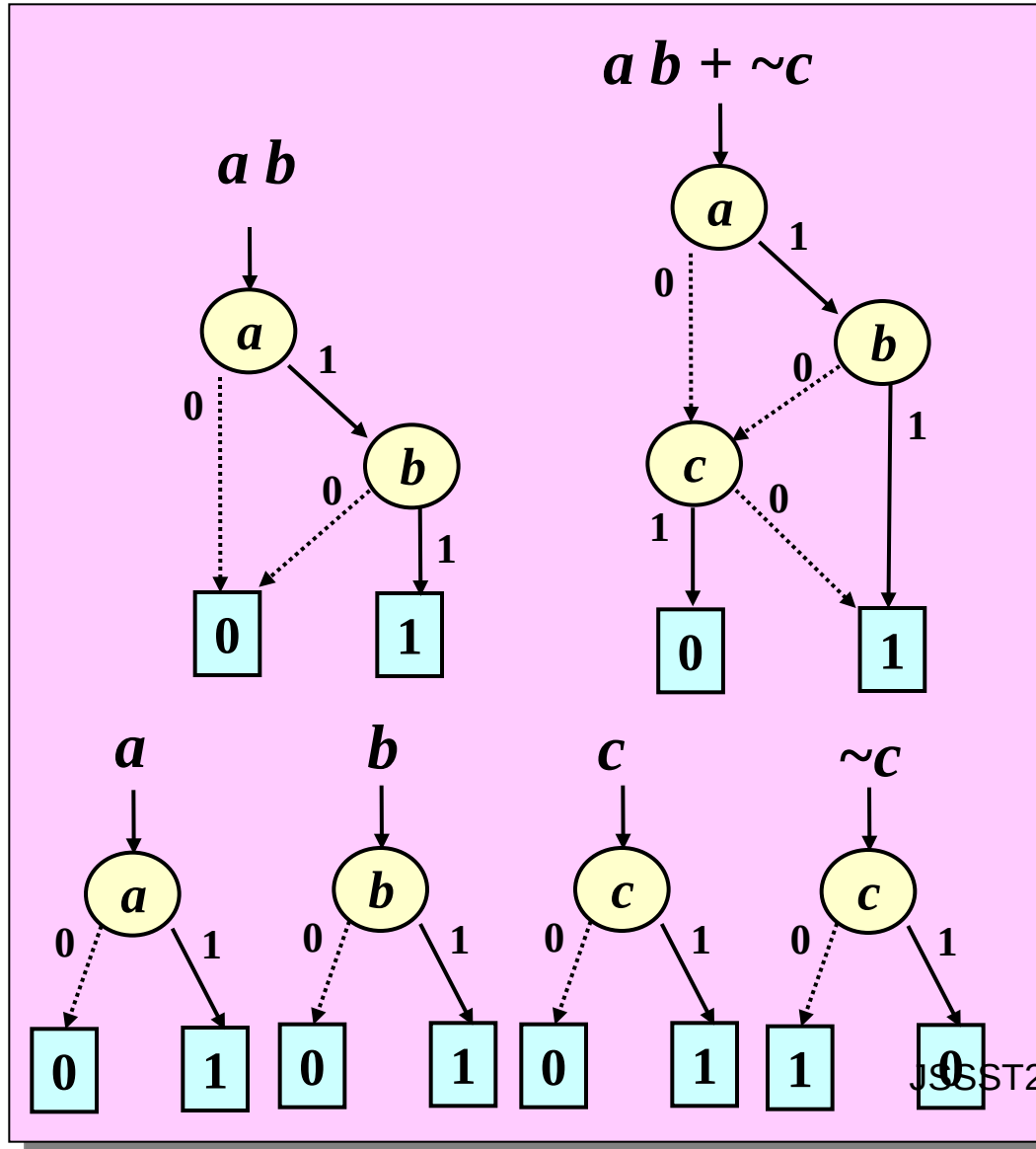


OR
演算

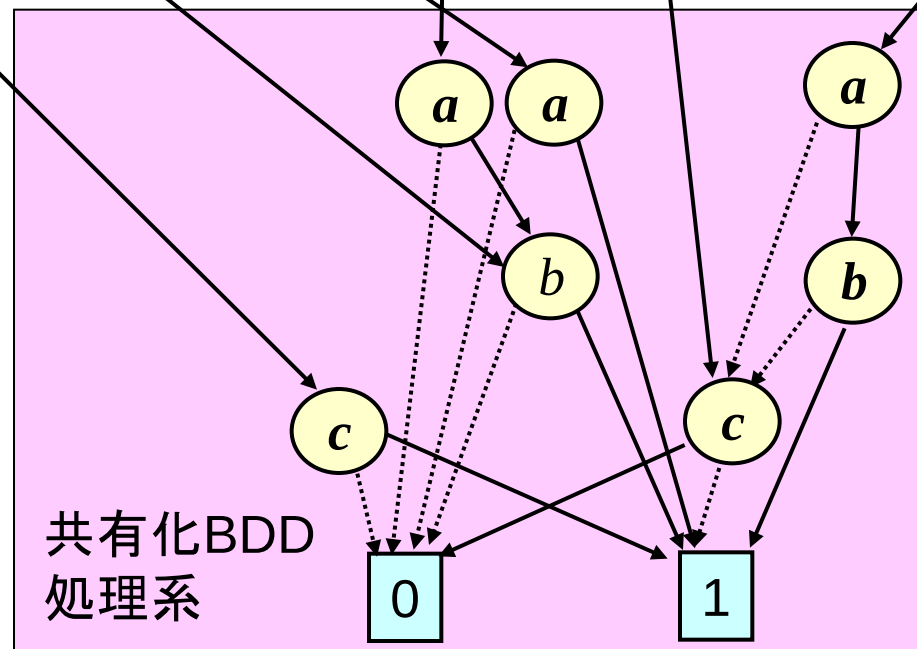
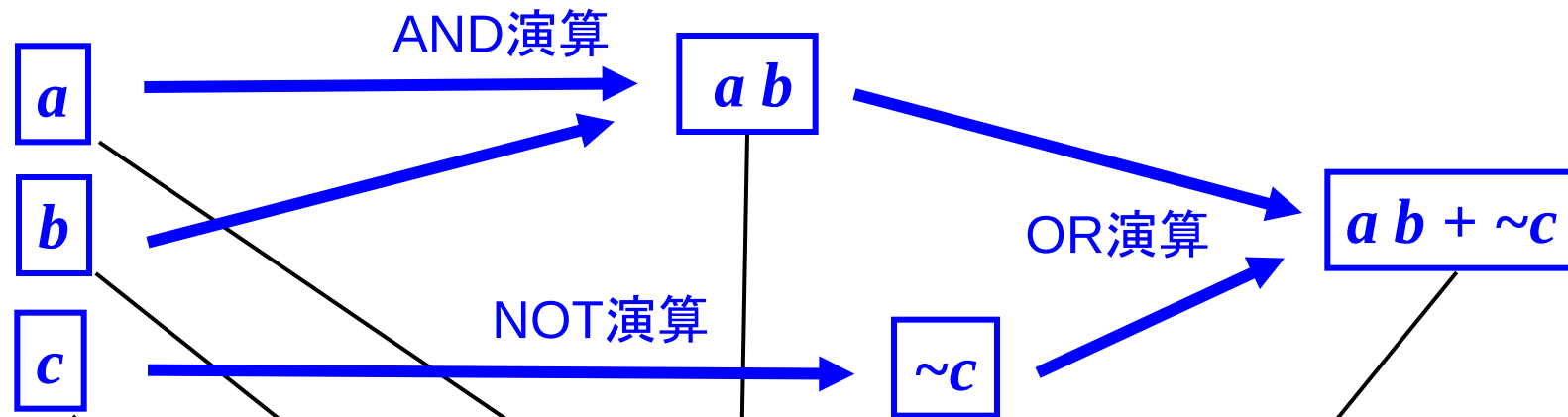


複数のBDDの共有化(Shared BDD)

- 変数の順序を揃える
- 全てのBDDを1つのグラフに共有化



共有化BDDでの論理演算処理



- 論理演算の実行に従い、共有化BDDが成長していく
- 共有化BDD処理系では、論理の一致判定はポインタの一致判定のみで行える(定数時間)

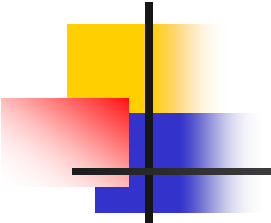
BDDパッケージ

- BDD処理系は世界各地の研究機関で1990年頃から盛んに開発された。
 - BDDパッケージとして無料配布されている物もいくつかある。
- 多くの場合、CまたはC++のライブラリとして提供されている。
 - BDDへのポインタを引数としてライブラリ関数を呼び出すと、メモリ上にBDDが生成され、新しいBDDへのポインタが返り値として戻ってくる。
 - ユーザはBDDの論理演算を呼び出すメインプログラムを書き、BDDパッケージをリンクしてコンパイルすると、BDD利用アプリケーションを作ることができる。
- 扱える最大節点数はPCの主記憶に依存する。
 - 主記憶2GBのマシンで約4000万節点。500GBマシンで約100億
 - メモリからあふれると異常終了する。計算結果は得られない。

BDDパッケージ

- BDD処理系は世界各地の研究機関で1990年頃から盛んに開発された。
 - BDDパッケージとして無料配布され、多くの場合、CまたはC++のライブラリとして提供される。
- 多くの場合、CまたはC++のライブラリとして提供される。
 - **BDDへのポインタを引数としてライブラリを呼び出すと、メモリ上にBDDが生成され、新しいBDDのポインタが返り値として戻ってくる。**
 - ユーザはBDDの論理演算を呼び出すメインプログラムを書き、BDDパッケージをリンクしてコンパイルすると、BDD利用アプリケーションを作ることができる。
- 扱える最大節点数はPCの主記憶に依存する。
 - 主記憶2GBのマシンで約4000万節点。500GBマシンで約100億
 - メモリからあふれると異常終了する。計算結果は得られない。

```
symbol x1, x2, x3, x4
F = bddand(x1, x2);
G = bddor(x3, x4);
H = bddxor(F, G);
bddprint(H);
```



BDDの応用分野の移り変わり

- BDDは、VLSI設計自動化の分野で考案され、当初は主にハードウェア設計分野で発展してきた。(1990～2000頃)
 - 論理回路の等価性判定問題
 - 形式的設計検証 / モデル検査
 - 論理回路合成 / 最適化
 - テストパターン生成
- 2005年以降、設計分野以外にも応用範囲が広がってきた。
(BDD → ZDDの利用が増加)
 - データマイニング(頻出パターンマイニング等)
[Minato2005,2008][Loekito,Bailey2006]
 - ベイジアンネットワークの確率推論計算の高速化
[Minato2007]

論理関数と組合せ集合

a	b	c	F
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	0
1	1	1	0

論理関数:

$$F = (a \wedge b \wedge \neg c) \vee (\neg b \wedge c)$$

組合せ集合:

$$F = \{ab, ac, c\}$$



(買い物客の購入品)

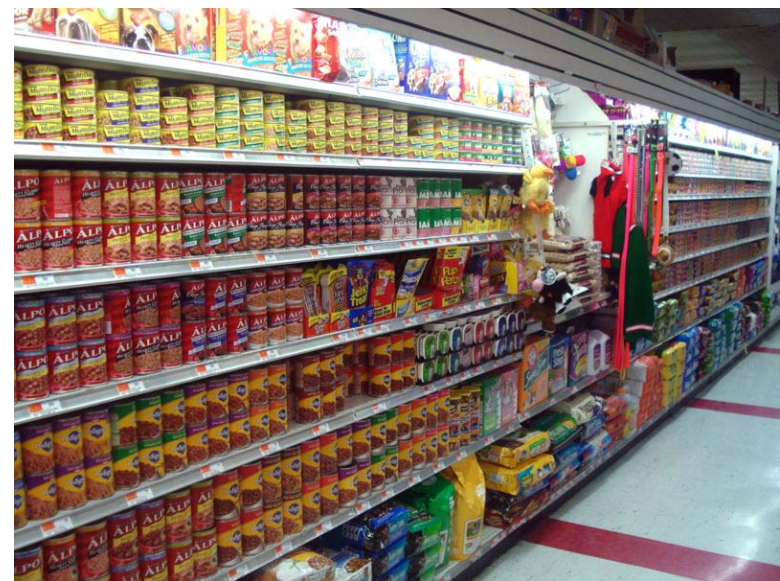
→ ab

→ c

→ ac

■ 組合せ集合と論理関数の演算は対応関係がある。

- Union of sets → logical OR
- Intersection of sets → logical AND
- Complement set → logical NOT



ZDD(ゼロサプレス型BDD)による集合の表現

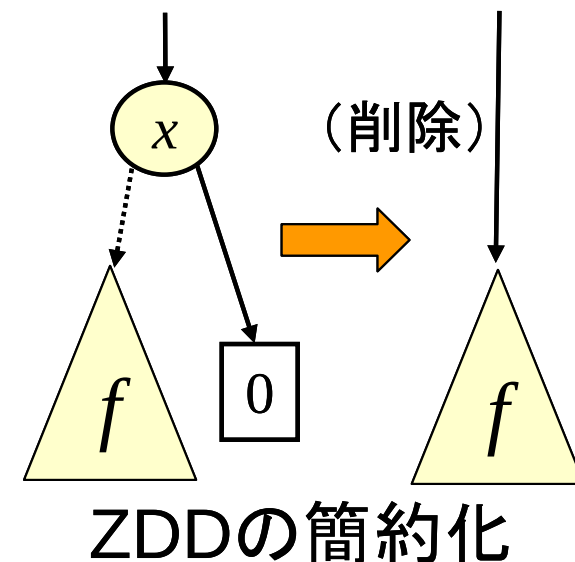
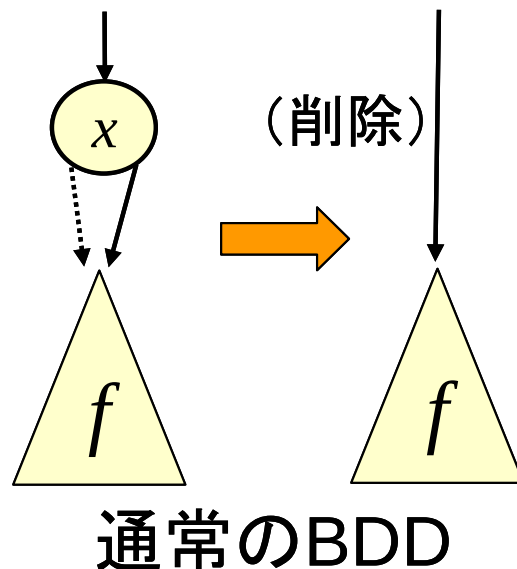
■ 「組合せ集合」を効率的に表現するためのBDDの改良

- 湊が世界で初めて考案し命名(1993年)

- 通常と異なる簡約化規則を考案。

- 疎な集合の族を扱う場合に著しい効果が得られる。

(例: 商店の陳列アイテム数に比べて1顧客の購入点数は極めて少ない。)



■ 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。



頻出アイテム集合抽出問題

■ データマイニングの最も基本的な問題

- 最小出現頻度 α 以上のレコードに含まれるアイテム組合せの部分集合を抽出・列挙する問題

レコード番号	アイテム集合
1	a b c
2	a b
3	a b c
4	b c
5	a b
6	a b c
7	c
8	a b c
9	a b c
10	a b
11	b c

最小頻度 $\alpha = 10$ → { b }

最小頻度 $\alpha = 8$ → { ab, a, b, c }

最小頻度 $\alpha = 7$ → { ab, bc, a, b, c }

最小頻度 $\alpha = 5$ → { abc, ab, bc, ac, a, b, c }

最小頻度 $\alpha = 1$ → { abc, ab, bc, ac, a, b, c }



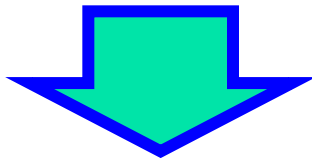
“LCM over ZDDs” [Minato et al. 2008]

- **LCM:** [Uno2003]

Output-linear time algorithm of frequent itemset mining.

- **ZDD:** [Minato93]

A compact graph-based representation for large-scale sets of combinations.



**Combination of
the two techniques**

Generates large-scale frequent itemsets on the main memory, with a very small overhead from the original LCM.

(→ **Sub-linear time and space** to the number of solutions when ZDD compression works well.)

LCM over ZDDs: An example

- The results of frequent itemsets are obtained as ZDDs on the main memory. (not generating a file.)

Record ID	Tuple
1	<i>a b c</i>
2	<i>a b</i>
3	<i>a b c</i>
4	<i>b c</i>
5	<i>a b</i>
6	<i>a b c</i>
7	<i>c</i>
8	<i>a b c</i>
9	<i>a b c</i>
10	<i>a b</i>
11	<i>b c</i>

LCM over ZDDs
Freq. thres. $\alpha = 7$
 $\{ ab, bc, a, b, c \}$

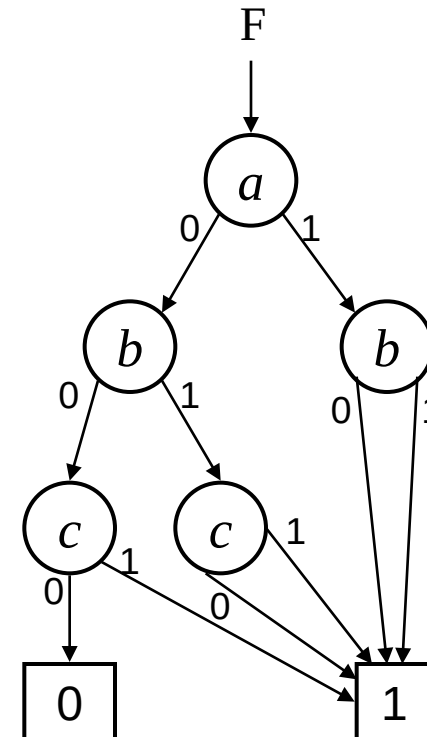
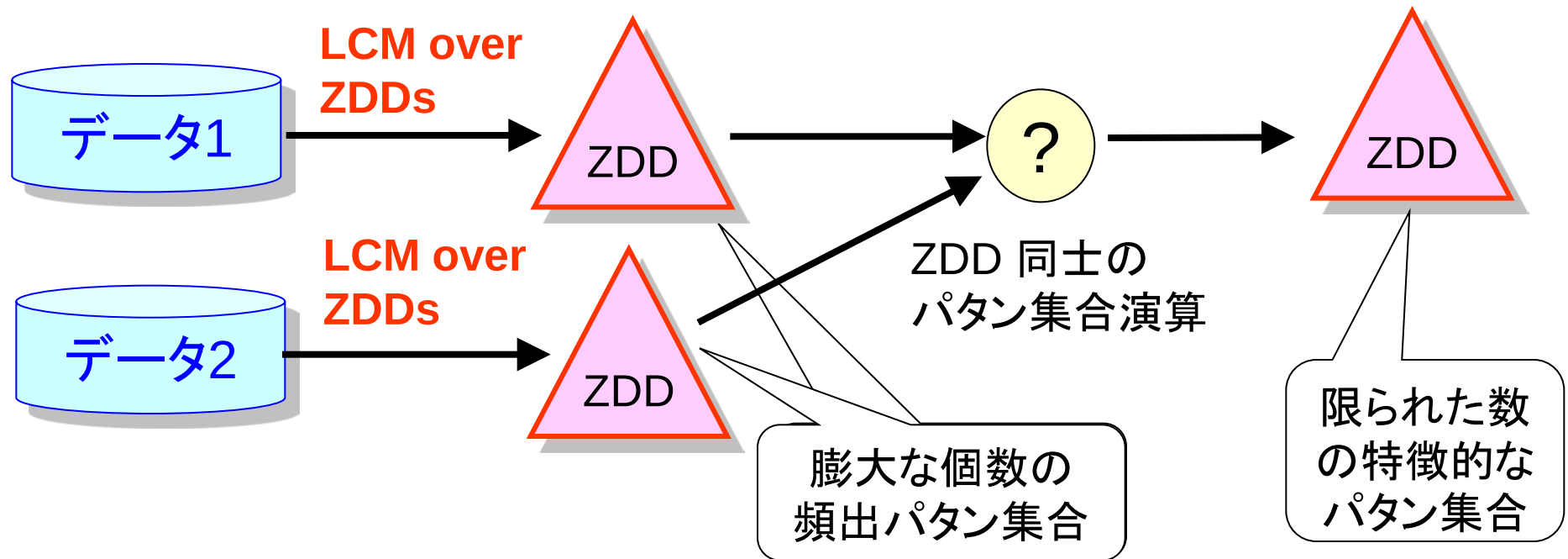


