Vol.2019-AL-171 No.8
2019/1/30

ZDDによるルールリストポリシーの等価判定

原田 崇司[1,a)]　田中 賢[1,b)]　三河 賢治[2,c)]

概要：パケット分類問題は，ネットワーク機器に到着するパケットの振る舞いを決定する問題である．ネットワーク機器に到着したパケットは，ルールリストの上位のルールから順番に照合され，最初に合致したルールのアクションが適用される．パケットとルールとの照合回数が増加すると通信に遅延が生じる．この遅延を減少させるために，ルールを並び替えたり，ルールリストを再構築する手法が提案されてきた．ルールの並び替え替えやルールリスト再構築によって生成されたルールリストは，元のルールリストと同じポリシーを満たさなければならない．二つのルールリストが与えられたとき，それらが表すポリシーが等しいかを判定する問題は **coNP** 完全である．本稿では，ルールリストが満たすポリシーを表現するZDD を構築することによって，ルールリストの等価判定を行う手法を提案する．また，計算機実験により提案手法の有効性を確かめる．

Deciding Equivalence of The Rule List Policies via ZDD

Harada Takashi[1,a)]　Tanaka Ken[1,b)]　Mikawa Kenji[2,c)]

1. はじめに

パケット分類問題は，ネットワーク機器に到着するパケットの振る舞いを決定する問題である．ネットワーク機器に到着したパケットは，ネットワーク管理者の意図を反映した分類ポリシーに従って分類される．分類ポリシーは，決定リスト [1] のようなルールのリストとして表現され，パケットはルールリストの上位のルールから順番に照合され，最初に合致したルールのアクションが適用される．パケットとルールとの照合回数が増加すると通信に遅延が生じる．この遅延を減少させるために，ルールを並び替えたり，ルールリストを再構築する手法が提案されてきた [2], [3], [4], [5], [6], [7], [8], [9], [10]．ルールの並び替え替えやルールリスト再構築によって生成されたルールリストは，元のルールリストと同じポリシーを満たさなければならない．けれども，文献 [2] の手法は，並び替えたルールリストがポリシーを満たさない場合がある．文献 [7] の手法で並び替えたルールリストや，ルールリストを再構築した場合も，並び替え前と並び替え後のルールリストの等価判定は簡単ではないので，ルールリストの等価判定を高速に行う手法が求められる．一方，ルールリストを決定リストとみなすと，二つのルールリストが与えられたとき，それらが表すポリシーが等しいかを判定する問題は **coNP** 完全である [11]．そこで本稿では，ルールリストが満たすポリシーを表現する ZDD[12] を構築することによって，ルールリストの等価判定を高速に行う手法を提案する．また，計算機実験により提案手法の有効性を確かめる．

2. パケット分類モデル

ネットワーク機器に到着するパケットの分類は，図 1 のようにモデル化される．到着したパケットは，ルールリストの上位のルールから順番に照合され，最初に合致したルールのアクションが適用される．

各ルールは，ルール番号 $i \in \{1, 2, \ldots, n\}$，アクション $e \in \{A_1, \ldots, A_m\}$ と，合致するパケットの集合を表現する文字列 $b_1 b_2 \ldots b_w \in \{0, 1, *\}^w$ から成る．ただし，n はルー

[1] 神奈川大学大学院理学研究科情報科学領域
Kanagawa University, Hiratsuka, Kanagawa 259–1293, Japan
[2] 新潟大学学術情報基盤機構情報基盤センター
Niigata University, Niigata, Niigata 950–2181, Japan
a) takaharada@jindai.jp
b) ktanaka@info.kanagawa-u.ac.jp
c) mikawa@cais.niigata-u.ac.jp

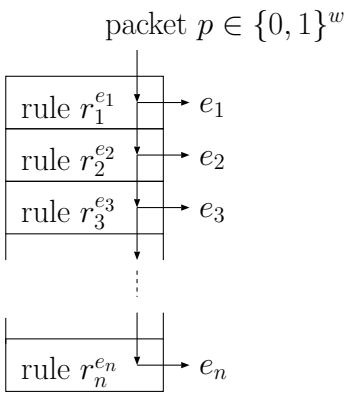

図 1 パケット分類モデル

表 1 ルールリスト

Filter $\mathcal{R}$
$r_1^D = * 1 0 0$
$r_2^P = 0 1 * *$
$r_3^D = * 0 0 1$
$r_4^P = * * * 1$
$r_5^P = 0 * 1 *$
$r_6^D = * 1 1 0$
$r_7^P = * 1 * *$
$r_8^D = * * * *$

