MTZDD によるフィルタリングルールに合致するパケット数の算出

原田 崇司 [1]　田中 賢 [1]　三河 賢治 [2]

[1] 神奈川大学大学院 理学研究科 理学専攻 情報科学領域

[2] 新潟大学学術情報機構情報基盤センター

2017 年 6 月 19 日

CAS・SIP・MSS・VLD, 新潟大学

本日の内容

研究背景

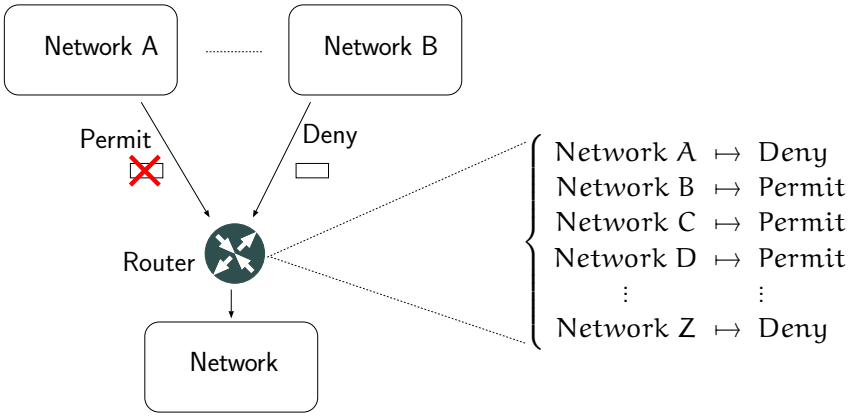
MTZDD

提案手法

計算機実験

まとめと今後の課題

パケットフィルタリング



ポリシに従ってパケットをフィルタリング

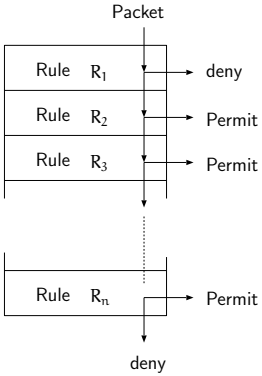
パケットフィルタリング

ポリシ　：　プログラムにおける仕様のようなもの
ルールリスト　：　プログラムにおける実装のようなもの

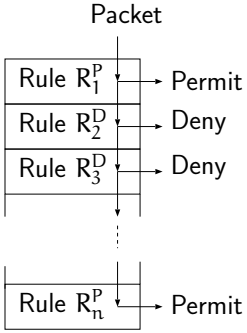
ポリシを満たすルールリストを作成

ルールリストを線型探索してフィルタリング
（$R_1, R_2, \ldots, R_n$ の順でパケットと照合）

ルール数が多いと遅延が発生
ルール数は数百 ~ 数千



フィルタリングルールの形式



フィルタリングモデル

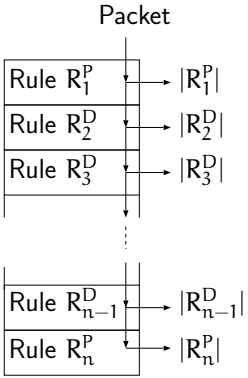
Filter $\mathbf{R}$
$R_1^P = * \; 0 \; * \; 1$
$R_2^P = 0 \; 0 \; 0 \; 0$
$R_3^P = 0 \; * \; 0 \; 0$
$R_4^D = 0 \; * \; 1 \; *$
$R_5^P = * \; 1 \; * \; 1$
$R_6^P = * \; * \; * \; 1$
$R_7^D = * \; * \; * \; *$

ルールリストの例

ルールの形式：$R_i^e = b_1 b_2 \ldots b_w,\ b_k \in \{0, 1, *\},\ e \in \{P, D\},\ i \in \mathbb{N}$

ルール番号 i はただの識別子，ルールリスト中の位置を表してはいない

ネットワーク機器の遅延



i 番目のルールによって評価型が決まるパケットは i 回の照合を受ける

1 回の照合を遅延 1 と考え，
ネットワーク機器の遅延を定義

$$L(\mathbf{R}, F) = \sum_{i}^{n-1} i|R_i^e(F)| + (n-1)|R_n^e(F)|$$

F はパケットの頻度分布

$|R_i^e(F)|$ は $\mathbf{R}$ において R_i^e によって評価型が決まるパケットの数，
即ち，$|R_i^e(F)|$ はルールリスト $\mathbf{R}$，頻度分布 F，ルールの位置 i をもらう函数