Table 2. Comparison

Dataset name: min. support	# solutions	M over Z	new method		prev. method	
	# itemsets	[ZBDD]	Time(s)	Time(s)	Time(s)	Time(s)
mushroom: 1,000	123,287	760	0.50	0.49	0.64	1.78
500	1,442,504	2,254	1.32	1.30	3.29	3.49
300	5,259,786	4,412	2.25	2.22	9.96	5.11
200	18,094,822	6,383	3.21	3.13	31.63	6.24
100	66,076,586	11,584	5.06	4.87	114.21	6.72
70	153,336,056	14,307	7.16	7.08	277.15	6.97
50	198,169,866	17,830	8.17	7.86	357.27	6.39
T10I4D100K: 100	27,533	8,482	0.85	0.85	0.86	209.82
50	53,386	16,872	0.97	0.92	0.98	242.31
20	129,876	58,413	1.13	1.08	1.20	290.78
10	411,366	173,422	1.55	1.36	1.64	332.22
5	1,923,260	628,491	2.86	2.08	3.54	370.54
3	6,169,854	1,576,184	5.20	3.15	8.14	386.72
2	19,561,715	3,270,977	9.68	5.09	22.66	384.60
BMS-WebView-1: 50	8,192	3,415	0.11	0.11	0.12	29.46
40	48,544	10,755	0.18	0.18	0.22	48.54
36	461,522	28,964	0.49	0.42	0.98	67.16
35	1,177,608	38,164	0.80	0.69	2.24	73.64
34	4,849,466	49,377	1.30	1.07	8.58	83.36
33	69,417,074	59,119	3.53	3.13	144.98	91.62
32	1,531,980,298	71,574	31.90	29.73	3,843.06	92.47
chess: 1,000	29,442,849	53,338	197.58	197.10	248.18	1,500.78
connect: 40,000	23,981,184	3,067	5.42	5.40	49.21	212.84
pumsb: 32,000	7,733,322	5,443	60.65	60.42	75.29	4,189.09
BMS-WebView-2: 5	26,946,004	353,091	4.84	3.62	51.28	118.01

ZDDを用いたパターン集合演算



- 数十億ものパターンを含む集合を圧縮して表現し、ZDDの集合演算を使って効率よく絞込みを行える。
 - 従来の明示的な表現方法では、意味のある解析処理を現実的な時間で行うことは不可能

BDD / ZDD に関する最近の話題

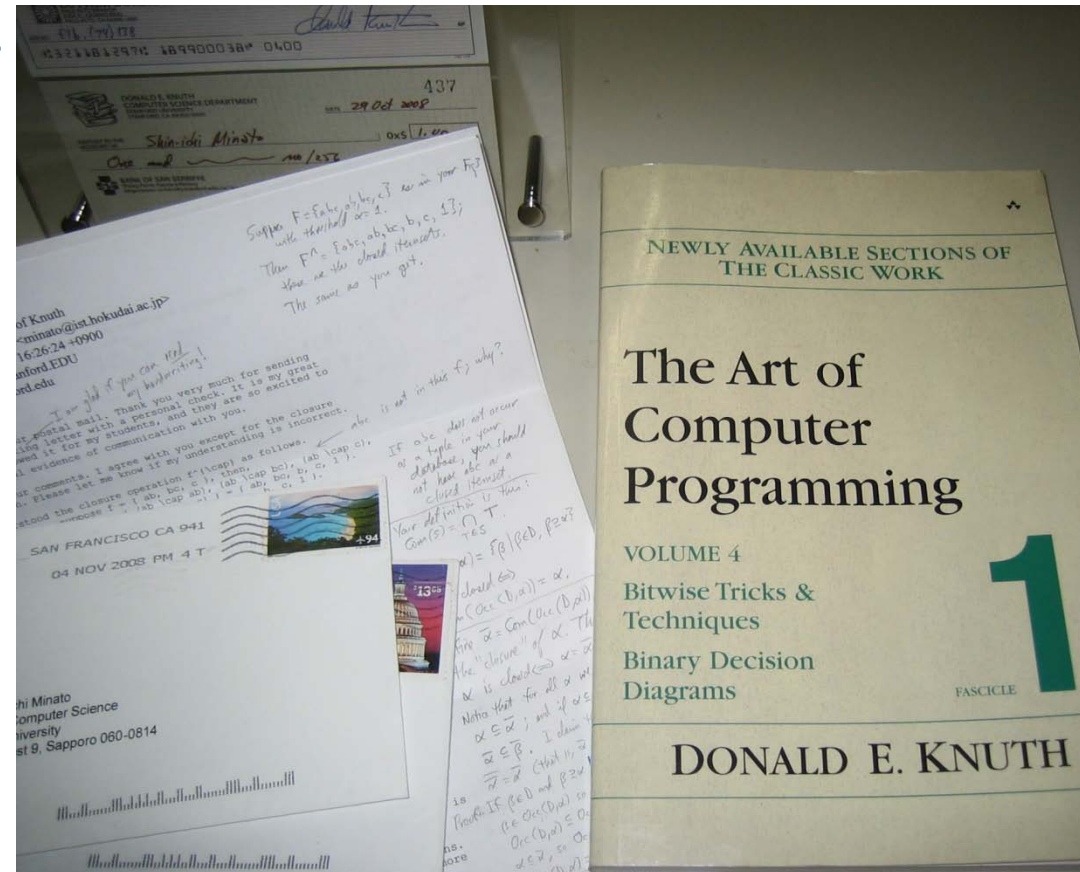
- Knuthの名著「The Art of Computer Programming」の最新巻 (Vol.4, Fascicle 1, 2009) で、BDDが詳しく解説された。

- BDDだけで140ページ、演習264問の大作

- 種々のBDD改良技法の中で、特にZDDを重要視

- 章の後半30ページ以上にわたり、ZDDについて記述
- ZBDDだった呼び名をZDDに短縮。
(BDDと同格扱いに)

- 日本人の研究成果が、このシリーズに項目として詳細に掲載されるのは初めて。
- Knuth氏本人から、ZDD考案者として校正作業への協力を依頼する長文のメールと手紙を受領。
- 2010年5月には湊がKnuth邸を訪問し、プロジェクトの方向性について意見交換を行った。



Algebraic operations for ZDDs

- Knuth evaluated not only the data structure of ZDDs, but more interested in **the algebra on ZDDs**.

$\phi, \{1\}$	<i>Empty</i> and <i>singleton set</i> . (0/1-terminal)	Basic operations (Corresponds to Boolean algebra)
$P.top$	Returns the <i>item-ID</i> at the top node of P .	
$P.onset(v)$ $P.offset(v)$	Selects the subset of itemsets including or excluding v .	
$P.change(v)$	Switching v (<i>add / delete</i>) on each itemset.	
$\cup, \cap, \setminus$	Returns <i>union, intersection</i> , and <i>difference set</i> .	
$P.count$	<i>Counts number</i> of combinations in P .	New operations introduced by Minato.
$P * Q$	<i>Cartesian product set</i> of P and Q .	
P / Q	<i>Quotient set</i> of P divided by Q .	
$P \% Q$	<i>Reminder set</i> of P divided by Q .	

Formerly I called this “unate cube set algebra,” but Knuth reorganized as “**Family algebra**.”

Useful for many practical applications.

Comparison of BDDs and ZDDs

- Many of real-life problems are likely **asymmetric**.

(Symmetric world)

- VLSI Logic Design
- Formal Verification
- etc.

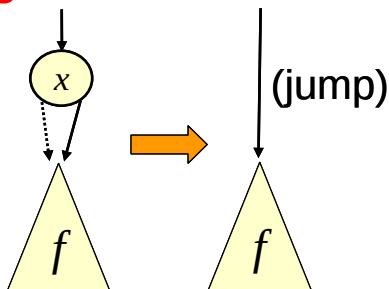
(Asymmetric world)

- Data mining & Knowledge discovery
- Market data analysis
- Formal concept analysis
- Calculation with Bayesian networks
- Machine learning & clustering
- Web data analysis
- Risk analysis
- etc.

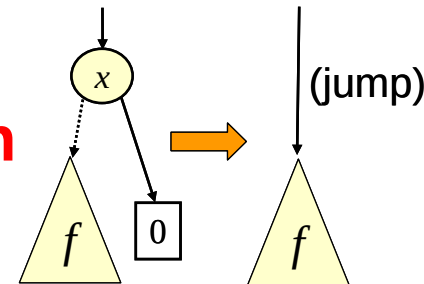
BDD

ZDD

Symmetric reduction



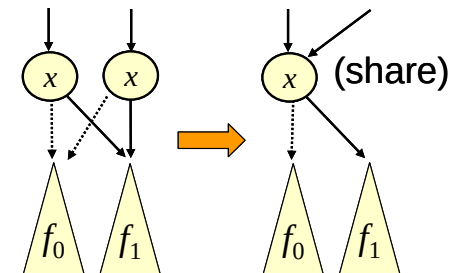
Asymmetric reduction



Quasi-reduced BDD

Sub-graph sharing
(Dictionary-based)

Naïve representation



本講演の内容

- 離散構造の処理とは
- 論理関数と組合せ集合の演算処理系
 - BDD: 論理関数
 - ZDD: 組合せ集合
- 種々の離散構造と演算処理系
 - ZDD-V: 組合せ頻度表
 - SeqBDD: 系列集合
- ERATO湊離散構造処理系プロジェクト
 - プロジェクトの概要
 - 最近の研究(1) ZDDによる超高速パス列挙
 - 最近の研究(2) π DD: 順列集合の演算処理系



組合せ頻度表を用いたデータベース解析

Itemset DB

Record ID	Itemset
1	a b c
2	a b
3	a b c
4	b c
5	a b
6	a b c
7	c
8	a b c
9	a b c
10	a b
11	b c

組合せ頻度表

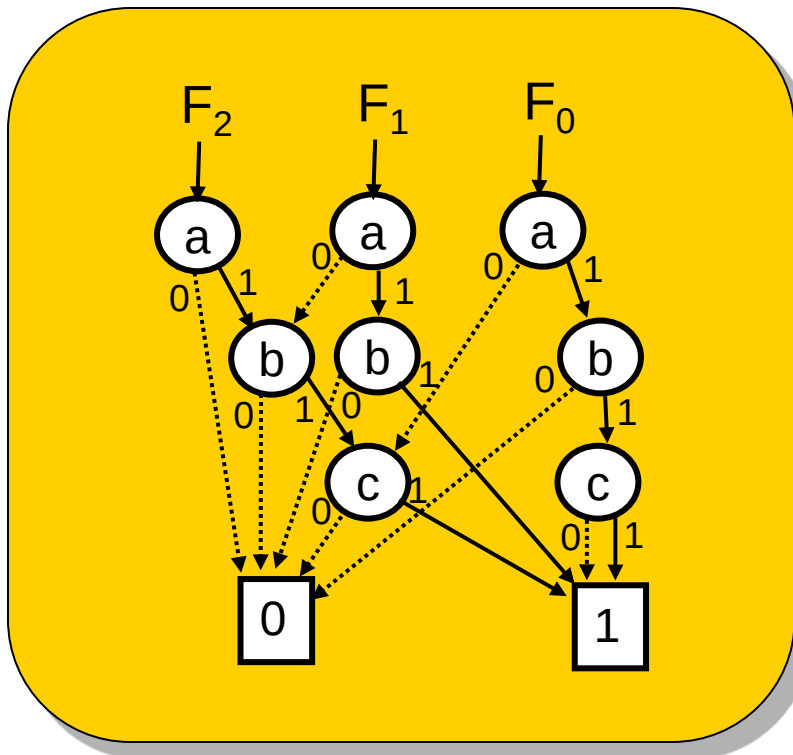
Itemset-histogram

Itemset	Freq.
a b c	5
a b	3
b c	2
c	1

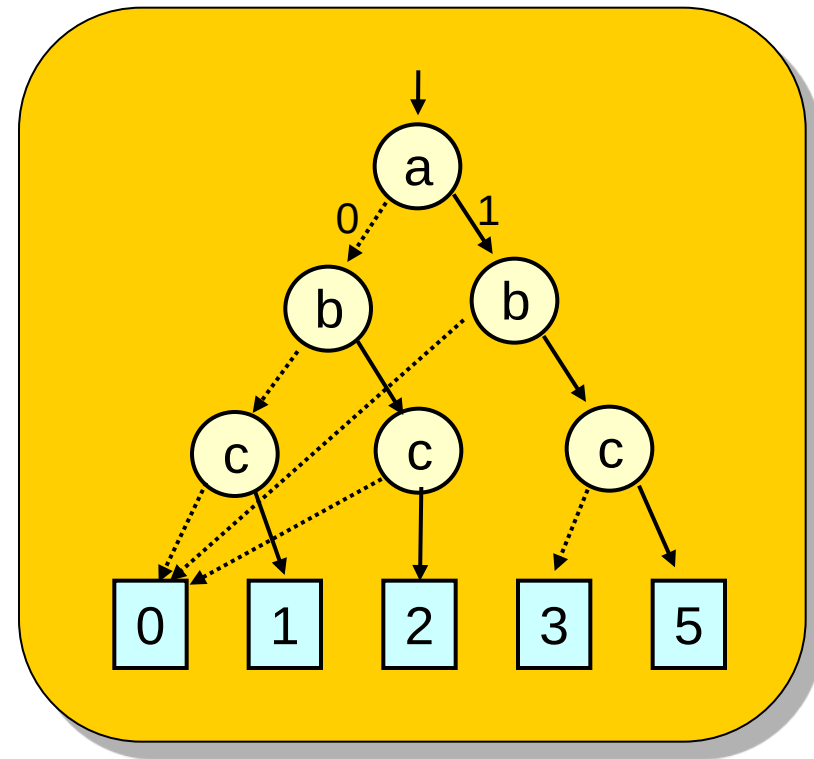
- We sometimes need to represent **not only yes/no** but also the **frequency** (number of occurrence) for each itemset.
 - This is a histogram of itemsets if frequency of all itemsets are generated.
 - Integer-valued Sum-Of-Product form. (VSOP)
 - Function from Combination to Integer. (CtoI)
 - Multi-set of combinations

ZDDによる組合せ頻度表の表現

- ZDDは集合を表すので、そのままでは頻度を表現できない。
- 数値データ表現への拡張方法は2通り知られている。



2進ベクトル表現



Multi-Terminal BDD (MTBDD)

ZDD-Vectorによる組合せ頻度表の表現

- 頻度(出現回数)を2進数で符号化し、2進ベクトルで表現

(例題DB)

タプル
a b c
a b
a b c
b c
a b
a b c
c
a b c
a b c
a b
b c

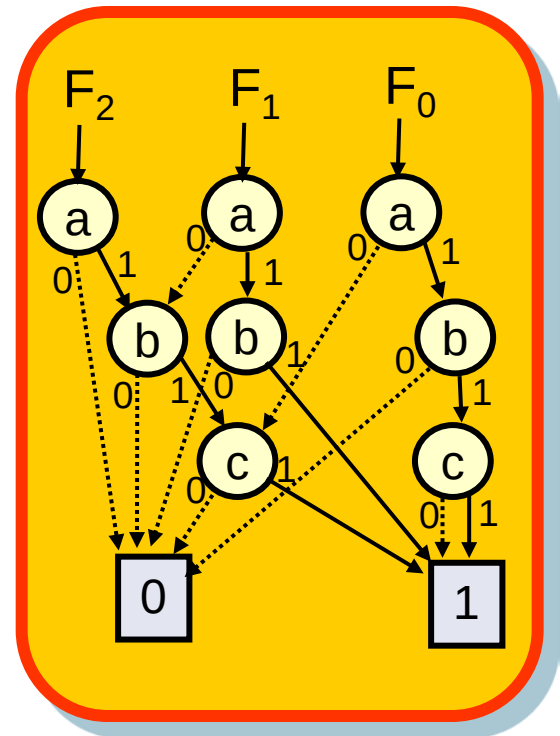
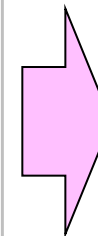


(タプル頻度表)

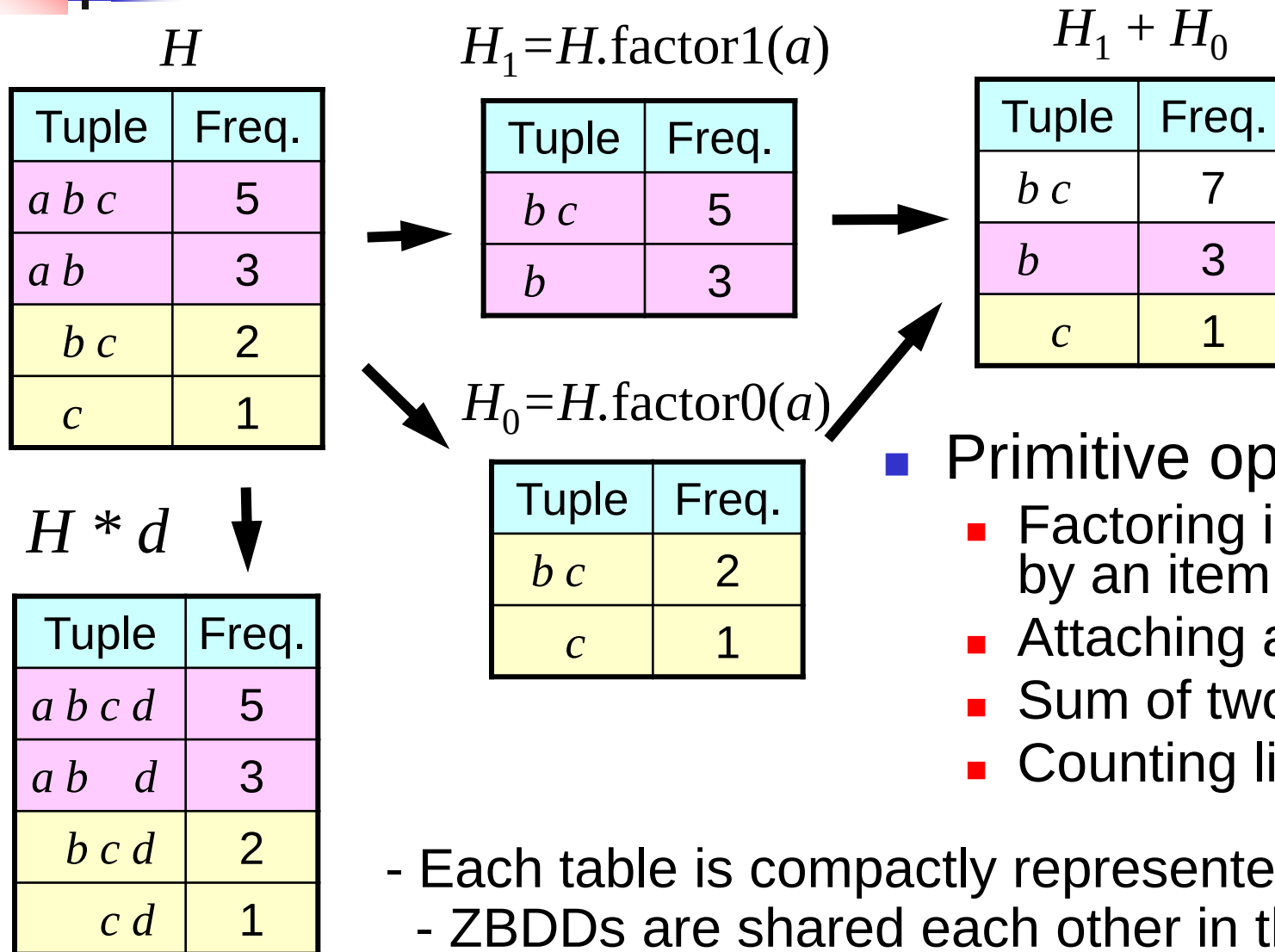
タプル	出現頻度	F_2	F_1	F_0
abc	5 (101)	1	0	1
ab	3 (011)	0	1	1
bc	2 (010)	0	1	0
c	1 (001)	0	0	1