表 2 表 1 が表す函数 f

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

ルリストのルール数，$A_1, A_2, \ldots, A_m$ はパケットに適用されるアクション，m はアクションの数，w は文字列の長さ，$*$ はワイルドカードマスクを表す．ルールは，式 (1) のように表される．

定義 2.1 *(ルールの形式)*

$$r_i^e = b_1 b_2 \ldots b_w,\ b_k \in \{0, 1, *\},\ e \in \{A_1, \ldots, A_m\} \quad (1)$$

ルールリストの例を表 1 に示す．表 1 の例では，文字列の長さ w は 4，アクションの数 m は 2 で，$A_1 = P$，$A_2 = D$ としている．P は Permit を D は Deny を表し，それぞれ，パケットの通過の許可と拒否を意味する．以降，5 章までは，アクションの種類は Permit と Deny の二つだけとする．

ルールリストは，パケット分類ポリシー $f : \mathcal{P} \to \{A_1, A_2, \ldots, A_m\}$ を表す．ただし，$\mathcal{P}$ はネットワーク機器に到着可能なパケット全体の集合 $\{0,1\}^w$ を表す．表 1 のルールリストは，表 2 のポリシー $f : \mathcal{P} \to \{P, D\}$ を表している．例えば，パケット 0111 は，$r_1^D = 0001$ から順に照合が行われ，$r_5^P = 0***$ に最初に合致するので，r_5^P のアクション Permit が適用される．

ルール r_i が合致する可能性のあるパケットの集合を $M(r_i)$ と表す．例えば，表 1 のルールリストの r_5^P に対して

$$M(r_4^P) = \{0001, 0011, 0101, 0111, 1001, 1011, 1101, 1111\}$$

となる．ルール r_i^e のアクション e は，$M(r_i^e)$ に関係しないので，以降 $M(r_i^e)$ を $M(r_i)$ と記す．

表 3 到着パケットの頻度分布 $F : \mathcal{P} \to \mathbb{N}$

$0000 \mapsto 23$	$0100 \mapsto 1$	$1000 \mapsto 0$	$1100 \mapsto 1$
$0001 \mapsto 7$	$0101 \mapsto 3$	$1001 \mapsto 0$	$1101 \mapsto 0$
$0010 \mapsto 0$	$0110 \mapsto 13$	$1010 \mapsto 0$	$1110 \mapsto 17$
$0011 \mapsto 9$	$0111 \mapsto 0$	$1011 \mapsto 2$	$1111 \mapsto 0$

表 4 ルールリスト

Filter $\mathcal{R}$	$\lvert w_i \rvert_F$
$r_1^D = * 1 0 0$	2
$r_2^P = 0 1 * *$	3
$r_3^D = * 0 0 1$	7
$r_4^P = * * * 1$	11
$r_5^P = 0 * 1 *$	13
$r_6^D = * 1 1 0$	17
$r_7^P = * 1 * *$	0
$r_8^D = * * * *$	23

表 5 再構築

Filter $\hat{\mathcal{R}}$	$\lvert w_i \rvert_F$
$r_1^D = 1 0 1 0$	0
$r_2^P = * 0 1 *$	11
$r_3^P = 0 1 1 0$	0
$r_4^D = * 1 * 0$	19
$r_5^P = * 1 * *$	3
$r_6^D = * * * *$	30