$|R_i^e(F)|$ は R_i^e に合致するパケットの数ではないことに注意

評価パケット数

評価パケット数 $|R_i^e(F)|$

分布 F においてルールリスト $\mathbf{R}$ の $1 \sim (\sigma(i) - 1)$ 番目のルールに合致せず，R_i^e に合致するパケットの数

Filter $\mathbf{R}$	$\lvert R_i(F)\rvert$
$R_1^P = * \; 0 \; * \; 1$	4
$R_2^P = 0 \; 0 \; 0 \; 0$	1
$R_3^P = 0 \; * \; 0 \; 0$	1
$R_4^D = 0 \; * \; 1 \; *$	3
$R_5^P = * \; 1 \; * \; 1$	3
$R_6^P = * \; * \; * \; 1$	0
$R_7^D = * \; * \; * \; *$	4
$L(\mathbf{R}, F) = 60$	

評価パケット数の例：

F を一様とすると R_4^D に合致するパケットは $\{0010, 0011, 0110, 0111\}$．このうち，0011 は R_1^P によって評価型が決まるので，R_4^D によって評価型が決まるパケットは $\{0010, 0110, 0111\}$ の 3 つ．

以上より $|R_4^D| = 3$

ルールリストとポリシー

Filter $\mathbf{R}$	$\lvert R_i(F)\rvert$
$R_1^P = * 0 * 1$	4
$R_2^P = 0 0 0 0$	1
$R_3^P = 0 * 0 0$	1
$R_4^D = 0 * 1 *$	3
$R_5^P = * 1 * 1$	3
$R_6^P = * * * 1$	0
$R_7^D = * * * *$	4
$L(\mathbf{R}, F) = 60$	

Filter $\mathbf{R}'$	$\lvert R_i(F)\rvert$
$R_1^P = * 0 * 1$	4
$R_4^D = 0 * 1 *$	3
$R_5^P = * 1 * 1$	3
$R_3^P = 0 * 0 0$	2
$R_2^P = 0 0 0 0$	0
$R_6^P = * * * 1$	0
$R_7^D = * * * *$	4
$L(\mathbf{R}', F) = 51$	

$0000 \mapsto P, 0001 \mapsto P, 0010 \mapsto D, 0011 \mapsto P, 0100 \mapsto P, 0101 \mapsto P, 0110 \mapsto D, 0111 \mapsto D,$
$1000 \mapsto D, 1001 \mapsto P, 1010 \mapsto D, 1011 \mapsto P, 1100 \mapsto D, 1101 \mapsto P, 1110 \mapsto D, 1111 \mapsto P$

フィルタ $\mathbf{R}$ と $\mathbf{R}'$ は同じポリシー（上記のポリシー）を表す
⇨ ルールを並び替えると遅延 $L(\mathbf{R}, F)$ を減らせる可能性有り

ルール順序最適化問題

ルール順序最適化問題

入力： ルールリスト $\mathbf{R}$，頻度分布 F

出力： $\sum_{i=1}^{n-1} \sigma(i) \times |R_i(F)| + (n-1) \times |R_n|$ を最小化する $\mathbf{R}$ と同一の函数を表す（ポリシーを維持する）ルールの順序 σ

ただし，$\forall\sigma\ \sigma(n) = n$（最後のルール R_n^e は並び替えない）とする．

ルール順序最適化問題は $\mathcal{NP}$ 困難 (Hamed,2006)

ただ，このルール順序最適化問題 (Hamed,2006) は...

評価パケット数の変動

Filter $\mathbf{R}$	$\lvert R_i(F)\rvert$
$R_1^P = * \; 0 \; * \; 1$	4
$R_2^P = 0 \; 0 \; 0 \; 0$	1
$R_3^P = 0 \; * \; 0 \; 0$	1
$R_4^D = 0 \; * \; 1 \; *$	3
$R_5^P = * \; 1 \; * \; 1$	3
$R_6^P = * \; * \; * \; 1$	0
$R_7^D = * \; * \; * \; *$	4
$L(\mathbf{R}, F) = 60$	

Filter $\mathbf{R}'$	$\lvert R_i(F)\rvert$
$R_1^P = * \; 0 \; * \; 1$	4
$R_4^D = 0 \; * \; 1 \; *$	3
$R_5^P = * \; 1 \; * \; 1$	3
$R_3^P = 0 \; * \; 0 \; 0$	2
$R_2^P = 0 \; 0 \; 0 \; 0$	0
$R_6^P = * \; * \; * \; 1$	0
$R_7^D = * \; * \; * \; *$	4
$L(\mathbf{R}', F) = 51$	