$F_0 = \{abc, ab, c\}$
 $F_1 = \{ab, bc\}, F_2 = \{abc\}$

ZDD-V表現の例



Algebra on itemset-histograms



Algebra on itemset-histograms

H

Tuple	Freq.
$a b c$	5
$a b$	3
$b c$	2
c	1

$H_1 = H.\text{factor1}(a)$

Tuple	Freq.
$b c$	5
b	3

$H_0 = H.\text{factor0}(a)$

Tuple	Freq.
$b c$	2
c	1

$H_1 + H_0$

Tuple	Freq.
$b c$	7
b	3
c	1

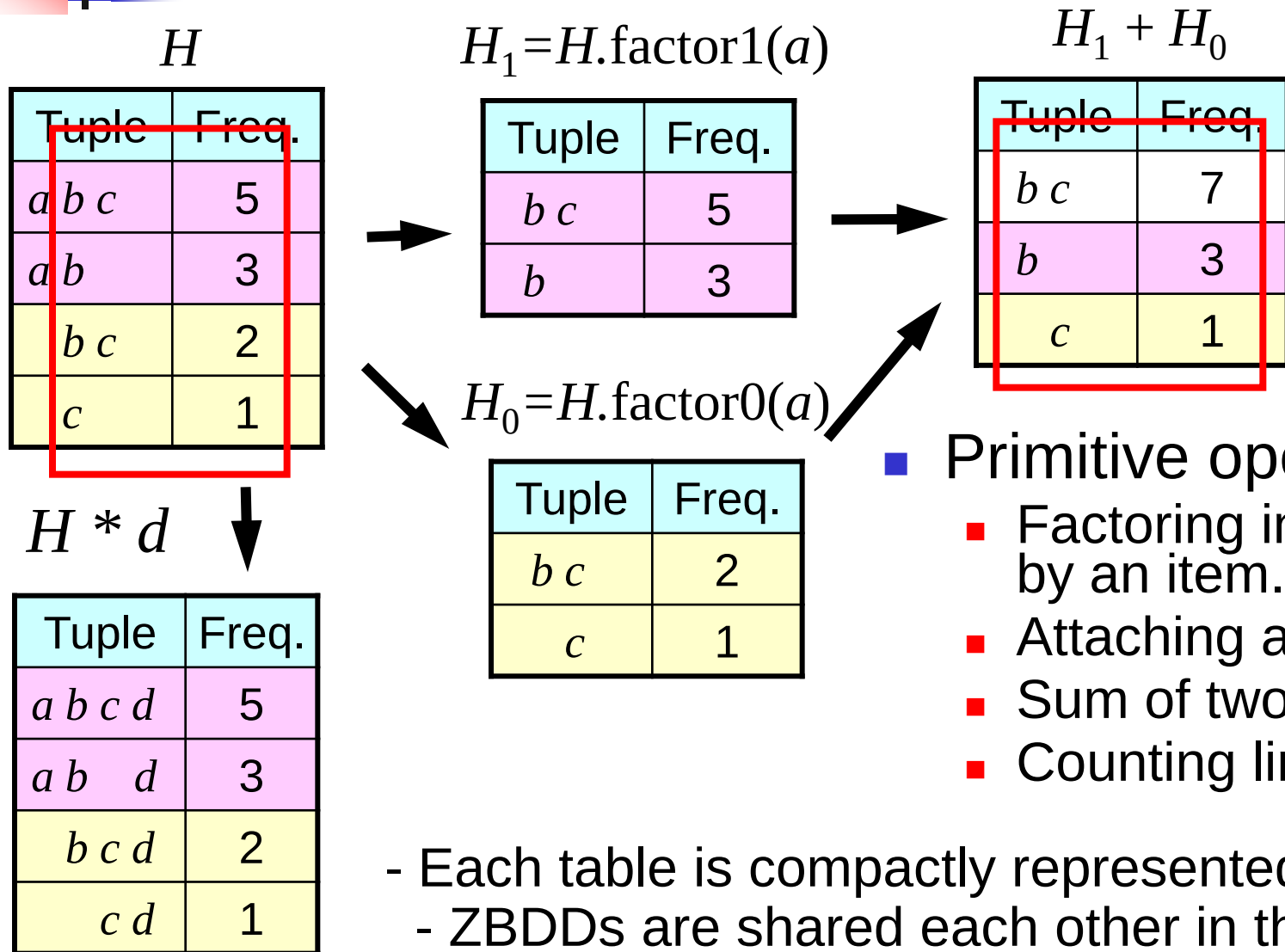
$H * d$

Tuple	Freq.
$a b c d$	5
$a b d$	3
$b c d$	2
$c d$	1

- Primitive operations:
 - Factoring into two parts by an item.
 - Attaching an item.
 - Sum of two histograms.
 - Counting lines in the table.

- Each table is compactly represented by ZBDDs.
- ZBDDs are shared each other in the memory.

Algebra on itemset-histograms

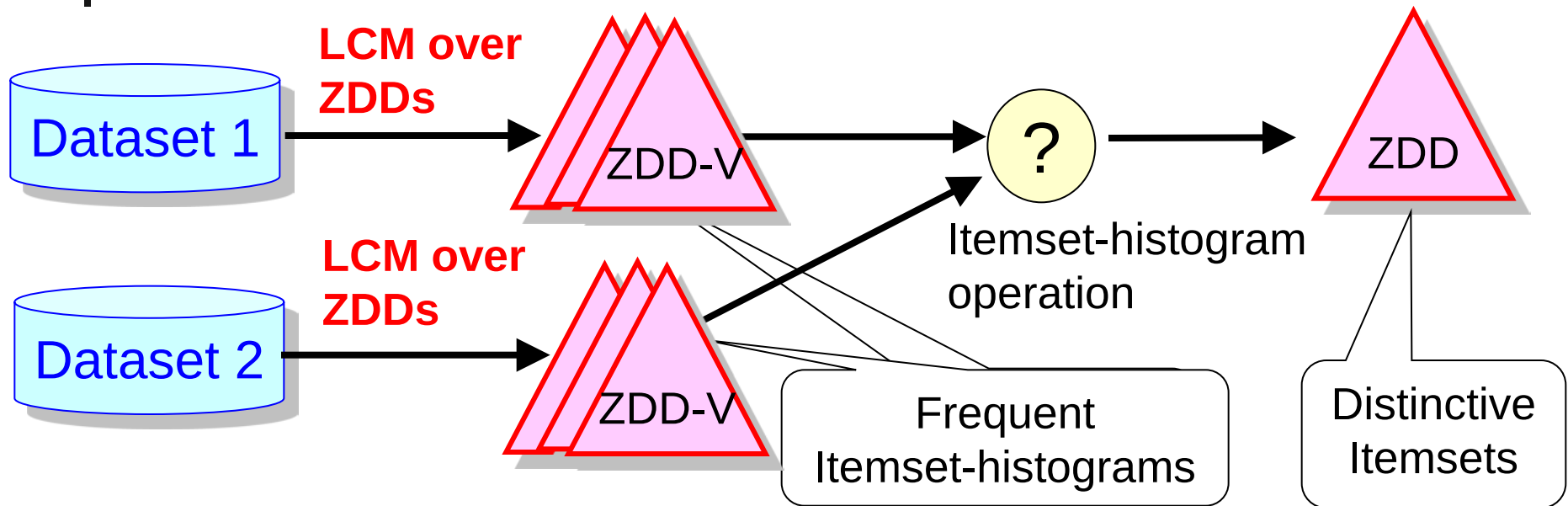


組合せ頻度表の基本演算 (Itemset-Histogram Algebra)

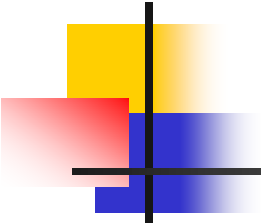
$c \ (c \in N)$	λ だけが c 回出現する頻度表を返す. ($c = 0$ のときは空集合を表す.)
$P.top$	頻度表 P の最上位の節点のアイテム番号を返す.
$P.offset(v)$	P の中でアイテム v を含まない組合せを集めた部分頻度表を取り出す.
$P.onset(v)$	P の中でアイテム v を含む組合せを集めた部分頻度表を取り出し, その各組合せから v を取り除いた頻度表を返す.
$P.change(v)$	P の各組合せについて, アイテム v の有無を反転させる.
$P + Q$	P と Q の対応する組合せ同士の頻度を算術加算した頻度表を返す.
$P - Q$	P と Q の対応する組合せ同士の頻度を引き算した頻度表を返す. (ただし減算結果が負になる場合は 0 とする.)
$P.count$	P の組合せの種類数を数える.
$P.upperbound$	P の組合せのうち最大頻度の数値を返す.
$P.unitset$	P に含まれる組合せの頻度をすべて 1 に正規化した頻度表を返す.

$max(P, Q)$	$P + (Q - P)$ を返す.
$min(P, Q)$	$P - (P - Q)$ を返す.
$P * Q$	P と Q の直積集合 (Cartesian product).
P/Q	P を Q で割った商 (quotient).
$P \% Q$	P を Q で割った剰余 (remainder).

Post Processing after LCM over ZDD(V)s



- The result can be analyzed flexibly using itemset-histogram operations.
 - Extracting long/short frequent patterns.
 - Comparison of two sets of frequent patterns.
 - Calculating statistical data (e.g. confidence, support)
 - Finding disjoint sub-factors in frequent patterns.

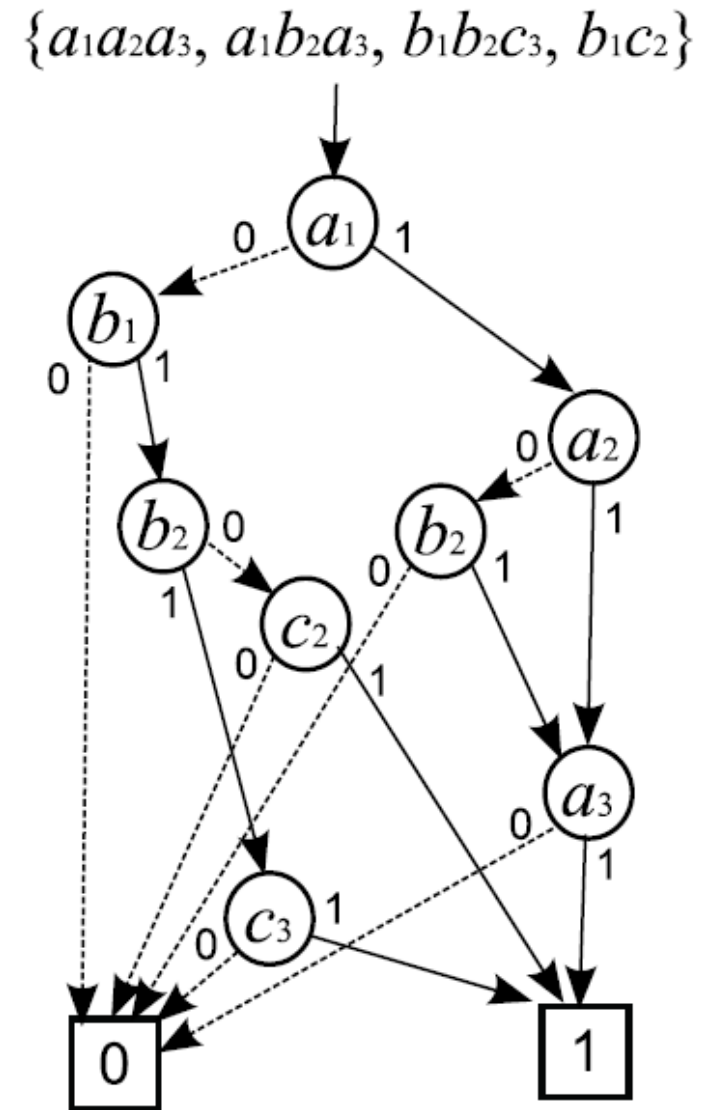


系列集合(Sets of Sequences)

- Sets of combinations:
 - Don't consider order and duplication of items
 - “abcc” and “bca” are the same.
- Sets of sequences:
 - Distinguishes all finite sequences.
 - $\varnothing$, $\{\lambda\}$, $\{ab, aba, bbc\}$, $\{a, aa, aaa, aaaa\}$, etc.
 - Here we exclude infinite sets such as $\{a^*\}$.
- So many real-life applications.
 - Text search and indexing
 - Web (html/xml) data mining
 - Bio informatics

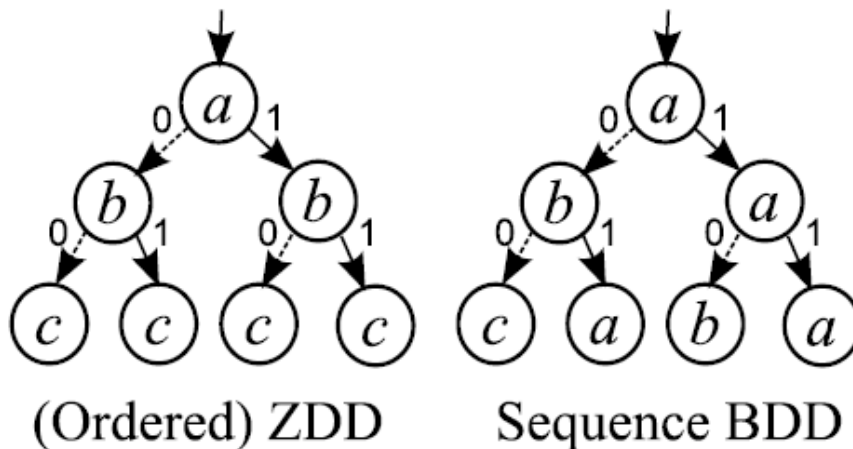
Encoded ZDDs for Sets of sequences

- Pair of (Item - position) is considered different symbol.
“aaa” $\rightarrow$ “a1 a2 a3”
“aba” $\rightarrow$ “a1 b2 a3”
- Alphabet size: $|\Sigma|$
Maximum length of sequences: n
Total encoded symbols: $|\Sigma| \times n$
- Not very efficient.
 - Many symbols needed.
 - We need to put a fixed maximum length of sequences.

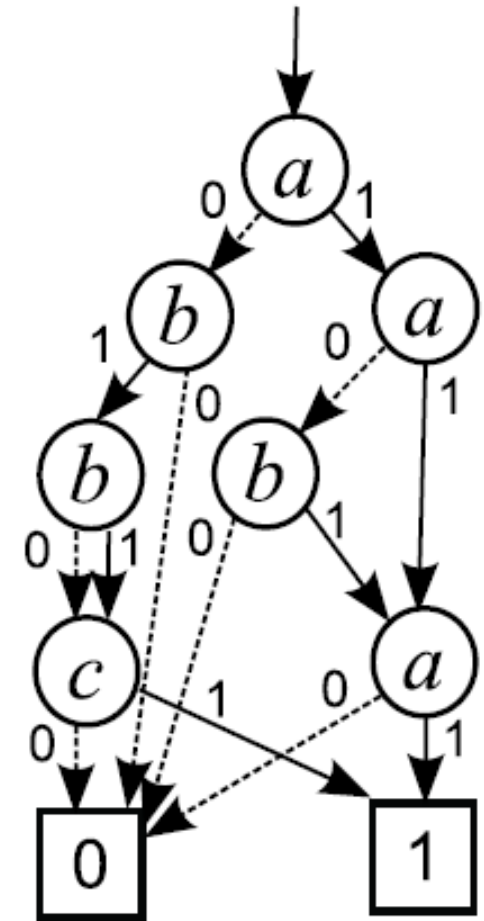


Sequence BDD (SeqBDD)

- Loekito, Bailey, and Pei (2009)
 - Same as ZDD reduction rule.
 - **Only 0-edges** keep variable ordering.
 - 1-edges has no restriction.
 - Still unique representation for a given set of sequences.
 - Each path from root to 1-terminal corresponds to a sequence.



$\{aaa, aba, bbc, bc\}$



SeqBDDの基本演算 (Sequence Family Algebra)

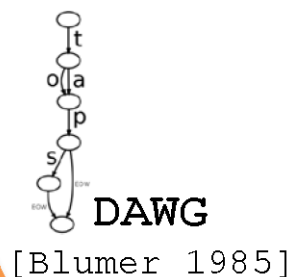
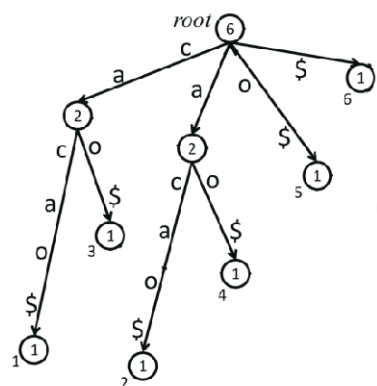
$\emptyset$	空集合. (0-定数節点を返す.)
$\{\lambda\}$	空列のみからなる集合. (1-定数節点を返す.)
$P.top$	P の最上位の節点の文字 ID を返す.
$P.onset(x)$	文字 x で始まる系列の部分集合を抽出し, 各系列の先頭の x を取り除く.
$P.offset(x)$	文字 x 以外で始まる系列の部分集合を抽出する.
$P.push(x)$	P の各系列の先頭に文字 x を付加する.
$P \cup Q$	P と Q の結び (union).
$P \cap Q$	P と Q の交わり (intersection).
$P \setminus Q$	差集合. (P にあって Q にないもの.)
$P.count$	P の要素数を数える.
$P * Q$	P と Q の直積集合 (Cartesian product). P と Q から任意の一つの系列を取り出し連結した系列の集合.

■ ZDD風のalgebra

- *onset, offset, push* 演算がZDDと異なる。
- 他の演算はほとんど同じ。

Suffix Tree

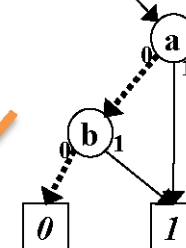
[McCreight 1976]



- Substring
- No operations

ZDD

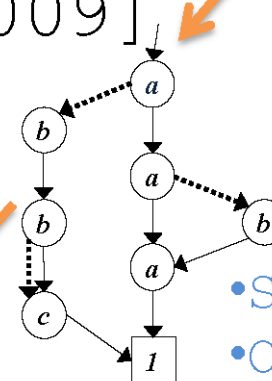
[Minato 1993]



- Combination
- Operations

SeqBDD

[Loekito 2009]



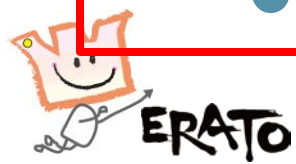
- String
- Operations



Suffix-DD

本講演の内容

- 離散構造の処理とは
- 論理関数と組合せ集合の演算処理系
 - BDD: 論理関数
 - ZDD: 組合せ集合
- 種々の離散構造と演算処理系
 - ZDD-V: 組合せ頻度表
 - SeqBDD: 系列集合
- ERATO湊離散構造処理系プロジェクト
 - プロジェクトの概要
 - 最近の研究(1) ZDDによる超高速パス列挙
 - 最近の研究(2) π DD: 順列集合の演算処理系



ERATOについて

■ JSTの戦略的創造研究推進事業

- さきがけ(牧場型)、CREST(ハケ岳型)、**ERATO(富士山型)**

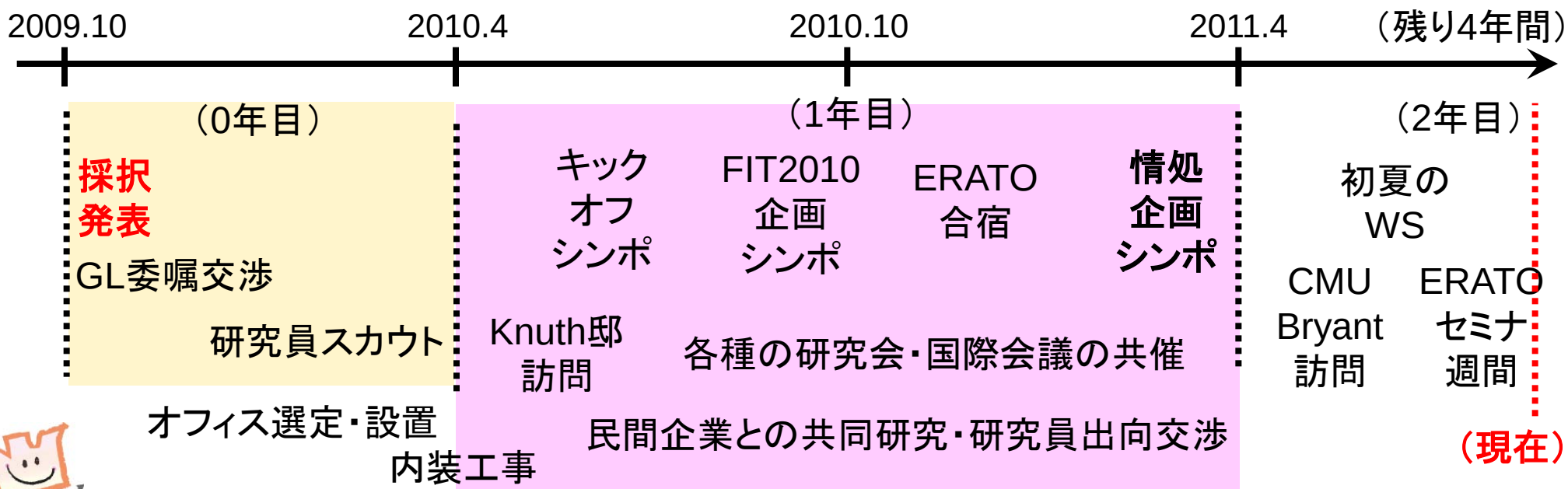
■ ERATOプロジェクトの特徴

- 新しい科学技術の源流を作るような研究を支援。
- 昭和56年発足。過去に98プロジェクトを採択。
(科学技術の全分野で年4~5件)
- プロジェクト期間:5年半 研究費総額:10~15億円(テーマによる)
- メンバ規模:10~15人(人件費に依存)
- 研究総括に自由裁量を与え、分野・組織にとらわれずに機動的なプロジェクトを構成。
- 単なる助成金ではなく、JSTが主体的にプロジェクトを推進。
本務研究室とは別に独自のオフィスを設置。