ルールリスト $\mathcal{R}$ が与えられると，ルール $r_i \in \mathcal{R}$ によって適用されるアクションが決まるパケットの集合が定まる．この集合のことを w_i と記す．例えば，表 1 のルールリストにおいて，

$$w_4 = \{0011, 1011, 1101, 1111\}$$

となる．

パケットの集合を $\mathcal{P}$，到着パケットの頻度分布を $F : \mathcal{P} \to \mathbb{N}$ とし，$\sum_{p \in \mathcal{P}} F(p)$ を $|\mathcal{P}|_F$ と表す．例えば，$\mathcal{P} = \{0000, 0001, 1001\}$ とし，F を表 3 のような頻度分布とすると

$$|\mathcal{P}|_F = 23 + 7 + 0 = 30$$

となる．

到着パケットの頻度分布 F とルールリスト $\mathcal{R}$ が与えられたとき，ルール r_i^e によってアクションが与えられるパケットの数を r_i^e の評価パケット数と呼び，$|w_i|_F$ と記す．例えば，表 1 のルールリスト，表 3 の頻度分布において，

$$|w_i|_F = 9 + 2 + 0 + 0 = 11$$

となる．

パケットとルールとの比較を遅延 1 と考えると．到着パケットの頻度分布 F，ルールリスト $\mathcal{R}$ の遅延 $L(\mathcal{R}, F)$ は式 (2) のように定義される．

定義 2.2 *(フィルタリング遅延)*

$$L(\mathcal{R}, F) = i|w_i|_F + (n-1)|w_n|_F \quad (2)$$

例えば，一様分布 $U : \mathcal{P} \to \{1\}$ における，表 1 のルールリスト $\mathcal{R}$ の遅延は以下のようになる．

$$\begin{aligned} L(\mathcal{R}, U) &= 1{\cdot}2 + 2{\cdot}3 + 3{\cdot}2 + 4{\cdot}4 + 5{\cdot}1 + 6{\cdot}1 + 7{\cdot}0 + 7{\cdot}3 \\ &= 62 \end{aligned}$$

また，頻度分布を表 3 の頻度分布 F とすると，遅延は $L(\mathcal{R}, F) = 401$ となる．

表 1 のルールリスト $\mathcal{R}$ を，表 2 のポリシーを保ちながら，$\mathcal{R}' = \langle r_3, r_1, r_2, r_4, r_5, r_6, r_7, r_8 \rangle$ と並び替えると，表 3 の頻度分布 F における遅延は $L(\mathcal{R}', F) = 391$ となる．さらに，表 1 のルールリスト $\mathcal{R}$ が表すポリシーを満たす，表 5 のようなルールリスト $\hat{\mathcal{R}}$ を再構築すると，表 3 の頻度分布 F における遅延は $L(\hat{\mathcal{R}}, F) = 293$ となる．このように，ルールを並び替えたり，ルールリストを再構築することにより，遅延を減らせる場合がある．

ルール並び替え，ルールリスト再構築に求められる条件を定義する．

定義 2.3 *(ポリシー違反)* ルール並び替え，ルールリスト再構築によって，ルールリスト $\mathcal{R}$ から生成されるルールリスト $\mathcal{R}'$ が

$$\forall p \in \mathcal{P}\ \mathcal{R}(p) = \mathcal{R}'(p)$$

を満たさないとき，すなわち，$\mathcal{R}(p) \neq \mathcal{R}'(p)$ となるパケットが存在するとき，$\mathcal{R}'$ をポリシー違反を起こすルールリストという．ただし，$\mathcal{R}(p)$ は，ルールリスト $\mathcal{R}$ がパケット p に適用するアクションである．

例えば，表 1 のルールリストを $\mathcal{R}'' = \langle r_2, r_1, r_3, r_4, r_5, r_6, r_7, r_8 \rangle$ と並び替えると，パケット 0100 に適用されるアクションが $\mathcal{R}(0100) = D$ から $\mathcal{R}''(0100) = P$ へと変わってしまうので，$\mathcal{R}''$ はポリシー違反を起こすルールリストであり，$\mathcal{R}''$ のように並び替えることは許されない．

ルールの並び替えにおけるポリシー違反を起こさないための基準として，ルール上の重複関係と従属関係を定義する．

定義 2.4 *(ルールの重複)* $r_i^{e_i}$ と $r_j^{e_j}$ の両方に合致するパケットが存在するとき，即ち，$M(r_i) \cap M(r_j) \neq \emptyset$ のとき，$r_i^{e_i}$ と $r_j^{e_j}$ は重複する，という．$r_i^{e_i}$ と $r_j^{e_j}$ が重複するとき，$O(r_i, r_j)$ と表す．