ルールの順序 σ により，評価パケット数が変動

評価パケット数算出問題の計算複雑さ

評価パケット数算出問題

入力： ルールリスト $\mathbf{R}$，頻度分布 F，順序 σ，ルール番号 i
出力： 頻度分布 F のもとで $1 \sim \sigma(i-1)$ 番目のルールに合致せず，R_i に合致するパケットの数

評価パケット数算出問題は $\#\mathcal{P}$-complete（証明は省略）

研究目的

ルール順序最適化問題に対する従来の研究は評価パケット数の変動を無視
（そもそもルールの評価パケット数を頻度分布から求めていない）

ルールリスト（ルールの順序）を精確に評価できない

評価パケット数算出法が必要

頻度分布が一様での評価パケット数算出問題は $\#\mathcal{P}$-complete

頻度分布が一様での MTZDD による評価パケット数算出法を提案

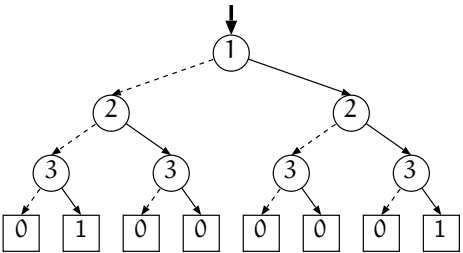
BDD/ZDD

BDD (Binary Decision Diagram)

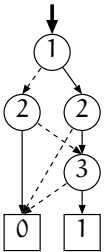
論理函数を効率よく扱えるデータ構造

ZDD (Zero-Suppressed Binary Decision Diagram)

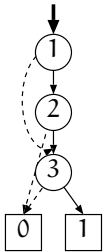
疎な組合せ集合を効率よく扱えるデータ構造



場合分け二分木



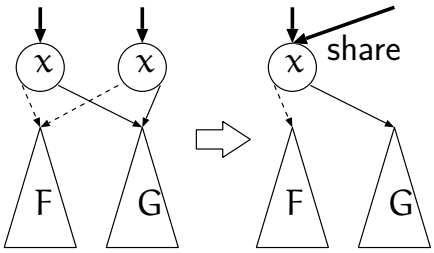
BDD



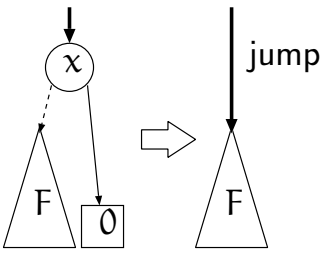
ZDD

ZDD の節点削除規則

論理函数（組合せ集合）に対する場合分け二分木に対して，以下の節点削除規則を既約になるまで適用すれば ZDD の出来上がり



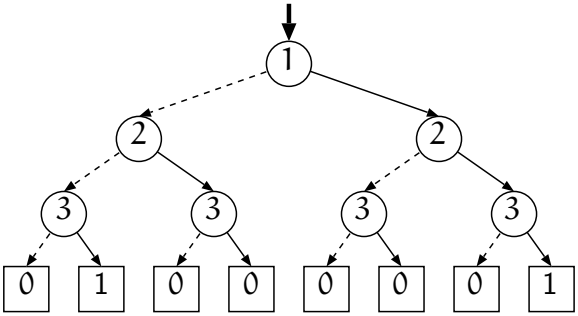
等価節点の共有



冗長節点の削除

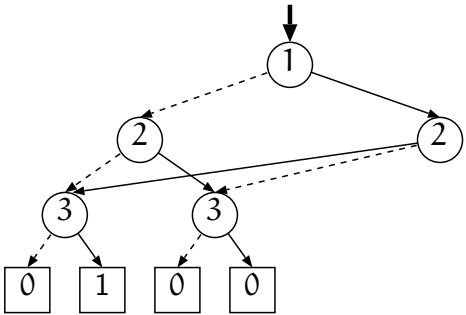
場合分け二分木を構築してから簡約することによって ZDD を構築するのは遅すぎるので，ZDD を高速に構築するための技法が数多存在