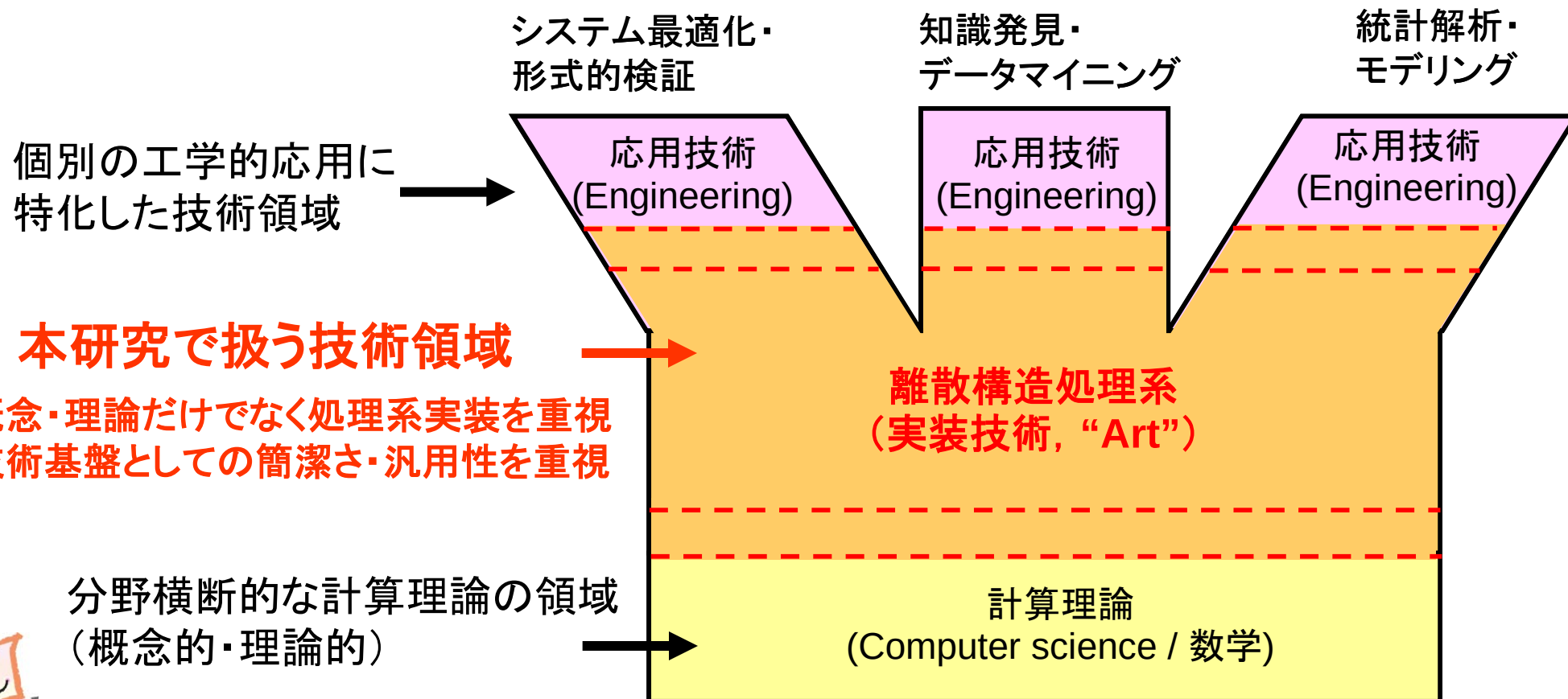
ERATOプロジェクトの選考過程と開始準備

- 公募ではなく、JSTが持つ1000人規模の研究者リストの中から、約10名まで候補を絞り込み、本人に研究計画の応募を依頼。
 - 自分からは応募できない。突然知らされる。研究内容は自由。競争率10倍。
- 応募者の中から書類審査・面接審査により各分野で1人を採択。
- 最初の半年間で、プロジェクトの骨格を準備
 - GL委嘱、オフィス施設計画・賃貸交渉、ポスдок研究員のスカウト ...

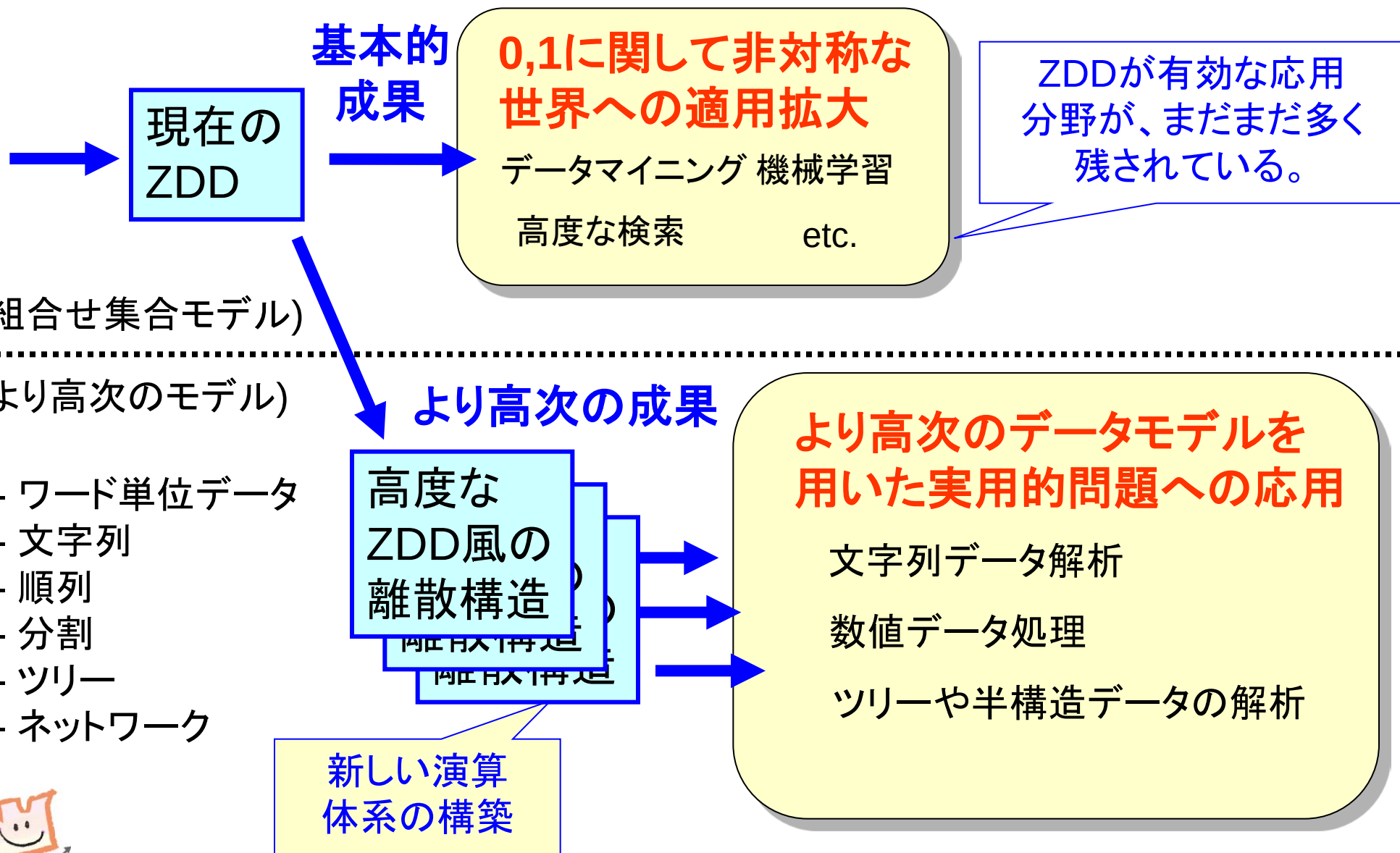


本研究プロジェクトの対象領域

BDD/ZDD技術の新しい切り口として、様々な離散構造を**統合的に演算処理**する技法を体系化し、**分野横断的かつ大規模な実問題**を高速に処理するための技術基盤を構築する。



本研究プロジェクトの技術面のポイント



研究実施場所とメンバ

■ 札幌メインオフィス(北大・情報科学研究科)

- ポスドク研究員4名
- 事務スタッフ3名、技術員1名、RA3名
- 北大研究者(GCOE他)と連携



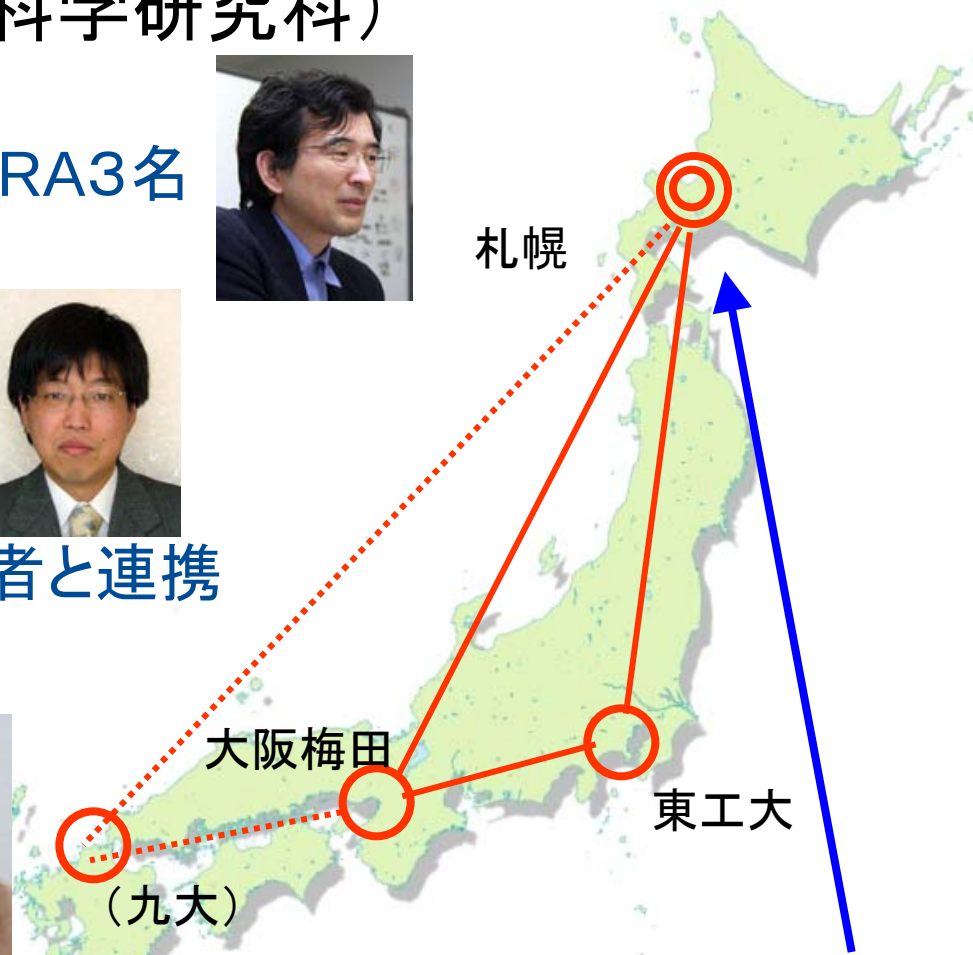
■ 東京サテラボ(東工大)

- GL:産総研・津田宏治氏
- ポスドク研究員1名、RA1名
- 東工大GCOE・東京地区研究者と連携



■ 大阪サテラボ(梅田駅北口)

- GL:阪大・鷺尾隆 教授
- 技術員1名、RA2名
- 関西地区研究者と連携



各地区は高品質TV会議
システムで常時接続

フルタイムの若手研究者は
極力、札幌に常駐させて、
活性化・相乗効果を図る。



札幌メインオフィス

■ 情報科学研究科に隣接する工学部本館内

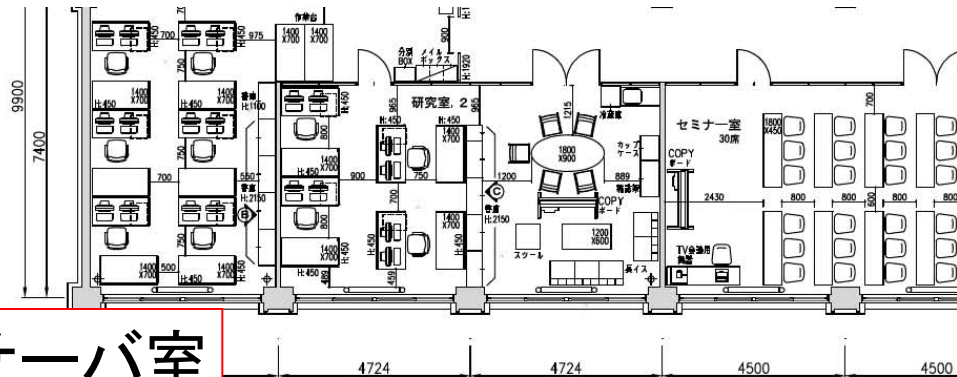
- 部局長の特別の措置でERATOに貸出し
- 約300m²の独立スペース
- 利便性も抜群
(生協食堂まで徒歩1分)
- 北大情報系の研究者との交流が容易。

情報科学研究科(本務研究室)

ERATOオフィス



(北大オフィスレイアウト)



北大オフィス現況



ERATO研究員(初年度採用者)



白井康之
(技術参事兼務)



櫻井祐子



吉仲 亮



川原 純



斎藤 寿樹



田部井 靖男
(東京サテラボ所属)

最近のメンバ異動



鶴間 技術員
(2010.11採用)
(NEC北海道より出向)



櫻井 研究員
(2011.3 転出)
(九大准教授)



岩下 研究員
(2011.4 採用)
(富士通研より出向)

井上 研究員
(2011.6 採用)
(NTT研究所より出向)

植野 研究員
(2011.4 採用)
(大阪サテラボ所属)

東 技術員
(2011.4 採用)
(大阪サテラボ所属)

富田悦次 研究推進委員
(2011.6 採用)
(東京サテラボ所属)

伝住 RA
(北大・有村研M2)
他 RA 数名(札幌・東京)

海外研究機関との連携

- Personal relationships with important researchers.
(→ prospective groups for inviting visitors)



2010年度に実施した主なイベント

- Stanford大学 Knuth邸訪問(4月)
 - 湊と宇野先生が訪問し、プロジェクトの方向性について助言を得た
- ERATOキックオフシンポジウム(5月) @北大
 - 30名超の研究者が北大に集まり全員が発表、熱い議論を展開
- FIT2010企画イベント(9月) @九大
- 国際ワークショップ(ドイツ開催)で招待講演(9月)
 - 開催地大学のプレスリリースに顔写真入りで紹介
- UCLA Darwiche教授 招聘(11月)
 - ベイジアンネットと命題論理で世界的に著名。相互に有意義な議論
- ERATO合宿シンポジウム(12月) @札幌近郊
 - 2泊3日で25件の密度の濃い研究発表。深夜まで議論尽きず。
- 情処全国大会企画イベント(3月) @東工大
 - 大震災の9日前でした。



Knuth邸訪問

- 2010年4月：
国際会議SDM2010 (SIAM Data Mining Conference)
発表のため米国出張 (NII・宇野先生と共著)
- 往路にStanford大学・Knuth氏のオフィス (自宅) を訪問
 - 湊から電子メールにて訪問を依頼
 - 教科書掲載の挨拶と学生への小切手受領のお礼
 - 最近のBDD/ZDD関連技術に関するディスカッション



Knuth邸訪問(パイプオルガンと書斎)



Knuth邸での議論

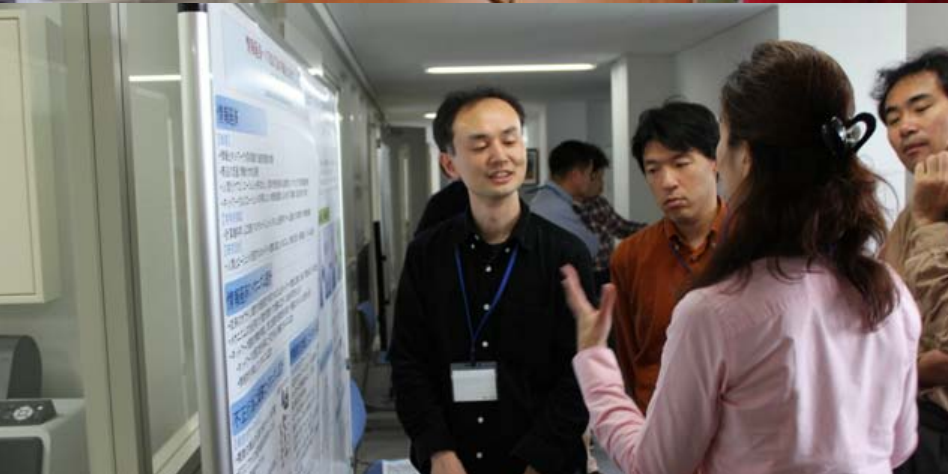
- 有限系列集合を表現する「Sequence BDD」とその応用について議論
 - 非常に興味を持っておられた。
- Knuth本にZDDが載ったことがきっかけで、私が大変なことになってしまっていることを報告
 - 「自分が教科書に何かを書くと、いろいろな人に影響を与えることは認識している。だから私にも少しは責任がある」
 - 「じゃあプロジェクトの今後の研究テーマについて、少し語ろう」
 - Knuth先生が考えるFuture workをいくつか挙げて下さった。基本的には、私がERATOの面接資料に書いた内容とほぼ一致。「ZDDをベースとした、より高度な離散構造のAlgebraの構築」

プロジェクトの方向性は間違っていないことを確認し、意を強くした



キックオフシンポジウム

- 30名超の研究者が北大に集まり、ほぼ全員が発表、熱い議論を展開



キックオフシンポジウム懇親会

■ 北大キャンパス内でのジンギスカン



バーベキュー会場にホワイトボードやノートPC
を持ちこんで議論を継続

ERATOセミナーの開催

■ ほぼ毎週、第一線で活躍中の研究者に講師を依頼して ERATOセミナー(研究会)を開催

- 興味のある研究者をERATOセミナー会員として登録
- メーリングリストへの登録・イベント案内の送付
- 定例のERATOセミナーへの自由参加
- 好きな時間に北大オフィスまたは各地区サテラボに立ち寄り、ERATOメンバとお茶を飲みながら議論して頂く

2010年度
開催数:39件

現在の会員数
約200名超

■ ERATOメンバとの具体的な共同研究テーマが定まって 来たら、数人単位で「ERATO連携研究」を推進

- 連携研究者として共著論文執筆や基盤ソフト開発等に参加。
- 必要な研究費(旅費・謝金・物品等)はJST直轄予算からサポート。
- 知財処理等が必要な場合は別個の共同研究契約に発展させる



最近の研究成果から

■ グラフのパス列挙

- KnuthがZDDを用いた超高速アルゴリズムを提案
- Knuth法の詳細を分析し、制約条件を扱うための改良法を開発
- 通信ネットワークや電力ネットワークの解析への応用

■ 順列集合の処理アルゴリズム (π DD)

- 「組合せ集合」から「順列集合」へ
- ランキング、マッチング、単射写像
- 信号処理、ロスレス符号、暗号

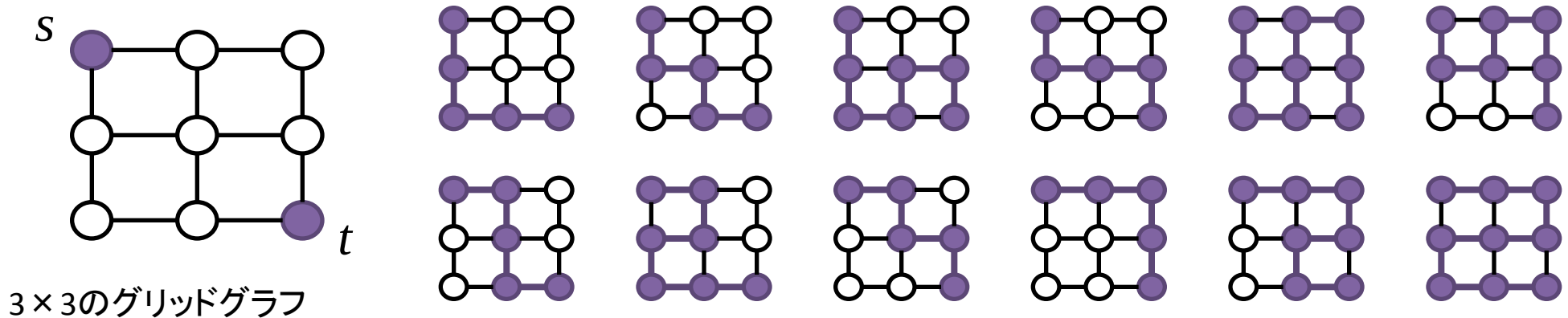
■ BDD演算アルゴリズムの計算量の理論的評価

- 25年間信じられてきたBryantのconjectureの反例を発見
- 入力・出力サイズの線形時間で計算できない場合がある
(実用的には、依然としてほぼ線形時間と考えて良い)



パスの列挙

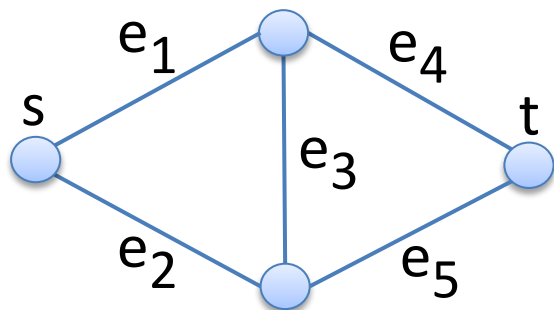
- 入力: グラフ $G=(V, E)$, 2頂点 s, t
- 出力: s から t までのパスを列挙 (保持したZDD)



応用は極めて多い

地理情報処理、ネットワークや論理回路の信頼性評価
電力網制御 (スマートグリッド等)、ソフトウェア解析、etc.