例えば，表 1 のルールリストにおいて，r_4^P と r_5^P は $M(r_4) \cap M(r_5) = \{0011, 0111\} \neq \emptyset$ なので，r_4^P と r_5^P は重複している．

定義 2.5 *(ルールの従属)* $r_i^{e_i}$ と $r_j^{e_j}$ が重複しており，$i < j$ 且つ $e_i \neq e_j$ のとき，$r_j^{e_j}$ は $r_i^{e_i}$ に従属する，という．$r_j^{e_j}$ が $r_i^{e_i}$ に従属するとき，$D(r_i, r_j)$ と表す．

例えば，表 1 のルールリストにおいて，r_5^P と r_6^D は $M(r_5) \cap M(r_6) = \{0110\} \neq \emptyset$ なので，r_5^P と r_6^D は重複しており，アクションが P と D と異なるので，r_5^P と r_6^D は従属関係にある．

表 1 のルールリストに対して，重複関係による先行関係と従属関係による先行関係のグラフを図 2, 3 に示す．ただし，推移的な枝は削除している．すなわち，(r_j, r_i) であっても，$r_j \to r_i$ となる経路が存在する場合は枝 (r_j, r_i) はグラフから取り除いている．

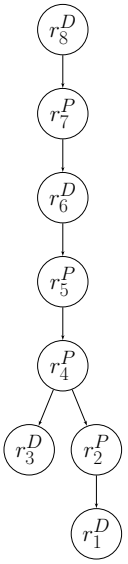

図 2 重複関係

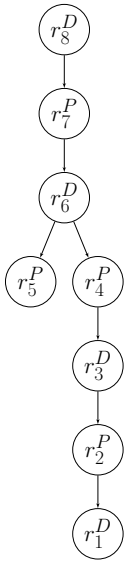

図 3 従属関係

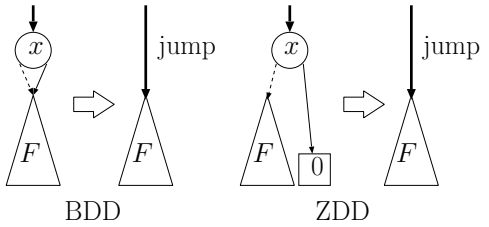

図 5 節点削除規則

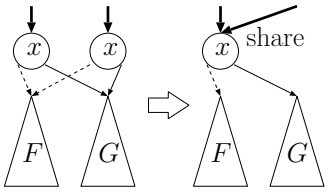

図 6 節点共有規則

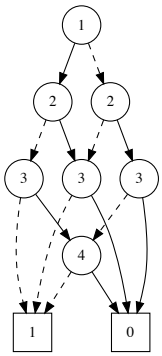

図 7 BDD

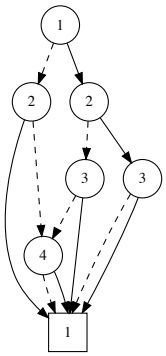

図 8 ZDD

3. BDD/ZDD

我々の提案手法は BDD[13] の亜種である ZDD[12] を用

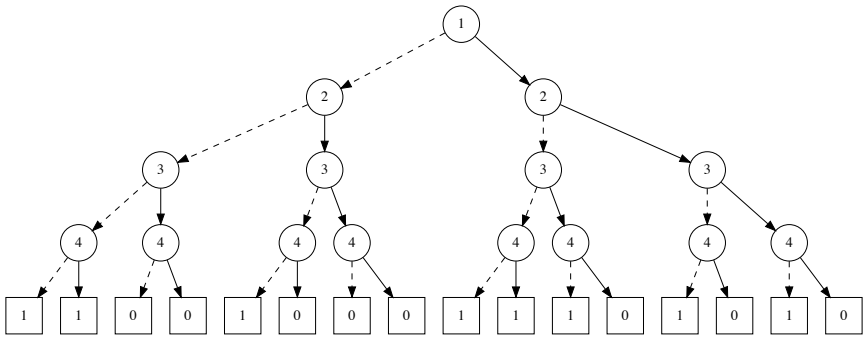

図 4 場合分け二分木

いるので，本章では BDD と ZDD について説明する．

BDD は論理函数を表す根つきの無閉路有向グラフである [14]．BDD は論理函数を表す場合分け二分木を図 6,5 の簡約規則を適用することによって得られる．例えば，図 7 の BDD は図 4 の場合分け二分木に図 6,5 の簡約規則を既約になるまで適用することにより得られる．図 4 の場合分け二分木は論理函数 $\bar{x_1}\bar{x_2}\bar{x_3} \vee \bar{x_1}x_2\bar{x_3}\bar{x_4} \vee x_1\bar{x_2}\bar{x_3} \vee x_1\bar{x_2}x_3\bar{x_4} \vee x_1x_2\bar{x_4}$ を表しており，丸で囲われた非終端節点は論理変数を表しており，丸い節点から出ている破線と実線はその節点に対応する変数が 0 と 1 をとることをそれぞれ表す．