場合分け二分木から ZDD 構築の例



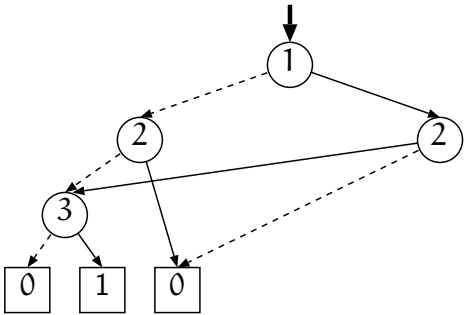
場合分け二分木（初期状態）

場合分け二分木から ZDD 構築の例



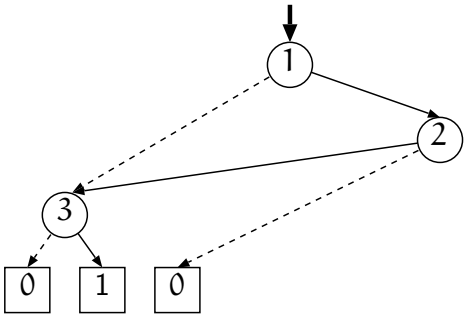
等価節点（0 枝と 1 枝の先が同じ節点 u と v）の共有

場合分け二分木から ZDD 構築の例



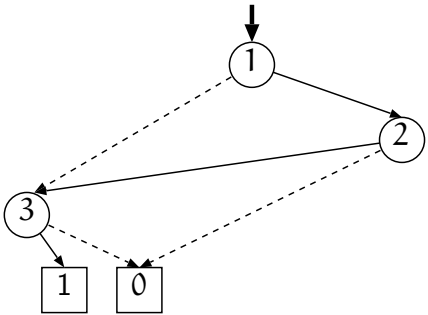
冗長節点（1 枝の先が 0 終端節点を指す節点）の削除

場合分け二分木から ZDD 構築の例



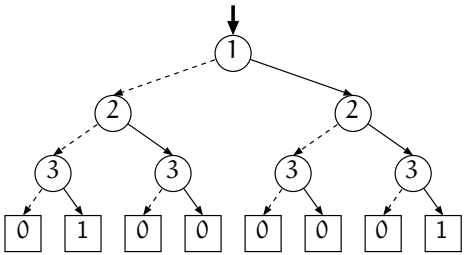
冗長節点（1 枝の先が 0 終端節点を指す節点）の削除

場合分け二分木から ZDD 構築の例

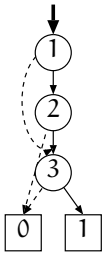


等価節点（0 枝と 1 枝の先が同じ節点 u と v）の共有

ZDD の読み方



場合分け二分木



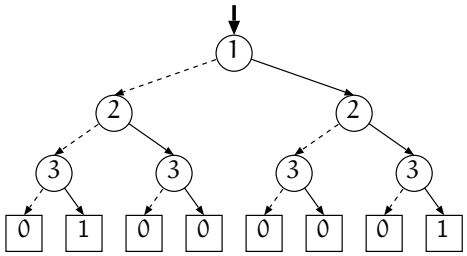
ZDD

根節点から終端節点 $\boxed{0}$, $\boxed{1}$ に向かって辿る

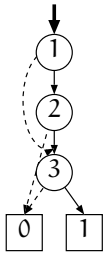
- 変数 $\textcircled{i}$ から 0 枝 (破線) を辿れば $x_i = 0$，1 枝 (実線) を辿れば $x_i = 1$
- 変数 $\textcircled{i}$ から 0 枝を辿った先が変数 $\textcircled{j}$ ならば，$x_{i+1}, x_{i+2}, \ldots, x_{j-1} = 0$

例えば，$\textcircled{1}$ $\dashrightarrow$ $\textcircled{3}$ $\longrightarrow$ $\boxed{1}$ は，$f(x_1 = 0, x_2 = 0, x_3 = 1) = 1$ を意味

ZDD の性質（提案手法に有用な）



場合分け二分木



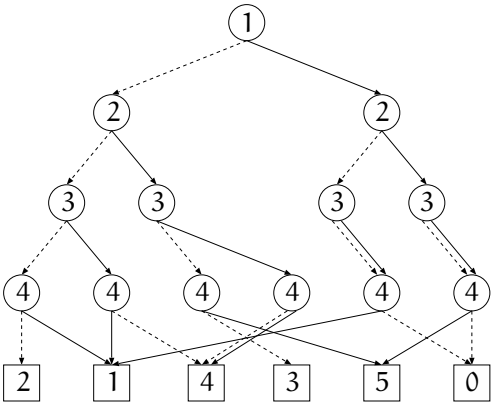
ZDD

根から $\boxed{1}$ 節点へのパスの本数が，函数を 1 とする割り当ての個数

例えば，上の ZDD に対応する函数を 1 にする割り当ての個数は，根から $\boxed{1}$ 節点へのパスの本数が 2 本なので，2 個.
ちなみに割り当ては，$(0,0,1)$ と $(1,1,1)$