Knuth によるアルゴリズム Simpath

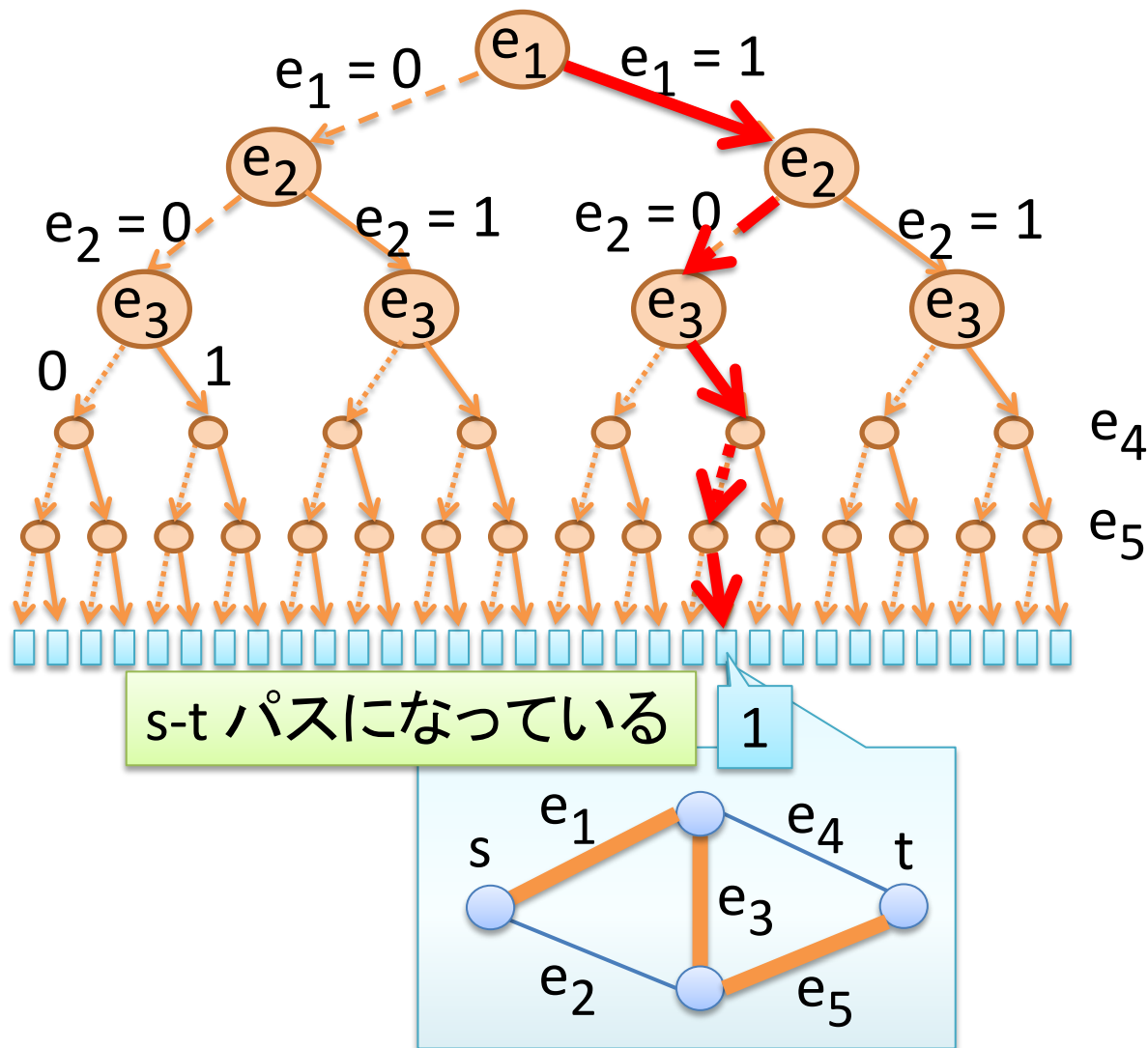


1. 枝に順番を付ける
(例えば、s から幅優先)
(もっと良い方法もあり)

2. 2分木を構築

枝 $e_1, e_2, \dots$ の順に処理

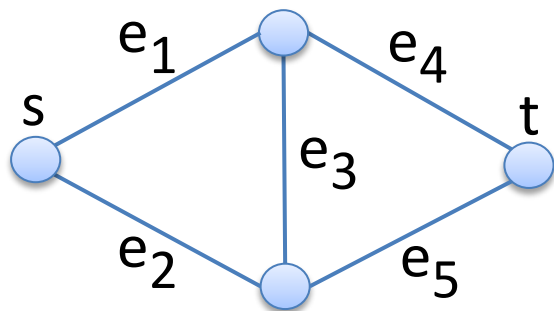
各枝変数 e_i に対し、
 $e_i = 0$ or 1 を決めていく



$e_i = 1$ である枝が s-t パスになっているか？



Knuth によるアルゴリズム Simpath

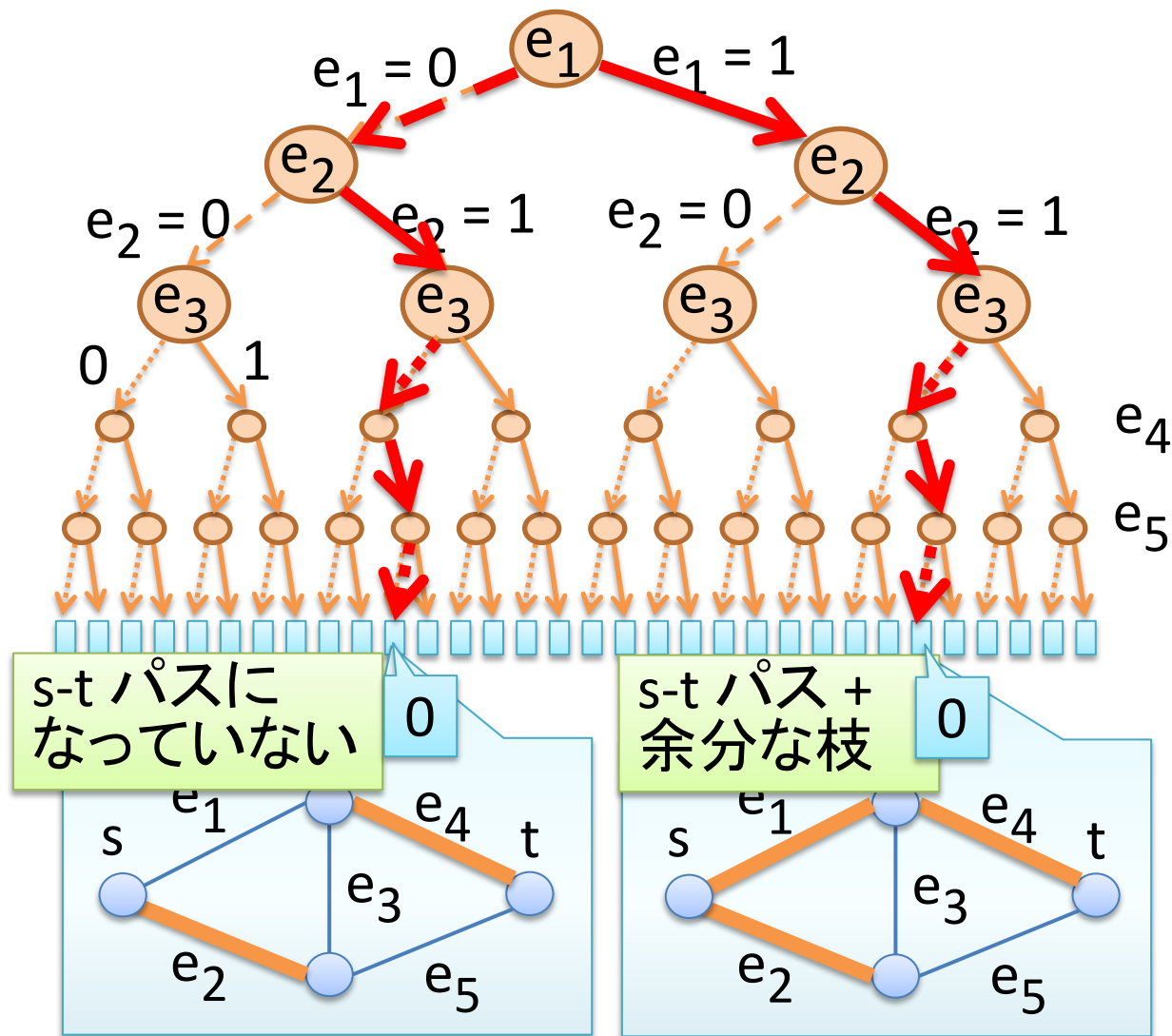


1. 枝に順番を付ける
(例えば、 s から幅優先)
(もっと良い方法もあり)

2. 2分木を構築

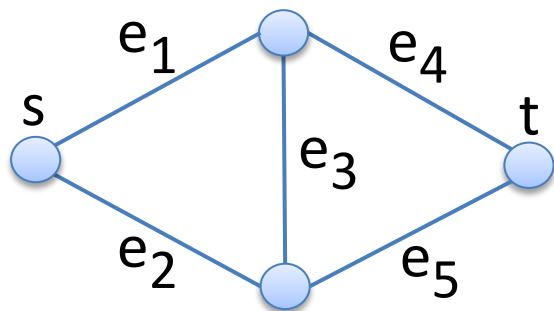
枝 $e_1, e_2, \dots$ の順に処理

各枝変数 e_i に対し、
 $e_i = 0$ or 1 を決めていく



$e_i = 1$ である枝が s - t パスになっているか？

Knuth によるアルゴリズム Simpath

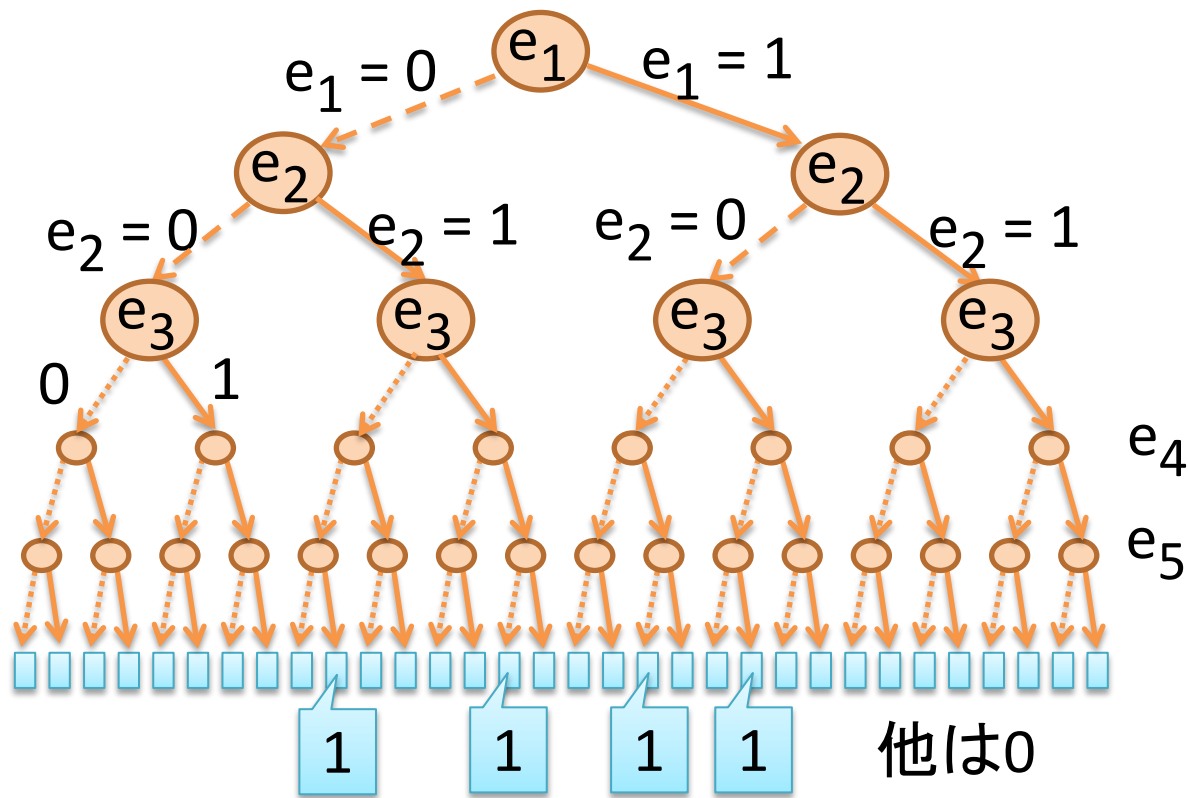


1. 枝に順番を付ける
(例えば、s から幅優先)
(もっと良い方法もあり)

2. 2分木を構築

枝 $e_1, e_2, \dots$ の順に処理

各枝変数 e_i に対し、
 $e_i = 0$ or 1 を決めていく

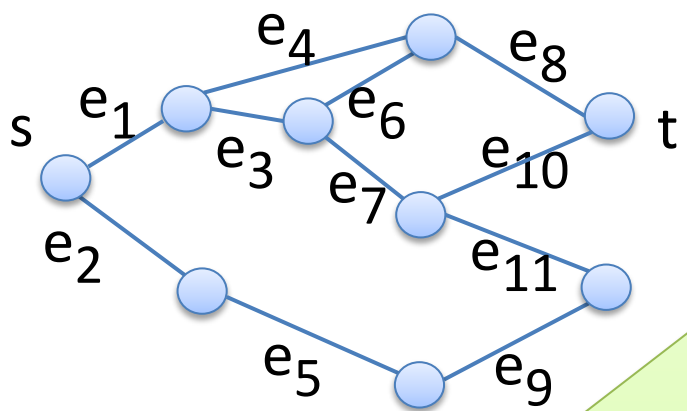


1 が1つのパスに対応

2分木は全パスを保持している

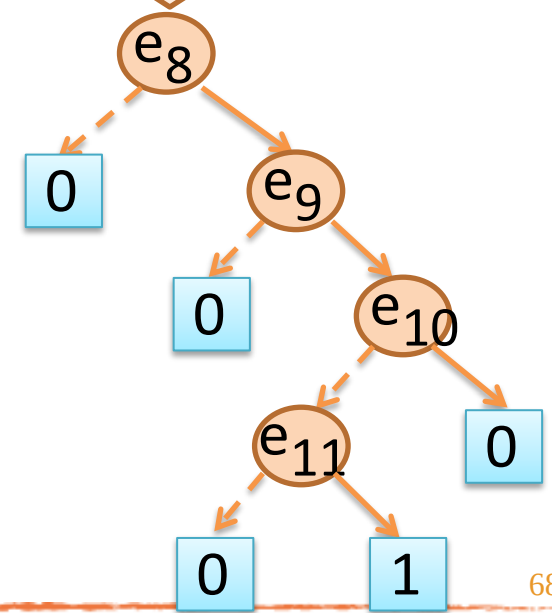
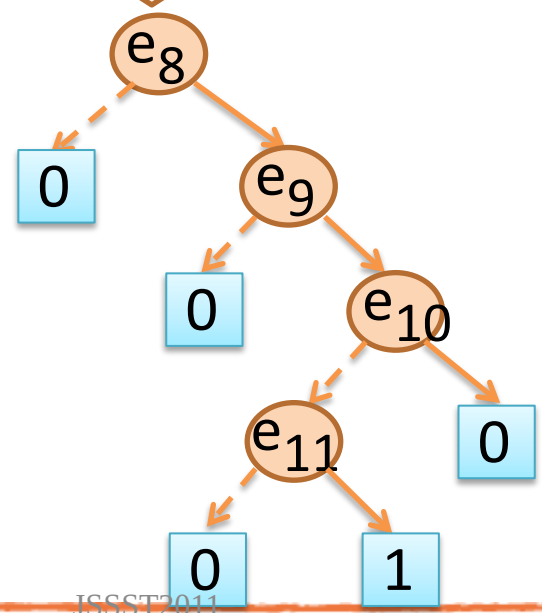
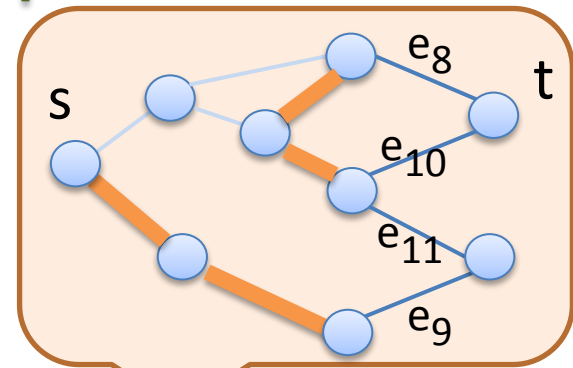
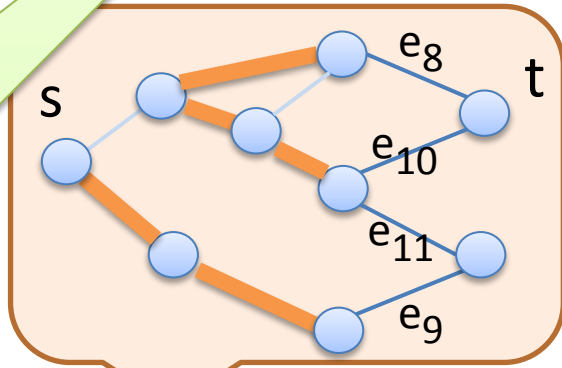
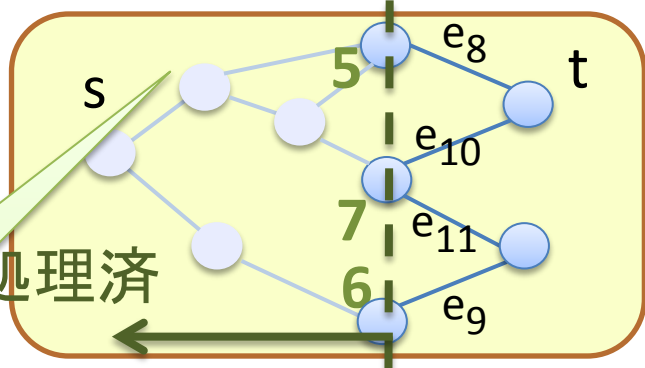