BDD の根から四角で囲われた $0,1$ 終端節点へのパスは，そのパスに対応する変数の割当の付値がそれぞれ $0,1$ に対応することを表す．なお，パスにおいて途中の変数を飛ばしている場合は，対応する割当において函数値が飛ばした変数に依存しないことを表す．例えば，図 7 の根節点，1 枝，変数番号 2 の非終端節点，0 枝，変数番号 3 の非終端節点，1 終端節点のパスは，割当 $1000, 1001$ に対する函数値が 1 であることを意味する．

w 個のアイテムの組合せは，長さ w のビットベクトル $(b_1, b_2, \ldots, b_w)$ で表現できる．各 $b_k (1 \leq k \leq w)$ は，k 番目のアイテムを含むか否かを表す．組合せ集合はビットベクトルの集合として表すことができ，$M(r_i)$ は組合せ集合と見做せる．

Zero-Suppressed Binary Decision Diagram (ZDD) は，組合せ集合を効率よく扱うために湊によって提案されたデータ構造である [12]．ZDD も BDD と同様に図 4 の場合分け二分木に図 6,5 の簡約規則を既約になるまで適用することにより得られる．丸で囲まれた非終端節点，四角で囲まれた終端節点と $0,1$ 枝の意味は BDD と同じである．根節点から 1 終端節点へのパスにおいて，変数番号 k の非終端節点を飛ばしているとき，パスに対応する組合せ集合にアイテム k は含まれない．例えば，図 8 の根節点, 0 枝，2 非終端節点，1 枝のパスは，変数番号 $3, 4$ に対応する節点を飛ばしているので，このパスに対応する組合せはアイテム 3 と 4 を含まない．

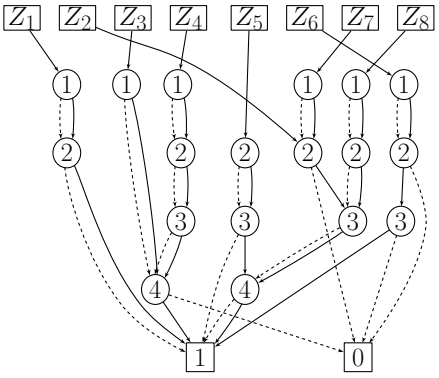

図 10 ZDDs

4. ルールリスト等価判定

ルール並び替えにおいて，重複関係，従属関係による先行関係を保持すれば，ポリシー違反が起きないので，ルール並び替えに関する既存研究のほとんどは，重複関係と従属関係に基づいてルールを並び替えている．より正確には，従属関係による先行関係を維持することと，ポリシー違反が起きないことがほぼ等価なので，従属関係に基づいて並び替えを行なっている．けれども，文献 [3] で指摘されているように，ルール $r_j^{e_j}$ が $r_i^{e_i}$ に従属していても，$r_j^{e_j}$ を $r_i^{e_i}$ よりも前方に配置してもポリシー違反が起きない場合がある．例えば，表 1 のルールリストにおいて，r_8^D は r_7^P に従属するが，実は r_7^P によってアクションが決まるパケットが存在しないので，r_8^D を r_7^P よりも上位に配置することができる．