Multi-Terminal Zero-suppressed Binary Decision Diagram

ZDD において終端節点を $\boxed{0}$, $\boxed{1}$ の二値から $\boxed{0}$, $\boxed{1}$, ..., $\boxed{k}$ の多値へと増やしたものが MTZDD



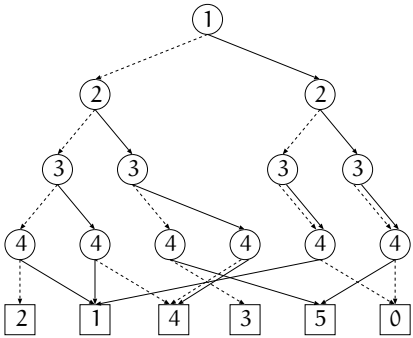
ZDD と同様に，場合分け二分木を構成して節点削除規則を適用すれば構築可能

MTZDD の読み方は ZDD と同じ

ZDD はブール函数，
MTZDD は $\{0,1\}^n \to \{0,1,\ldots,k\}$ という函数に対応するグラフ

提案手法

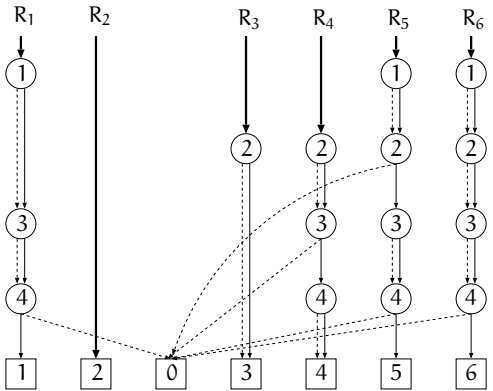
Filter $\mathbf{R}$
$R_1^P = * \; 0 \; * \; 1$
$R_2^P = 0 \; 0 \; 0 \; 0$
$R_3^D = 0 \; * \; 0 \; 0$
$R_4^D = 0 \; * \; 1 \; *$
$R_5^P = * \; 1 \; * \; 1$
$R_6^P = * \; * \; * \; 1$
$R_7^D = * \; * \; * \; *$



- 根から $\boxed{i}$ へのパスの数が $|R_i|$ となるよう MTZDD を構築
- $|R_i|$ が必要なときは，根から $\boxed{i}$ へのパスの数を数える
  (パスの数を数えるアルゴリズムは節点に比例するオーダ)

ルール R_i に対応する MTZDD

Filter $\mathbf{R}$
$R_1^P = * 0 * 1$
$R_2^P = 0 0 0 0$
$R_3^D = 0 * 0 0$
$R_4^D = 0 * 1 *$
$R_5^P = * 1 * 1$
$R_6^P = * * * 1$
$R_7^D = * * * *$



$n-1$ 個の各ルールに対応する MTZDD を構築，各 MTZDD は $\boxed{0}$ を共有

$\boxed{0}$ は R_n（ルールリストの最後のルール）に対応

ルール R_i に対応する MTZDD 構築

例：$R_1^P = *0*1$ に対応する MTZDD を構築

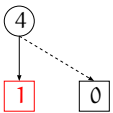
1. make terminal nodes $\boxed{0}$, $\boxed{i}$
2. pointer p points to $\boxed{i}$
3. $k \leftarrow n$ (n is the number of variables)
4. **while** $k \geq 1$ **do**
   - **if** cond[k] = '1' **then**
     - make $\textcircled{k}$ whoes 0 points to $\boxed{0}$ and 1 points to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - **if** cond[k] = '$*$' **then**
     - make $\textcircled{k}$ whoes 0 and 1 point to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - $k \leftarrow k - 1$