Knuth によるアルゴリズム Simpath

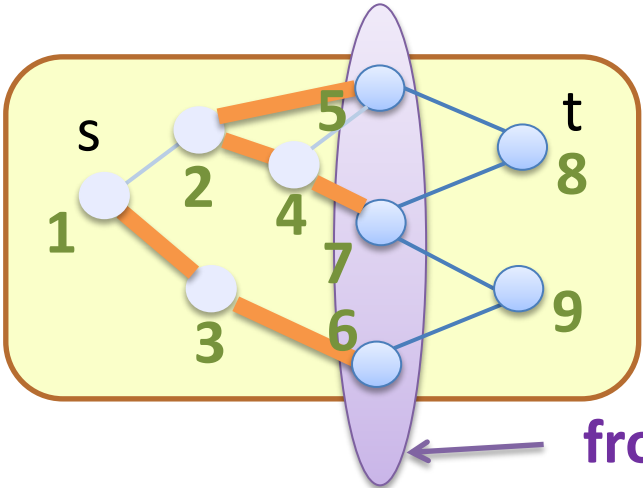


6 は s とつながっている
5 は 7 とつながっている

処理済



接続情報の記憶法



mate この部分のみ記憶

i	1	2	3	4	5	6	7	8	9
mate[i]	6	0	0	0	7	1	5	8	9

- 頂点がパスの端 → 逆端の番号
- 頂点がパスの途中 → 0
- 頂点がいずれのパスにも含まれない → 自身の番号

どの頂点の接続情報を覚えればよいか？

枝 i を処理するとき、
枝 i に接続している頂点を「訪れた」と表現する



$F := \{\text{訪れたことのある頂点}\} \setminus \{\text{二度と訪れない頂点}\}$

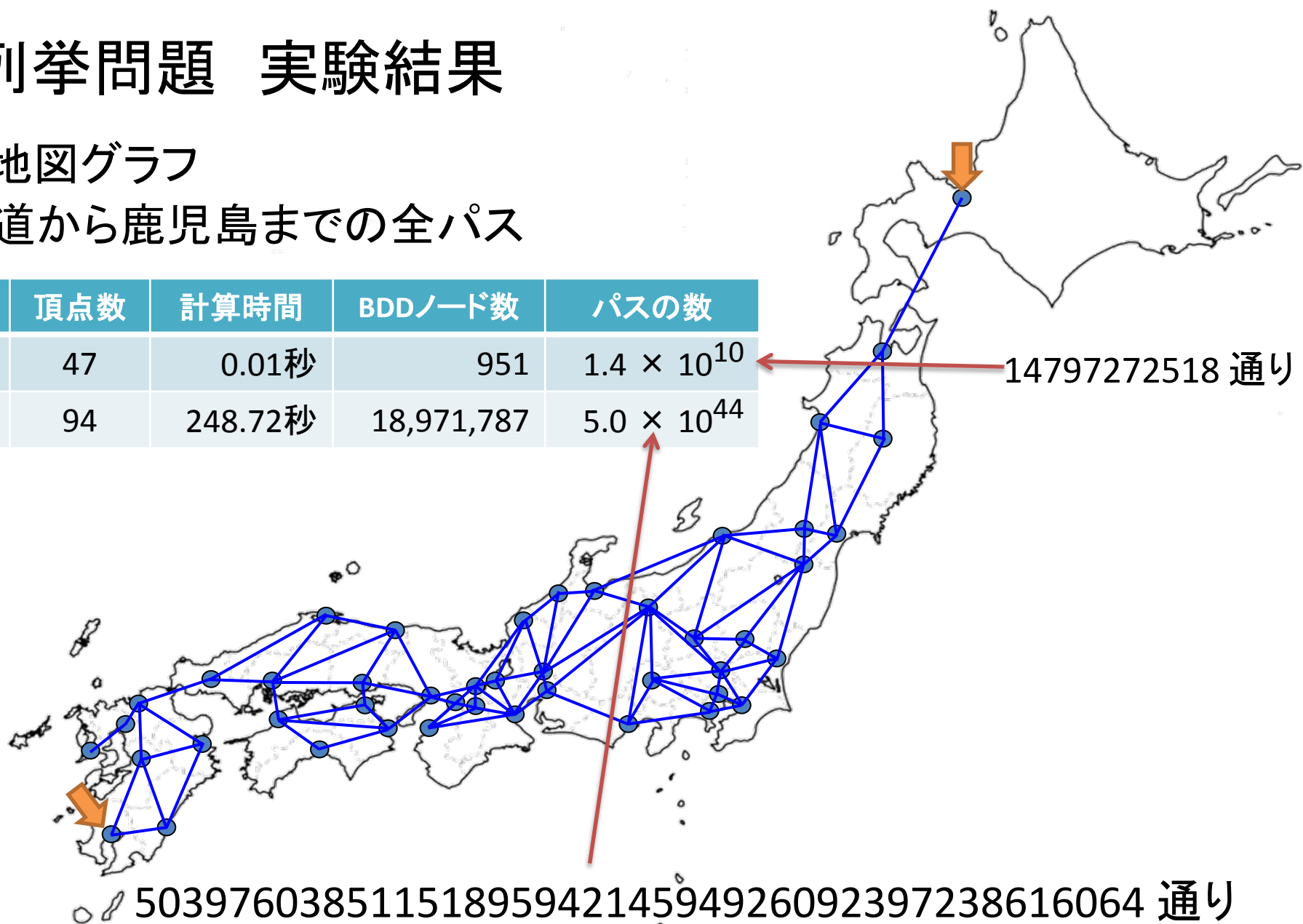
F を **frontier** という。frontier 頂点の接続情報を覚える

パス列挙問題 実験結果

日本地図グラフ

北海道から鹿児島までの全パス

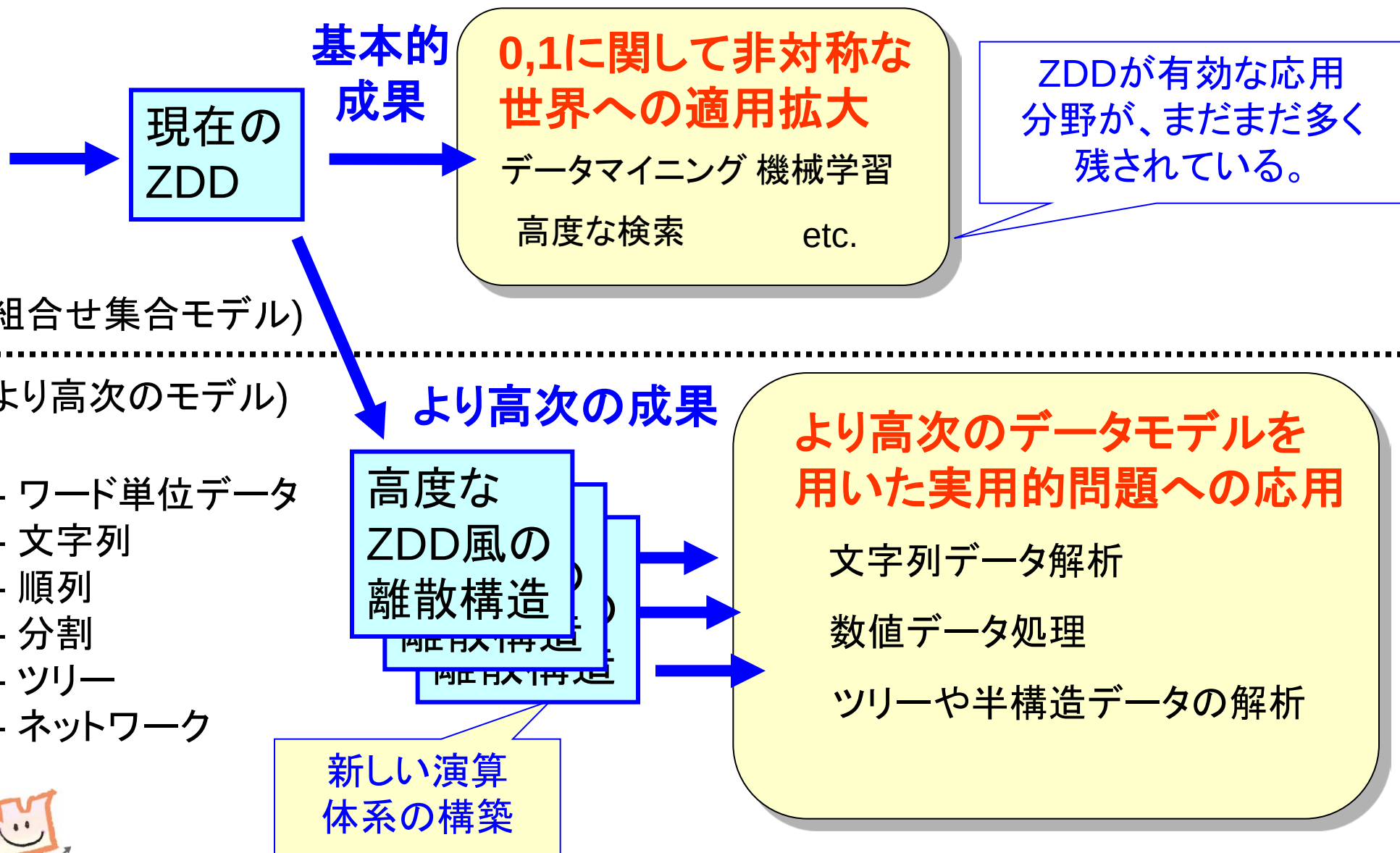
	頂点数	計算時間	BDDノード数	パスの数
日本地図	47	0.01秒	951	1.4×10^{10}
2重化	94	248.72秒	18,971,787	5.0×10^{44}



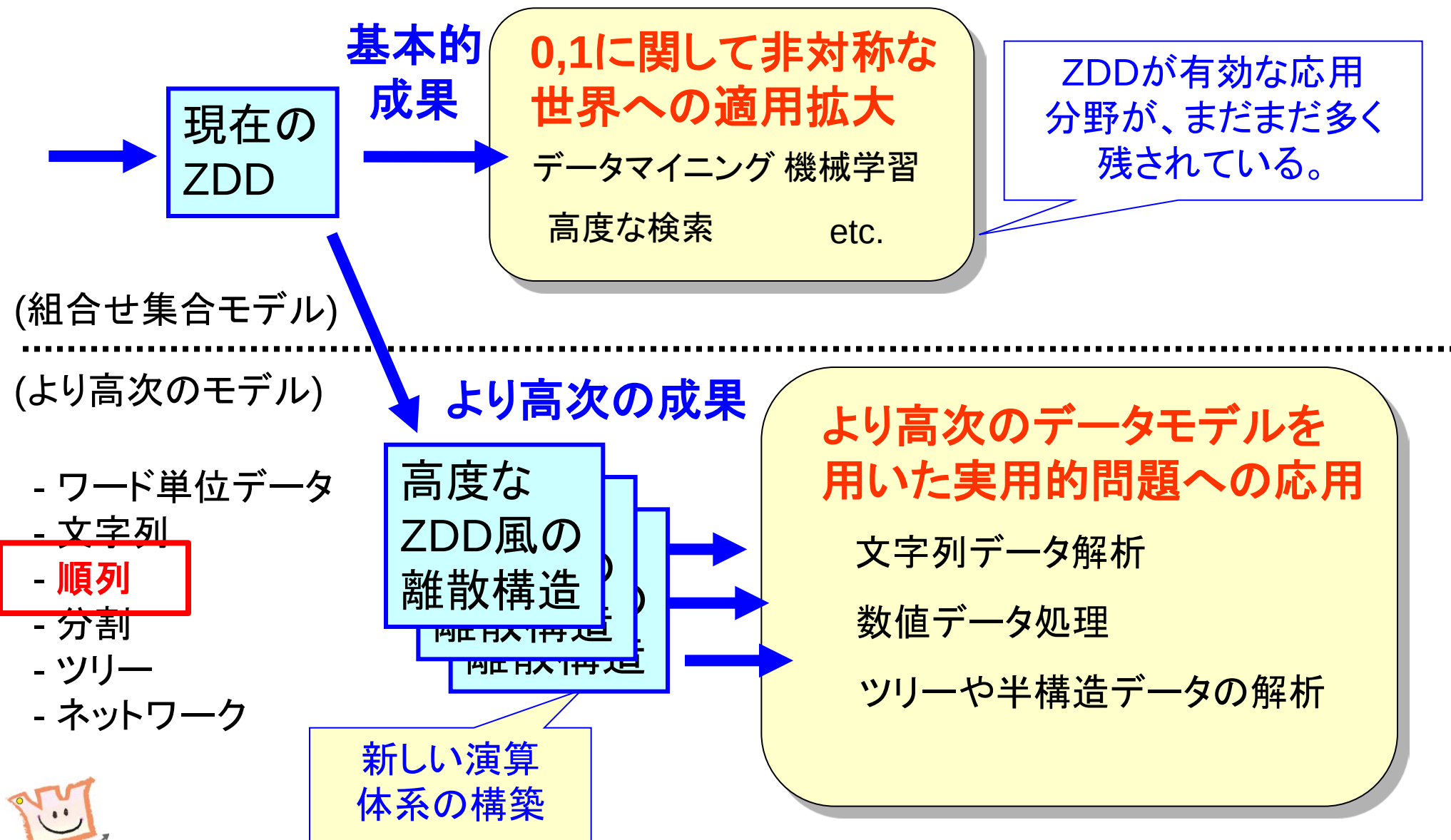
14797272518 通り

5039760385115189594214594926092397238616064 通り
(= 503正9760潤3851溝1518穰9594杼2145垓9492京6092兆3972億3861万6064)

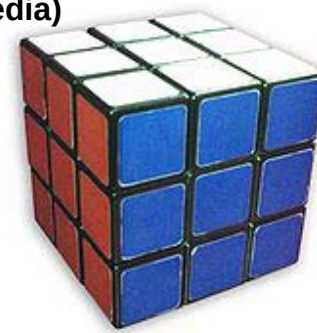
本研究プロジェクトの技術面のポイント



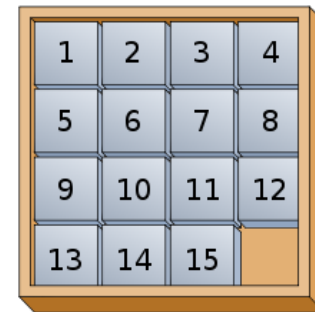
本研究プロジェクトの技術面のポイント



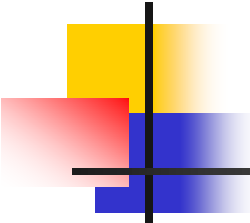
順列集合の処理



- ルービックキューブ:
 - $P = \{ \pi \mid \text{基本操作による順列} \}$
 - P は18通り (= 3通り × 6面) の順列の集合
 - 直積演算 $P \times P$ は、2回の基本操作で得られるパタンの集合を表す。
 - P^{20} で原理的には全ての可能なパタンの集合が得られる。
(実際は場合の数が多すぎてメモリあふれが起きる)
- 15パズル・トランプ等も順列集合で表現される
 - 最適な操作系列の解析
- あみだくじ・ソーティングネットワーク
 - 1対1のマッチング問題に適用可能
 - 順列は単射写像に対応。群論と深い関係がある。
- 情報無損失符号の設計・暗号理論への応用
 - Reversible logic設計問題への応用(量子計算とも関連)



(© Wikipedia)



Family of permutations

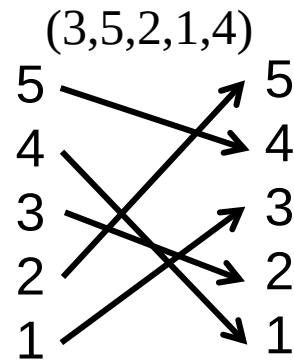
- Family of sets:
 - Don't consider order and duplication of items
 - “abcc” and “bca” are the same.
- Family of sequences:
 - Distinguishes all finite sequences.
 - φ , $\{\lambda\}$, $\{ab, aba, bbc\}$, $\{a, aa, aaa, aaaa\}$, etc.
- Family of permutations:
 - Set of orders in a fixed number of items.
 - φ , $\{123\}$, $\{12, 21\}$, $\{123456, 132456, 246135\}$

Decomposition of permutation

- Transpositions $\tau_{(x,y)}$: exchange of two items x and y .
- **Any n -item permutation π can be decomposed by at most $(n-1)$ transpositions.**

Let $x = \dim(\pi)$, then $\pi \tau_{(x, x\pi)}$ must not move x .
Thus, $\dim(\pi \tau_{(x, x\pi)}) \leq \dim(\pi) - 1$

→ Repeating this process provides a decomposed form.



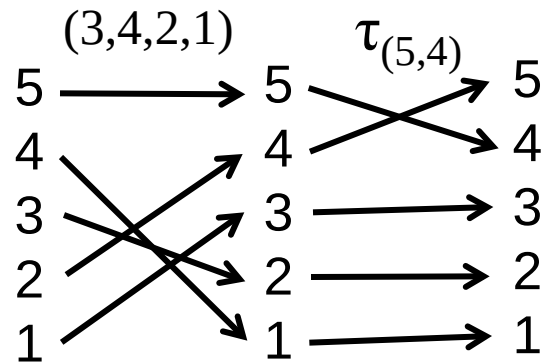
$$\pi = (3, 5, 2, 1, 4)$$

Decomposition of permutation

- Transpositions $\tau_{(x,y)}$: exchange of two items x and y .
- **Any n -item permutation π can be decomposed by at most $(n-1)$ transpositions.**

Let $x = \dim(\pi)$, then $\pi \tau_{(x, x\pi)}$ must not move x .
Thus, $\dim(\pi \tau_{(x, x\pi)}) \leq \dim(\pi) - 1$

→ Repeating this process provides a decomposed form.



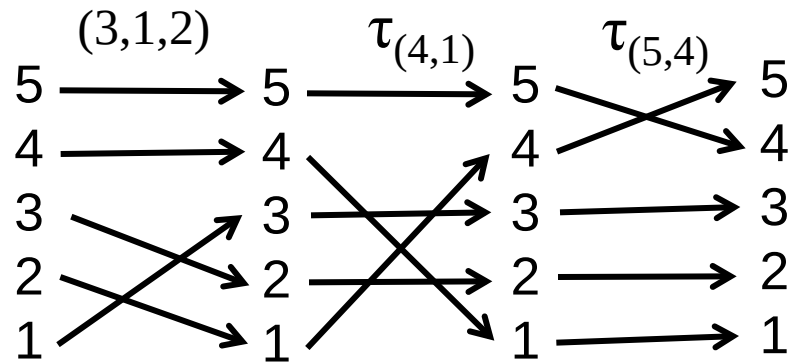
$$\pi = (3,5,2,1,4) = (3,4,2,1) \tau_{(5,4)}$$

Decomposition of permutation

- Transpositions $\tau_{(x,y)}$: exchange of two items x and y .
- **Any n -item permutation π can be decomposed by at most $(n-1)$ transpositions.**

Let $x = \dim(\pi)$, then $\pi \tau_{(x, x\pi)}$ must not move x .
 Thus, $\dim(\pi \tau_{(x, x\pi)}) \leq \dim(\pi) - 1$

→ Repeating this process provides a decomposed form.



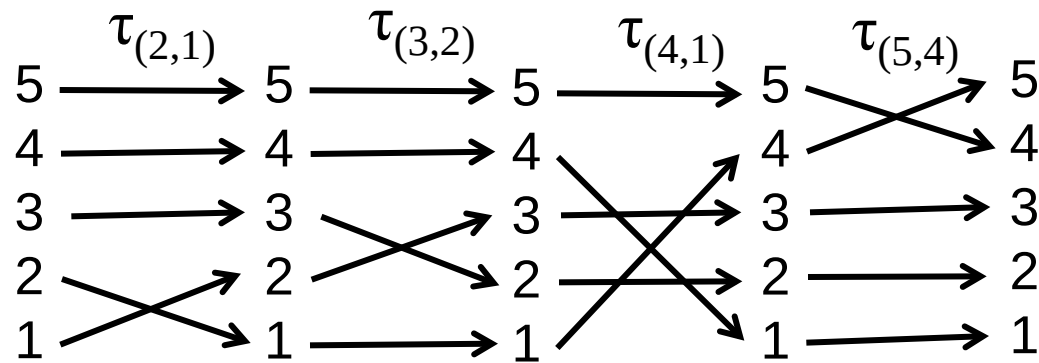
$$\pi = (3,5,2,1,4) = (3,1,2) \tau_{(4,1)} \tau_{(5,4)}$$

Decomposition of permutation

- Transpositions $\tau_{(x,y)}$: exchange of two items x and y .
- **Any n -item permutation π can be decomposed by at most $(n-1)$ transpositions.**

Let $x = \dim(\pi)$, then $\pi \tau_{(x, x\pi)}$ must not move x .
 Thus, $\dim(\pi \tau_{(x, x\pi)}) \leq \dim(\pi) - 1$

→ Repeating this process provides a decomposed form.