ルールリストが，重複関係，もしくは従属関係によって並び替えられたのであれば，並び替えられたルールリストがポリシーを保持するかどうかは，任意のルールに対して従属関係が保持されているかを確認すればよい．この確認は，ルール数 n と条件式の長さ w に関して $O(n^2w)$ で実行できる．けれども，従属関係に依らず，文献 [3] のような

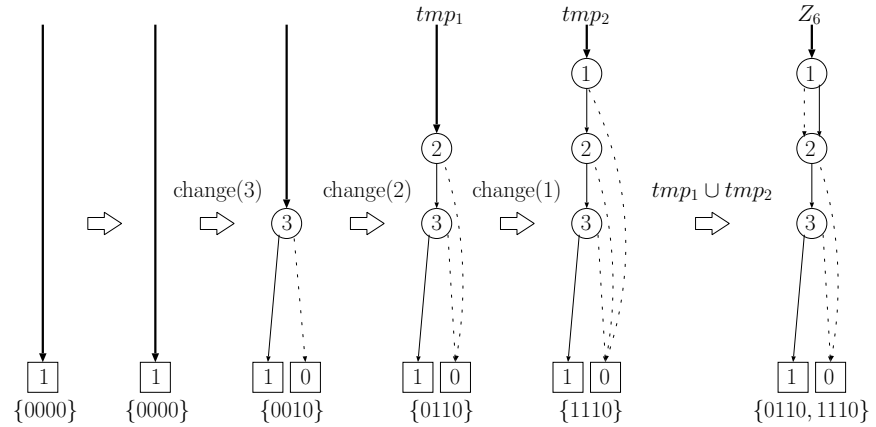

図 9 Algorithm 1 による r_6^D に合致するパケットの集合に対応する ZDD の構築過程

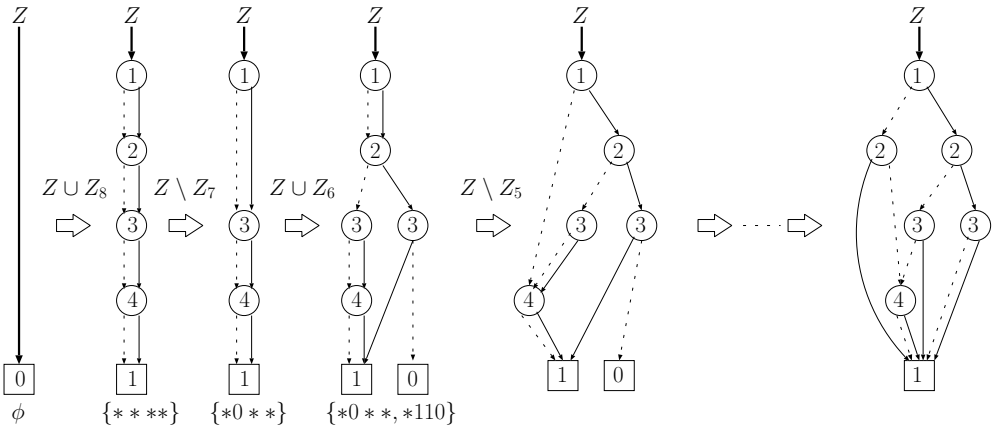

図 11 Algorithm 2 によるポリシーを表現する ZDD の構築過程

手法で並び替えられたルールリストに対しては，ポリシー違反の有無を簡単には確認できない．また，再構築されたルールリストのポリシー違反の有無も簡単には確認できない．

二つの決定リスト L_1 と L_2 が与えられたとき，その二つの決定リストが同じ論理函数を表現しているかを判定する問題は **coNP** 完全である [11]．決定リストの等価判定をルールリストの等価判定へと多項式時間で帰着できるので，二つのルールリスト $\mathcal{R}_1$ と $\mathcal{R}_2$ の等価判定も **coNP** 完全である．

このように，ルールの並び替え，ルールリスト再構築を行なったときにポリシー違反が起きていないかを判定することは一般には難しい．けれども，ある程度のサイズのルールリストに対しては，ルールリストに対応する ZDD を構築することによって，ポリシー違反の有無を判定できる．以下では，ルールリストに対応する ZDD の構築法を示し，ZDD を用いたルールリストポリシーの等価判定法を提案する．