$\boxed{1}$ $\boxed{0}$

ルール R_i に対応する MTZDD 構築

例：$R_1^P = * \, 0 * 1$ に対応する MTZDD を構築

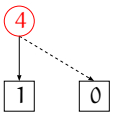
1. make terminal nodes $\boxed{0}$, $\boxed{i}$
2. pointer p points to $\boxed{i}$
3. $k \leftarrow n$ (n is the number of variables)
4. **while** $k \geq 1$ **do**
   - **if** cond[k] = '1' **then**
     - make $\textcircled{k}$ whoes 0 points to $\boxed{0}$ and 1 points to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - **if** cond[k] = '$*$' **then**
     - make $\textcircled{k}$ whoes 0 and 1 point to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - $k \leftarrow k - 1$



ルール R_i に対応する MTZDD 構築

例：$R_1^P = *0*1$ に対応する MTZDD を構築

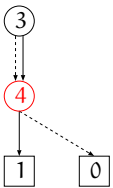
1. make terminal nodes $\boxed{0}$, $\boxed{i}$
2. pointer p points to $\boxed{i}$
3. $k \leftarrow n$ (n is the number of variables)
4. **while** $k \geq 1$ **do**
   - **if** cond[k] = '1' **then**
     - make $\textcircled{k}$ whoes 0 points to $\boxed{0}$ and 1 points to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - **if** cond[k] = '$*$' **then**
     - make $\textcircled{k}$ whoes 0 and 1 point to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - $k \leftarrow k - 1$



ルール R_i に対応する MTZDD 構築

例：$R_1^P = *0*1$ に対応する MTZDD を構築

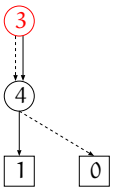
1. make terminal nodes $\boxed{0}$, $\boxed{i}$
2. pointer p points to $\boxed{i}$
3. $k \leftarrow n$ (n is the number of variables)
4. **while** $k \geq 1$ **do**
   - **if** cond[k] = '1' **then**
     - make $\textcircled{k}$ whoes 0 points to $\boxed{0}$ and 1 points to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - **if** cond[k] = '$*$' **then**
     - make $\textcircled{k}$ whoes 0 and 1 point to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - $k \leftarrow k - 1$



ルール R_i に対応する MTZDD 構築

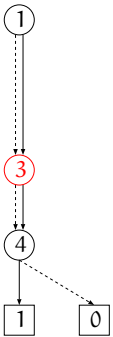
例：$R_1^P = * \, 0 * 1$ に対応する MTZDD を構築

1. make terminal nodes $\boxed{0}$, $\boxed{i}$
2. pointer p points to $\boxed{i}$
3. $k \leftarrow n$ (n is the number of variables)
4. **while** $k \geq 1$ **do**
   - **if** cond[k] = '1' **then**
     - make $\textcircled{k}$ whoes 0 points to $\boxed{0}$ and 1 points to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - **if** cond[k] = '$*$' **then**
     - make $\textcircled{k}$ whoes 0 and 1 point to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - $k \leftarrow k - 1$



ルール R_i に対応する MTZDD 構築

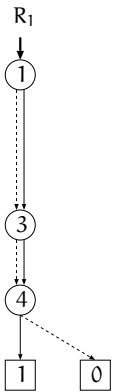
例：$R_1^P = * \, 0 * 1$ に対応する MTZDD を構築



1. make terminal nodes $\boxed{0}$, $\boxed{i}$
2. pointer p points to $\boxed{i}$
3. $k \leftarrow n$ (n is the number of variables)
4. **while** $k \geq 1$ **do**
   - **if** cond[k] = '1' **then**
     - make $\textcircled{k}$ whoes 0 points to $\boxed{0}$ and 1 points to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - **if** cond[k] = '$*$' **then**
     - make $\textcircled{k}$ whoes 0 and 1 point to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - $k \leftarrow k - 1$

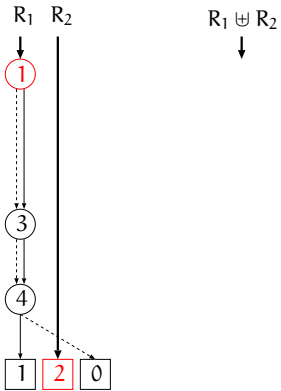
ルール R_i に対応する MTZDD 構築

例：$R_1^P = * \, 0 * 1$ に対応する MTZDD を構築



1. make terminal nodes $\boxed{0}$, $\boxed{i}$
2. pointer p points to $\boxed{i}$
3. $k \leftarrow n$ (n is the number of variables)
4. **while** $k \geq 1$ **do**
   - **if** cond[k] = '1' **then**
     - make $\textcircled{k}$ whoes 0 points to $\boxed{0}$ and 1 points to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - **if** cond[k] = '$*$' **then**
     - make $\textcircled{k}$ whoes 0 and 1 point to $\textcircled{p}$
     - p points to $\textcircled{k}$
   - $k \leftarrow k - 1$