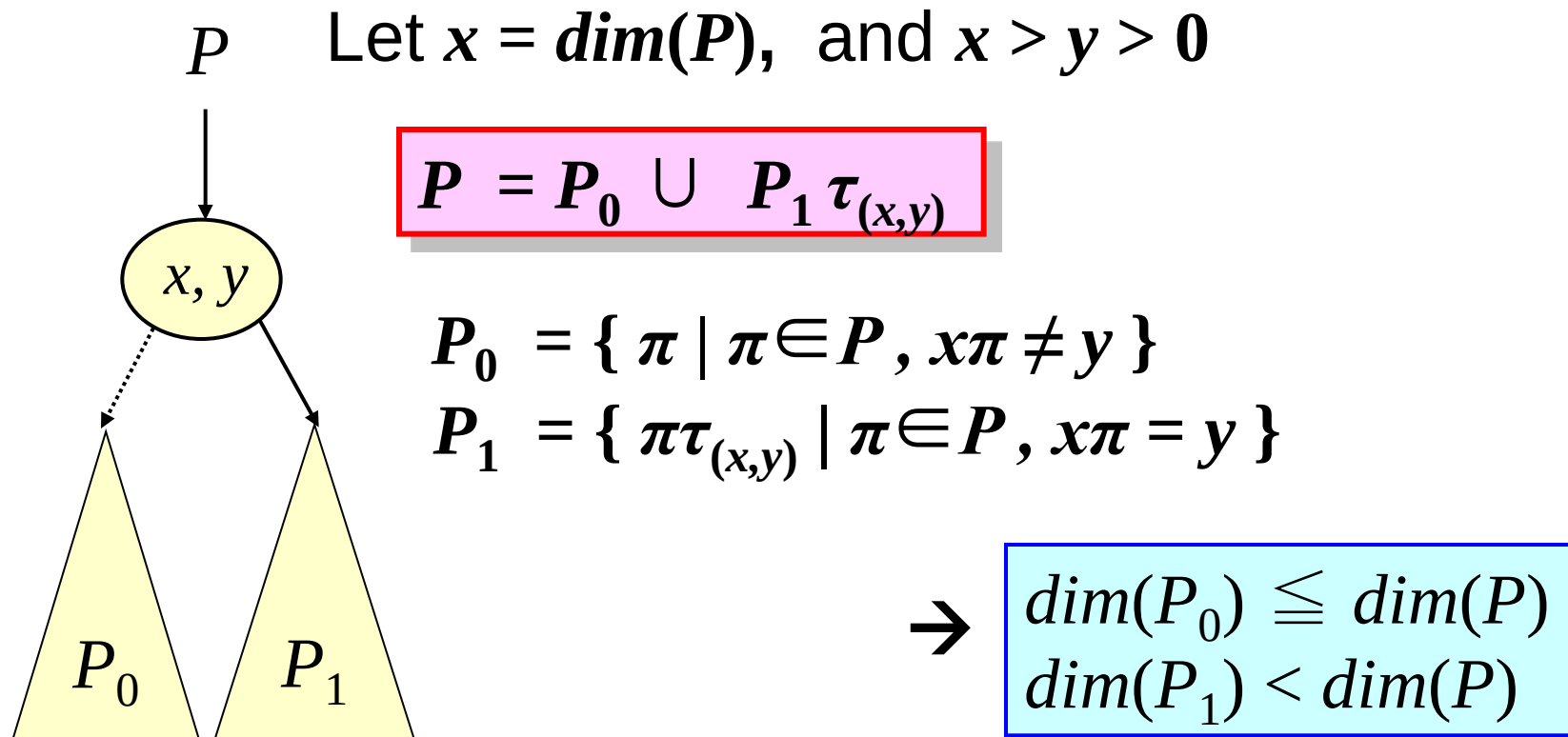


$$\pi = (3,5,2,1,4) = \tau_{(2,1)} \tau_{(3,2)} \tau_{(4,1)} \tau_{(5,4)}$$

Deterministic
 process.
 → canonical form
 for any given π .

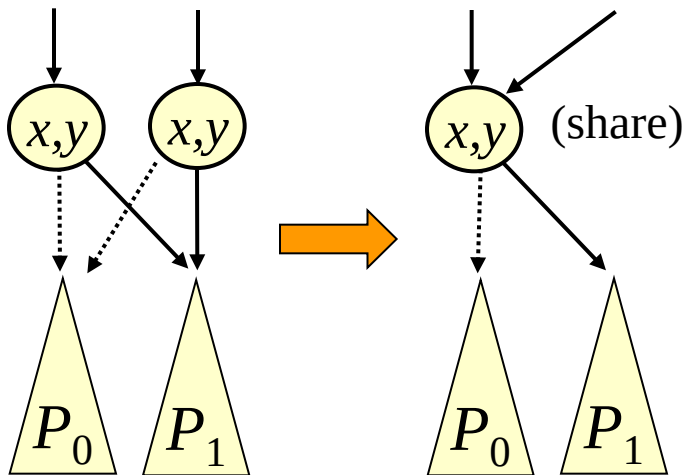
Main idea of π DDs

- Using a pair of IDs (x, y) for each decision node.

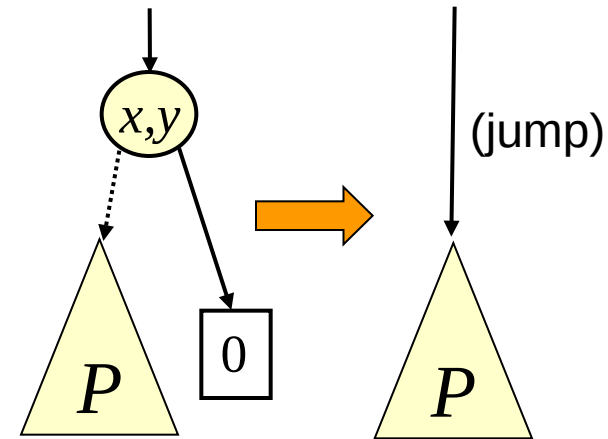


Node reduction rules for π DDs

- Same reduction rules as ZDDs.
 - Ordinary BDD rules don't work.

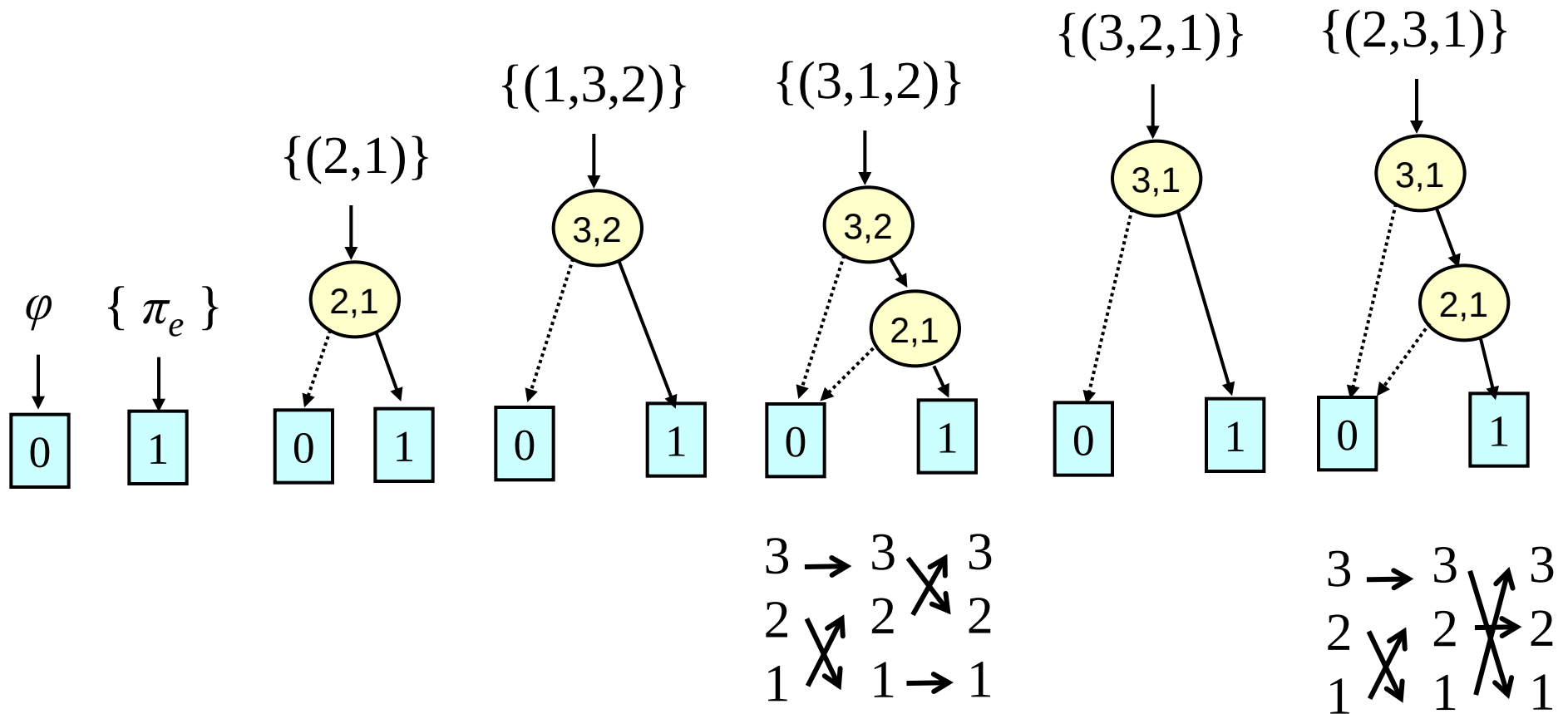


Node sharing

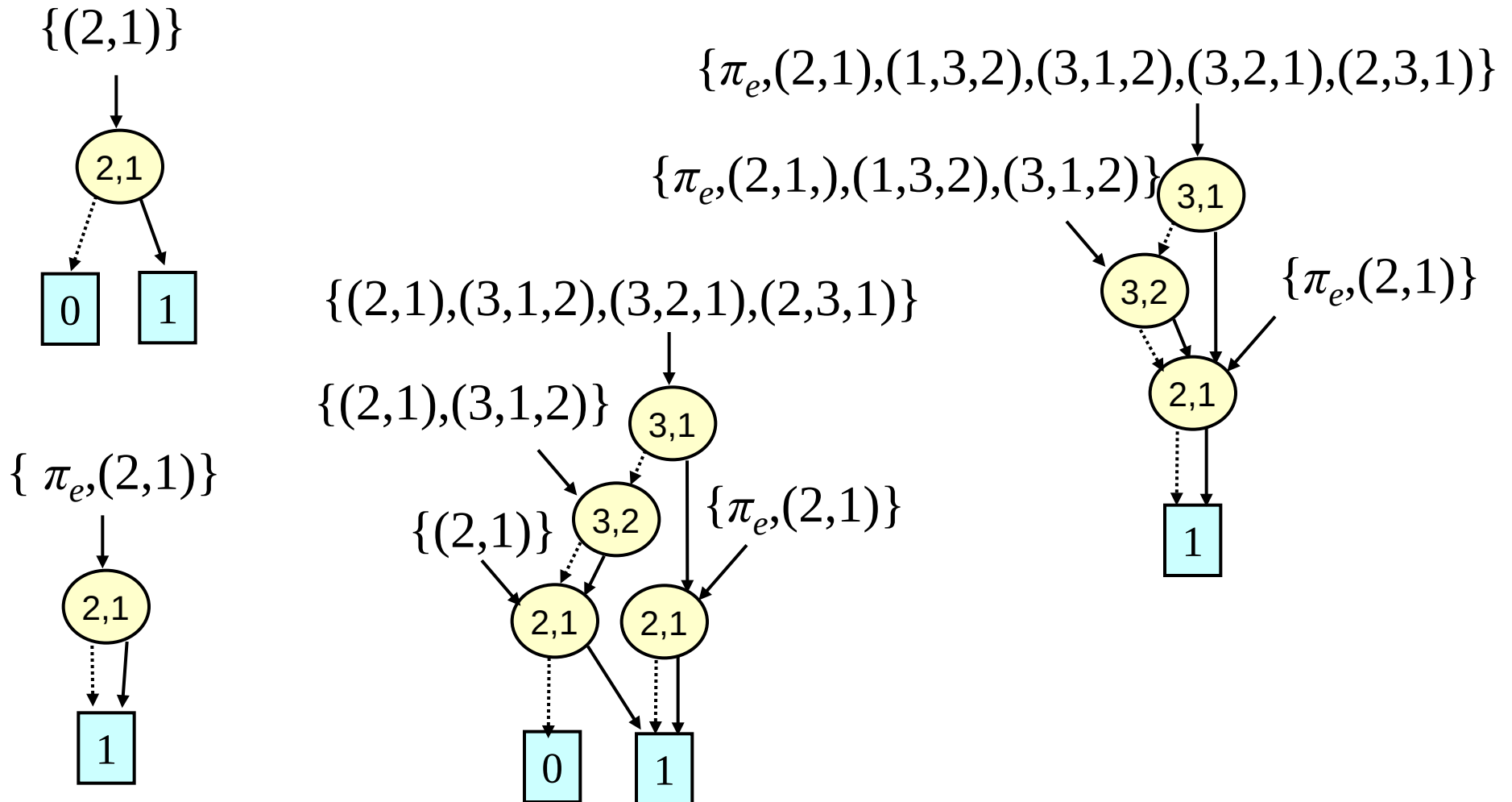


Zero-suppressed
node elimination

π DDs of single permutation



π DDs for sets of permutations

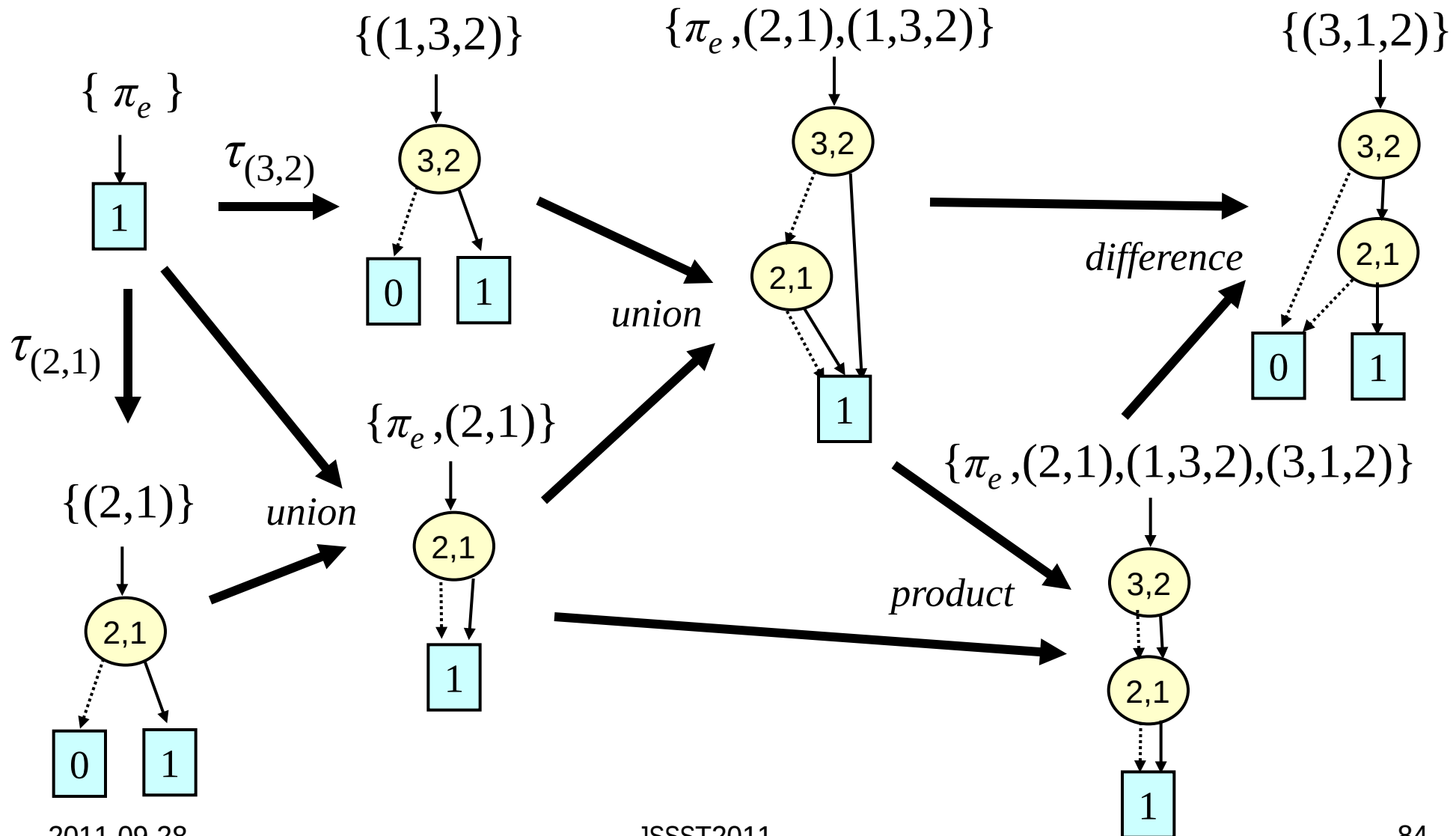


Algebraic operations for π DDs

■ “Permutation family algebra”

$\emptyset$	Returns the empty set. (0-terminal node)
$\{\pi_e\}$	Returns the singleton set. (1-terminal node)
$P.top$	Returns the IDs (x, y) at the root node of P .
$P \cup Q$	Returns $\{\pi \mid \pi \in P \text{ or } \pi \in Q\}$.
$P \cap Q$	Returns $\{\pi \mid \pi \in P, \pi \in Q\}$.
$P \setminus Q$	Returns $\{\pi \mid \pi \in P, \pi \notin Q\}$.
$P.\tau(x, y)$	Returns $P \cdot \tau_{(x, y)}$.
$P * Q$	Returns $\{\alpha\beta \mid \alpha \in P, \beta \in Q\}$.
$P.cofact(x, y)$	Returns $\{\pi\tau_{(x, y)} \mid \pi \in P, x\pi = y\}$.
$P.count$	Returns the number of permutations.