MTZDD の統合： $P \uplus Q$

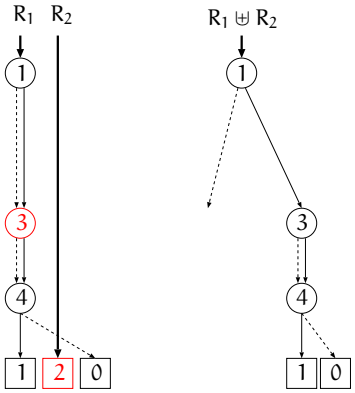


1. **if** $P = \emptyset$ **then return** Q;
2. **if** $Q = \emptyset \vee P = Q$ **then return** P;
3. **if** P and Q are terminal nodes **then**
4. **if** $P.val < Q.val$ **then return** P; **else return** Q;
5. $R \leftarrow$ cache$[P \uplus Q]$;
6. **if** R exists **then return** R;
7. **if** $P.top < Q.top$ **then**
   $R \leftarrow$ getNode$(P.top, P.val, (P_0 \uplus Q), P_1)$;
8. **else if** $Q.top < P.top$ **then**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P \uplus Q_0), Q_1)$;
9. **else**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P_0 \uplus Q_0), (P_1 \uplus Q_1))$;
10. cache$[P \uplus Q] \leftarrow R$;
11. **return** R;

getNode(top, val, P_0, P_1) は変数が top, 値が val,
0 枝が P_1,1 枝が P_1 を指すような節点を返す演算

$P \uplus Q$ の例：P は R_1 の MTZDD，Q は R_2 の MTZDD

MTZDD の統合： $P \uplus Q$

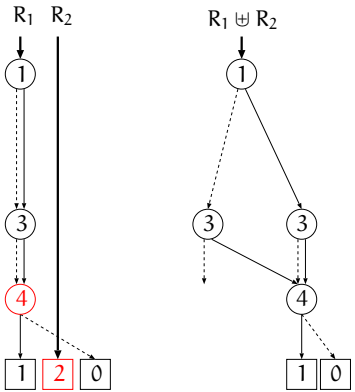


1. **if** $P = \emptyset$ **then return** Q;
2. **if** $Q = \emptyset \vee P = Q$ **then return** P;
3. **if** P and Q are terminal nodes **then**
4. **if** $P.val < Q.val$ **then return** P; **else return** Q;
5. $R \leftarrow$ cache$[P \uplus Q]$;
6. **if** R exists **then return** R;
7. **if** $P.top < Q.top$ **then**
   $R \leftarrow$ getNode$(P.top, P.val, (P_0 \uplus Q), P_1)$;
8. **else if** $Q.top < P.top$ **then**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P \uplus Q_0), Q_1)$;
9. **else**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P_0 \uplus Q_0), (P_1 \uplus Q_1))$;
10. cache$[P \uplus Q] \leftarrow R$;
11. **return** R;

getNode(top, val, P_0, P_1) は変数が top, 値が val,
0 枝が P_1,1 枝が P_1 を指すような節点を返す演算

$P \uplus Q$ の例：P は R_1 の MTZDD，Q は R_2 の MTZDD

MTZDD の統合： $P \uplus Q$

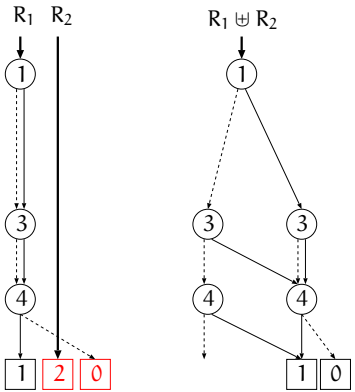


1. **if** $P = \emptyset$ **then return** Q;
2. **if** $Q = \emptyset \vee P = Q$ **then return** P;
3. **if** P and Q are terminal nodes **then**
4. **if** $P.val < Q.val$ **then return** P; **else return** Q;
5. $R \leftarrow$ cache$[P \uplus Q]$;
6. **if** R exists **then return** R;
7. **if** $P.top < Q.top$ **then**
   $R \leftarrow$ getNode$(P.top, P.val, (P_0 \uplus Q), P_1)$;
8. **else if** $Q.top < P.top$ **then**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P \uplus Q_0), Q_1)$;
9. **else**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P_0 \uplus Q_0), (P_1 \uplus Q_1))$;
10. cache$[P \uplus Q] \leftarrow R$;
11. **return** R;