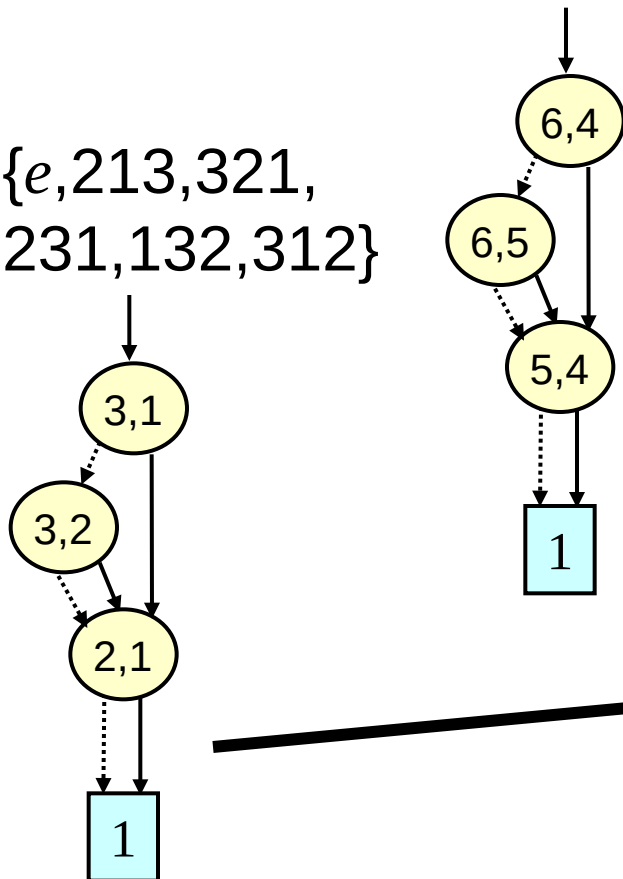
Synthesis of π DDs by algebraic operations



Product operation for disjoint permutations

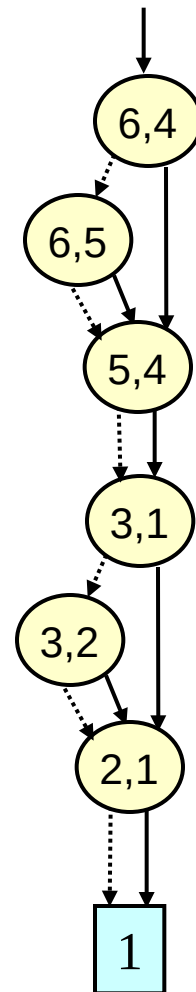
$\{e, 123546, 123654, 123564, 123465, 123645\}$

$\{e, 213, 321, 231, 132, 312\}$



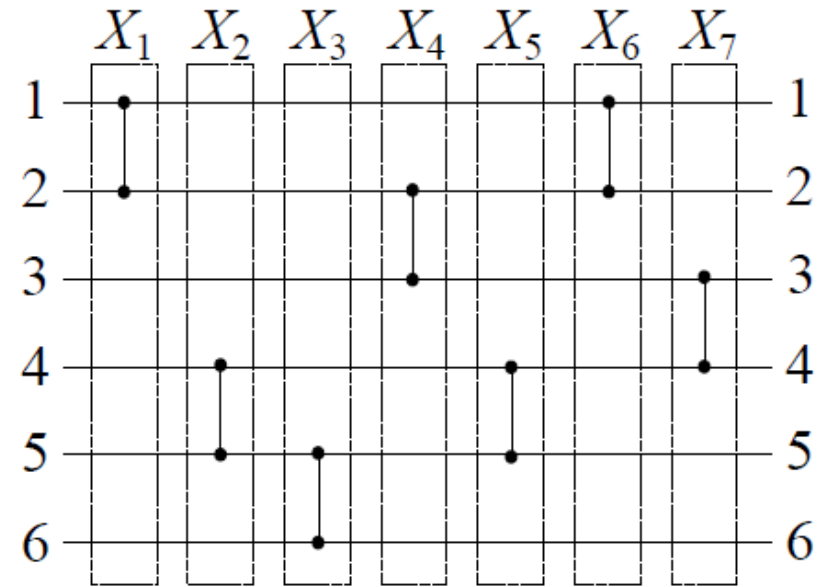
product

$\{e, 123546, 123654, 123564, 123465, 123645, 213456, 213546, 213654, 213564, 213465, 213645, 321456, 321546, 321654, 321564, 321465, 321645, 231456, 231546, 231654, 231564, 231465, 231645, 132456, 132546, 132654, 132564, 132465, 132645, 312456, 312546, 312654, 312564, 312465, 312645\}$



応用1: あみだくじの解析

- あみだくじの幅を n とし、 k 本の隣接互換の線 $X_1 \sim X_k$ を引くとする。
最低何本の隣接互換を使えば所望の順列を作り出すことができるか？
- k 本以下の線で作り出せる順列の集合は、以下の単純な漸化式で生成できる。→ π DDで計算できる。



$$P_0 = \pi_e$$

$$P_1 = P_0 \cup \left(\bigcup_{i=1}^{n-1} \tau_{(i,i+1)} \right)$$

$$P_k = P_{k-1} * P_1 \quad (\text{for } k \geq 2)$$

応用1:あみだくじの解析(実験結果1)

■ 幅10のあみだくじに関する実験結果

- 45本の線で10! 通りの全順列を生成できる。(44本までで (10! - 1) 通り)
- π DDのサイズは途中にピークがあり、最後はわずか45ノード

P_k	π DD size	# of perm.	total # τ	P_k	π DD size	# of perm.	total # τ	P_k	π DD size	# of perm.	total # τ
P_0	0	1	0	P_{16}	3956	528441	3412177	P_{32}	8655	3497165	24691907
P_1	9	10	9	P_{17}	4685	690778	4522462	P_{33}	7669	3544208	25039740
P_2	31	54	97	P_{18}	5455	878737	5821218	P_{34}	6590	3576891	25279788
P_3	63	209	546	P_{19}	6249	1089826	7296041	P_{35}	5470	3598561	25439624
P_4	109	649	2152	P_{20}	7047	1319957	8915085	P_{36}	4374	3612201	25539440
P_5	172	1717	6704	P_{21}	7834	1563651	10645703	P_{37}	3353	3620296	25598543
P_6	261	4015	17632	P_{22}	8591	1814400	12433871	P_{38}	2444	3624785	25630975
P_7	390	8504	40751	P_{23}	9293	2065149	14239194	P_{39}	1671	3627083	25647411
P_8	558	16599	84985	P_{24}	9905	2308843	15996836	P_{40}	1055	3628151	25654943
P_9	773	30239	162995	P_{25}	10397	2538974	17671711	P_{41}	602	3628591	25657983
P_{10}	1034	51909	291537	P_{26}	10735	2750063	19206325	P_{42}	305	3628746	25659023
P_{11}	1353	84592	491272	P_{27}	10894	2938022	20584666	P_{43}	136	3628790	25659303
P_{12}	1727	131635	786100	P_{28}	10857	3100359	21772380	P_{44}	59	3628799	25659355
P_{13}	2169	196524	1201963	P_{29}	10614	3236212	22773147	P_{45}	45	3628800	25659360
P_{14}	2688	282578	1764353	P_{30}	10157	3346222	23579581	P_{46}	45	3628800	25659360
P_{15}	3286	392588	2495497	P_{31}	9497	3432276	24214975				

応用1: あみだくじの解析(実験結果2)

- 幅 n のあみだくじに関する実験結果
 - 全順列を生成するための最小本数 m を調べた。
 - π DDサイズ(ピーク時と最終)を見ると、データ圧縮率は非常に大きい。

n	m	π DD size		# of perm.	total # τ	time (sec)
		(peak)	(final)			
1	0	0	0	1	0	0.00
2	1	1	1	2	1	0.00
3	3	3	3	6	7	0.00
4	6	9	6	24	46	0.00
5	10	27	10	120	326	0.00
6	15	89	15	720	2556	0.01
7	21	292	21	5040	22212	0.02
8	28	972	28	40320	212976	0.06
9	36	3241	36	362880	2239344	0.26
10	45	10894	45	3628800	25659360	1.19
11	55	36906	55	39916800	318540960	5.77
12	66	125904	66	479001600	4261576320	27.06
13	78	435221	78	6227020800	61148511360	126.80
14	91	1520439	91	87178291200	937030429440	666.29

応用2:ルービックキューブの解析

- 今回は8個(24面)の角キューブの移動だけに着目
 - 角キューブを合わせた後、角キューブに触らずに他のキューブを移動可能。
- X,Y,Z軸に関する90度の回転操作は、以下の式で書ける。

$$\pi_x = T(3,5)T(3,17)T(3,15)T(1,6)T(1,16)T(1,14)T(2,4)T(2,18)T(2,13)$$

$$\pi_y = T(2,14)T(2,24)T(2,12)T(3,13)T(3,23)T(3,10)T(1,15)T(1,22)T(1,11)$$

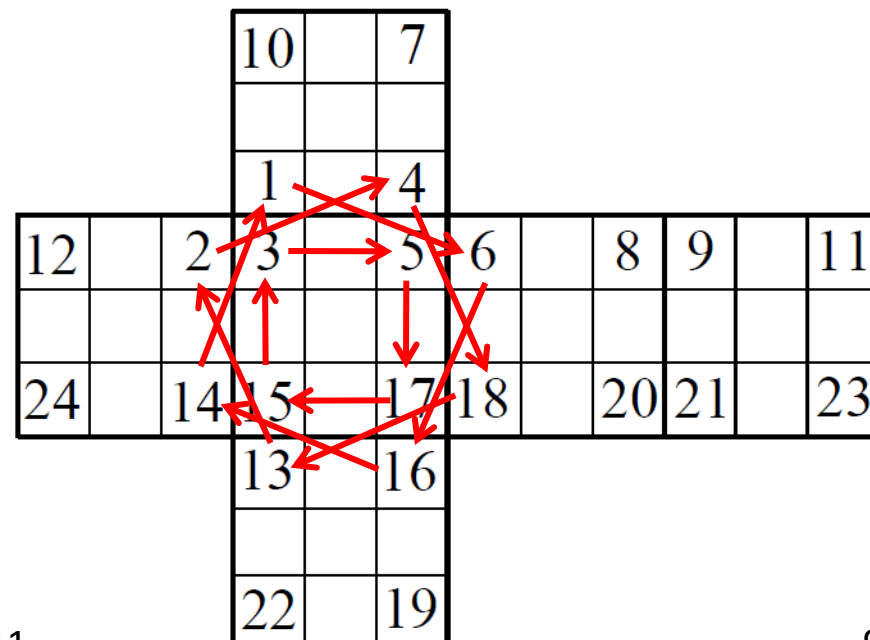
$$\pi_z = T(1,10)T(1,7)T(1,4)T(3,12)T(3,9)T(3,6)T(2,11)T(2,8)T(2,5)$$

			10		7						
			1		4						
12		2	3		5	6		8	9		11
24		14	15		17	18		20	21		23
			13		16						
			22		19						

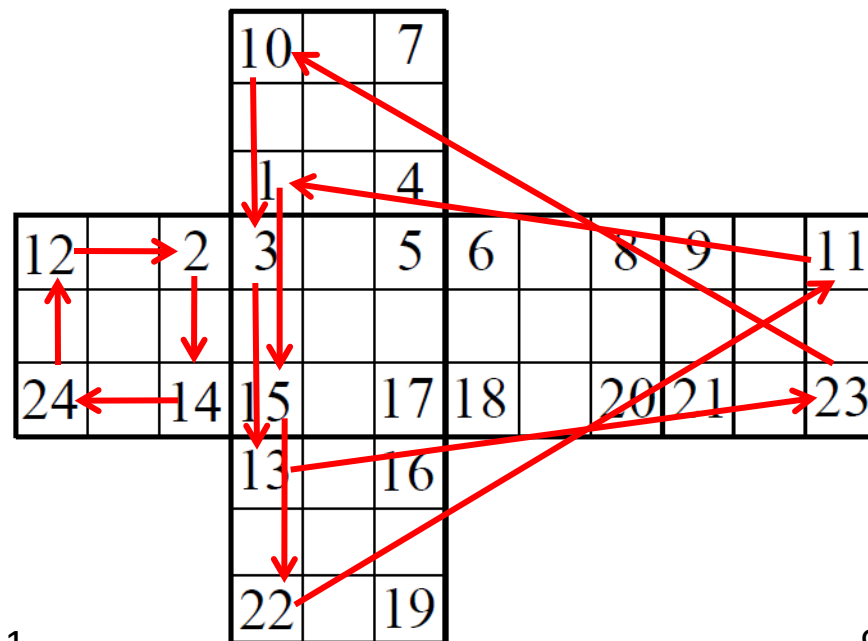
- $$\pi_x = \tau_{(3,5)}\tau_{(3,17)}\tau_{(3,15)}\tau_{(1,6)}\tau_{(1,16)}\tau_{(1,14)}\tau_{(2,4)}\tau_{(2,18)}\tau_{(2,13)}$$

$$\pi_y = \tau_{(2,14)}\tau_{(2,24)}\tau_{(2,12)}\tau_{(3,13)}\tau_{(3,23)}\tau_{(3,10)}\tau_{(1,15)}\tau_{(1,22)}\tau_{(1,11)}$$

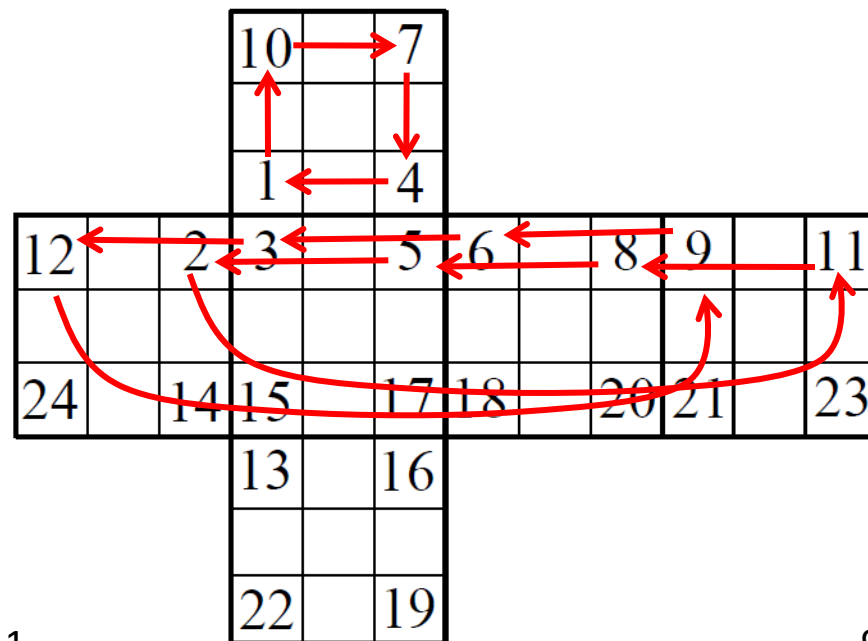
$$\pi_z = \tau_{(1,10)}\tau_{(1,7)}\tau_{(1,4)}\tau_{(3,12)}\tau_{(3,9)}\tau_{(3,6)}\tau_{(2,11)}\tau_{(2,8)}\tau_{(2,5)}$$



- $$\begin{aligned} \pi_x &= \tau_{(3,5)}\tau_{(3,17)}\tau_{(3,15)}\tau_{(1,6)}\tau_{(1,16)}\tau_{(1,14)}\tau_{(2,4)}\tau_{(2,18)}\tau_{(2,13)} \\ \pi_y &= \tau_{(2,14)}\tau_{(2,24)}\tau_{(2,12)}\tau_{(3,13)}\tau_{(3,23)}\tau_{(3,10)}\tau_{(1,15)}\tau_{(1,22)}\tau_{(1,11)} \\ \pi_z &= \tau_{(1,10)}\tau_{(1,7)}\tau_{(1,4)}\tau_{(3,12)}\tau_{(3,9)}\tau_{(3,6)}\tau_{(2,11)}\tau_{(2,8)}\tau_{(2,5)} \end{aligned}$$



- $$\begin{aligned}\pi_x &= \tau_{(3,5)}\tau_{(3,17)}\tau_{(3,15)}\tau_{(1,6)}\tau_{(1,16)}\tau_{(1,14)}\tau_{(2,4)}\tau_{(2,18)}\tau_{(2,13)} \\ \pi_y &= \tau_{(2,14)}\tau_{(2,24)}\tau_{(2,12)}\tau_{(3,13)}\tau_{(3,23)}\tau_{(3,10)}\tau_{(1,15)}\tau_{(1,22)}\tau_{(1,11)} \\ \pi_z &= \tau_{(1,10)}\tau_{(1,7)}\tau_{(1,4)}\tau_{(3,12)}\tau_{(3,9)}\tau_{(3,6)}\tau_{(2,11)}\tau_{(2,8)}\tau_{(2,5)}\end{aligned}$$



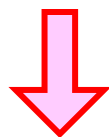
応用2:ルービックキューブの解析

- 今回は8個(24面)の角キューブの移動だけに着目
 - 角キューブを合わせた後、角キューブに触らずに他のキューブを移動可能。
- X,Y,Z軸に関する90度の回転操作は、以下の式で書ける。

$$\pi_x = \tau(3,5)\tau(3,17)\tau(3,15)\tau(1,6)\tau(1,16)\tau(1,14)\tau(2,4)\tau(2,18)\tau(2,13)$$

$$\pi_y = \tau(2,14)\tau(2,24)\tau(2,12)\tau(3,13)\tau(3,23)\tau(3,10)\tau(1,15)\tau(1,22)\tau(1,11)$$

$$\pi_z = \tau(1,10)\tau(1,7)\tau(1,4)\tau(3,12)\tau(3,9)\tau(3,6)\tau(2,11)\tau(2,8)\tau(2,5)$$



1手以内で作れるパタンの集合

$$P_1 = \pi_e + \pi_x + \pi_x^2 + \pi_x^3 \\ + \pi_y + \pi_y^2 + \pi_y^3 + \pi_z + \pi_z^2 + \pi_z^3$$

			10		7						
			1		4						
12		2	3		5	6		8	9		11
24		14	15		17	18		20	21		23
			13		16						
			22		19						

応用2:ルービックキューブの解析

- 今回は8個(24面)の角キューブの移動だけに着目
 - 角キューブを合わせた後、角キューブに触らずに他のキューブを移動可能。
- X,Y,Z軸に関する90度の回転操作は、以下の式で書ける。

$$\pi_x = \tau(3,5)\tau(3,17)\tau(3,15)\tau(1,6)\tau(1,16)\tau(1,14)\tau(2,4)\tau(2,18)\tau(2,13)$$

$$\pi_y = \tau(2,14)\tau(2,24)\tau(2,12)\tau(3,13)\tau(3,23)\tau(3,10)\tau(1,15)\tau(1,22)\tau(1,11)$$

$$\pi_z = \tau(1,10)\tau(1,7)\tau(1,4)\tau(3,12)\tau(3,9)\tau(3,6)\tau(2,11)\tau(2,8)\tau(2,5)$$

↓ 1手以内で作れるパタンの集合

$$P_1 = \pi_e + \pi_x + \pi_x^2 + \pi_x^3 \\ + \pi_y + \pi_y^2 + \pi_y^3 + \pi_z + \pi_z^2 + \pi_z^3$$

k手以内で作れるパタンの集合

$$P_k = P_{k-1} * P_1 \quad (\text{for } k \geq 2)$$

			10		7						
			1		4						
12		2	3		5	6		8	9		11
24		14	15		17	18		20	21		23
			13		16						
			22		19						

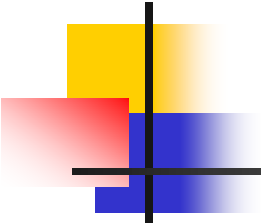
応用2:ルービックキューブの解析(実験結果)

- 角キューブだけに着目すると、11手以内で全てのパタンを生成可能
 - 11手以内で必ず初期状態に戻せる。
 - 全パタンを表す π DDはわずか511ノード
- P_k を作ってしまうえば、あるパタンを作り出すのに最低何手必要かを容易に調べることができる。
 - 例えば8個の角キューブのうち、2個だけが移動して、残り6個は移動しないパタンは以下の式で書ける。

$$S_k = P_k \cdot \text{cofact}(9, 9) \cdot \text{cofact}(11, 11) \cdot \text{cofact}(15, 15) \\ \cdot \text{cofact}(17, 17) \cdot \text{cofact}(21, 21) \cdot \text{cofact}(23, 23)$$

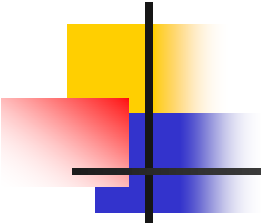
→ 実際に計算すると、 $k=10$ で4通り、 $k=11$ で5通り存在。
(その生成手順も、 π DDの演算で容易に見つけられる。)

P_k	π DD size	# of perm.	total # τ
P_0	0	1	0
P_1	63	10	72
P_2	392	64	888
P_3	1789	385	5634
P_4	6860	2232	34446
P_5	23797	12224	194406
P_6	84704	62360	1012170
P_7	290018	289896	4752582
P_8	608666	1159968	19087266
P_9	580574	3047716	50272542
P_{10}	18783	3671516	60540732
P_{11}	511	3674160	60579900
P_{12}	511	3674160	60579900



ルービックキューブ解析についての考察

- 最近、 3×3 の全パタンについて20手以内で必ず初期状態に戻せることが示されている。[Rokicki et al. 2010]
 - 全てのパタンについての最小手数を列挙したわけではない。
最小手数が20を超えるパタンがないことを列挙して確認
 - 最初に数学的な枝刈りを行い、最後に大規模並列PC(クラウド)を用いて、全列挙(総CPU時間35年分)
- 我々の手法を角キューブに限らない 3×3 のルービックキューブにそのまま適用すると、 π DDが大きくなり過ぎてメモリあふれを起こす。
 - 全てのパタンについて最小手数を列挙しているので、上記の計算よりも細かい計算を行っている。
 - 本気で枝刈りを行わず、 π DDの圧縮能力だけを使って計算している。
 - 1CPUだけを用いている。
 - パズルの条件が変わっても、数式を修正するだけで柔軟に対応可能
(1手の定義、使える操作の制約、人間が覚えやすい解法の探索、etc.)



Upper bound of π DD sizes

- Number of Families of permutations up to n items: $2^{n!}$

- n 0, 1, 2, 3, 4, 5, ...

- $n!$ 1, 1, 2, 6, 24, 120, ...

- $2^{n!}$ 2, 2, 4, 64, 16777216,
 1329227995784915872903807060280344576, ...

- At least $\log n$ bit needed to distinguish n objects.
 - Thus, π DD size must be $O(n!)$ bit in worst cases.

まとめ

- VLSI設計分野から始まったBDD/ZDD技術は、近年、応用範囲が大きく広がった。
 - 2000年以降の計算機の性能向上(特にメモリ容量)が大きい。
 - データベースやAI分野では、多くの場合ZDDが有効。
 - Knuth本に掲載 → 離散構造処理系の基盤技術として再評価
- ERATOプロジェクト
 - アルゴリズム研究は特殊な実験装置はなく、「人」と「場所」が全て。
 - 西尾章治郎先生(審査委員長)からは「札幌を、優秀な人材が集まる梁山泊のようにして欲しい」との激励を頂いている。
 - 東京・関西のサブグループ拠点と活発な研究交流を行いながら、優秀な若手人材を見出し、札幌に引っ張ってきて、高いレベルの「梁山泊」を形成し、テーマを絞って一定期間じっくりと研究を行う。
 - 1年半が経過し、期待以上の相乗効果が得られ始めている。
 - コアとなる「Art」を形成し、学界・産業界に還流させていきたい。