getNode(top, val, P_0, P_1) は変数が top, 値が val,
0 枝が P_1,1 枝が P_1 を指すような節点を返す演算

$P \uplus Q$ の例：P は R_1 の MTZDD，Q は R_2 の MTZDD

MTZDD の統合： $P \uplus Q$

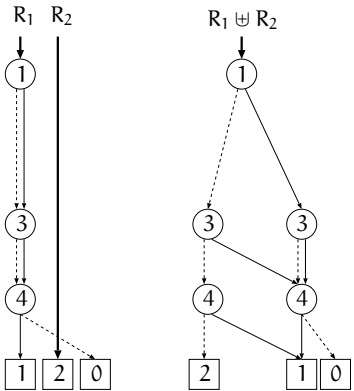


1. **if** $P = \emptyset$ **then return** Q;
2. **if** $Q = \emptyset \vee P = Q$ **then return** P;
3. **if** P and Q are terminal nodes **then**
4. **if** $P.val < Q.val$ **then return** P; **else return** Q;
5. $R \leftarrow$ cache$[P \uplus Q]$;
6. **if** R exists **then return** R;
7. **if** $P.top < Q.top$ **then**
   $R \leftarrow$ getNode$(P.top, P.val, (P_0 \uplus Q), P_1)$;
8. **else if** $Q.top < P.top$ **then**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P \uplus Q_0), Q_1)$;
9. **else**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P_0 \uplus Q_0), (P_1 \uplus Q_1))$;
10. cache$[P \uplus Q] \leftarrow R$;
11. **return** R;

getNode(top, val, P_0, P_1) は変数が top，値が val，
0 枝が P_1，1 枝が P_1 を指すような節点を返す演算

$P \uplus Q$ の例：P は R_1 の MTZDD，Q は R_2 の MTZDD

MTZDD の統合： $P \uplus Q$



1. **if** $P = \emptyset$ **then return** Q;
2. **if** $Q = \emptyset \vee P = Q$ **then return** P;
3. **if** P and Q are terminal nodes **then**
4. **if** $P.val < Q.val$ **then return** P; **else return** Q;
5. $R \leftarrow$ cache$[P \uplus Q]$;
6. **if** R exists **then return** R;
7. **if** $P.top < Q.top$ **then**
   $R \leftarrow$ getNode$(P.top, P.val, (P_0 \uplus Q), P_1)$;
8. **else if** $Q.top < P.top$ **then**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P \uplus Q_0), Q_1)$;
9. **else**
   $R \leftarrow$ getNode$(Q.top, Q.val, (P_0 \uplus Q_0), (P_1 \uplus Q_1))$;
10. cache$[P \uplus Q] \leftarrow R$;
11. **return** R;

getNode(top, val, P_0, P_1) は変数が top，値が val，
0 枝が P_1，1 枝が P_1 を指すような節点を返す演算

$P \uplus Q$ の例：P は R_1 の MTZDD，Q は R_2 の MTZDD

実験環境

OS	:	CentOS Release 6.2
CPU	:	Intel Core i7-980X 3.33 GHz
主記憶容量	:	24GB
実装言語	:	C++
コンパイラ	:	gcc version 4.8.2

- ルール長 120，ルール数が百 ~5 万のルールリストをランダムに生成
- 節点数，構築時間 (s) を計測

実験結果

ルール数	節点数	構築時間 (ms)
100	9604	7
500	62315	47
1k	163558	324
5k	1456809	5847
10k	4521134	18981
50k	51908415	494257

ルール数 5 万という現実的なサイズのルールリストに対して MTZDD を構築可能

まとめと今後の課題

まとめ

- ルール順序最適化問題では評価パケット数の変動を考慮すべし
- 一様分布での MTZDD による評価パケット数算出法を提案

今後の課題

- ルールリストに対する MTZDD の領域計算量の算出
- 任意の分布における評価パケット数算出法の考案
- 一部のルールの並び替えに伴う評価パケット数変動の算出
- 評価パケット数の変動を考慮したルール順序最適化問題に対する発見的解法の考案