Algorithm 1 に，CUDD[15] などの BDD/ZDD のライブラリを用いた，ルールリストの各ルール r_i^e の $M(r_i)$ に対する ZDD の構築法を示す．Algorithm 1 では初めに，$00\cdots0$ だけを含む組合せ集合の ZDD である Z を作成する．そして，$r_i^e = b_1 b_2 \cdots b_w$ の条件を後ろから走査して，$b_k = 1$ ならば，change(k) 演算を Z に適用し，$b_k = *$ ならば，Z と change(k) を適用した Z の結びを取る．Algorithm 1 によって，表 1 の r_6^D の $M(r_6)$ に対応する ZDD を構築する例を図 9 に示す．

Algorithm 1: make ZDD for Rule r using BDD/ZDD Library like CUDD [15]

```
   input  : rule r_i^e = b_1 b_2 ... b_w
   output: ZDD for rule r_i^e
1  make ZDD Z for the set of null combination {00...0};
2  i ← w ;
3  while i > 0 do
4      if b_i = '1' then Z.change(i);
5      else if b_i = '*' then
6          Z ← Z ∪ Z.change(i);
       end
7      i ← i − 1 ;
   end
8  return Z;
```

Algorithm 2 に，ルールリスト $\mathcal{R} = \langle r_1^{e_1}, r_2^{e_2}, \ldots, r_n^{e_n} \rangle$ に

Algorithm 2: make ZDD for Rule List $\mathcal{R}$

input : rule list $\mathcal{R} = \langle r_1^{e_1}, r_2^{e_2}, \ldots, r_n^{e_n} \rangle$
output: ZDD for rule list $\mathcal{R}$

```
1 make zdd Z for the empty set ;
2 i ← n ;
3 while i > 0 do
4 |   make zdd Z_i for r_i^{e_i} by Algorithm 1 ;
5 |   if e^i = D then Z ← Z ∪ Z_i ;
6 |   else Z ← Z \ Z_i ;
7 |   i ← i − 1 ;
  end
8 return Z ;
```

表 6 リスト 1

Filter $\mathcal{R}_1$
$r_1^B = * 1 1 0$
$r_2^A = 0 * 1 *$
$r_3^B = 1 0 * 0$
$r_4^C = 1 1 0 1$
$r_5^C = 1 1 1 1$
$r_6^D = 1 * * *$
$r_7^A = * 1 * 1$
$r_8^D = * 0 * 0$
$r_9^C = * * * *$

表 7 リスト 2

Filter $\mathcal{R}_2$
$r_1^A = 0 0 1 0$
$r_2^B = * * 1 0$
$r_3^C = 0 0 0 1$
$r_4^A = 0 * * 1$
$r_5^D = 1 1 0 0$
$r_6^C = * 1 * *$
$r_7^B = 1 0 0 0$
$r_8^D = 1 0 * 1$
$r_9^D = * * * *$

対する ZDD の構築法を示す．Algorithm 2 では初めに，空集合のみを含む組合せ集合の ZDD である Z を作成する．そして，ルール $r_n^{e_n}$ からルール番号の降順にルールを捜査する．r_i^e に対する ZDDZ_i を構築し，$e = D$ ならば，Z を $Z \cup Z_i$ とし，$e = P$ ならば，Z を $Z \setminus Z_i$ とする．Algorithm 2 によって，表 1 のルールリストに対応する ZDD を構築する例を図 11 に示す．

ルールリスト $\mathcal{R}_1$ と $\mathcal{R}_2$ のポリシーが等価かどうかは，上記のように $\mathcal{R}_1$ と $\mathcal{R}_2$ のそれぞれに対して ZDD_1 と ZDD_2 を構築し，ZDD_1 と ZDD_2 が同じかどうかを確認すればよい．二つの ZDD ZDD_1 と ZDD_2 が同じかどうかは，ZDD_1 と ZDD_2 が同じ節点を指しているかを確認すればよいので 1 ステップで判定できる．つまり，ルールリストに対応する ZDD を構築さえできれば，ルールリストポリシーの等価判定が行える．

5. 多値ルールリストの等価判定

前章では，パケットに適用されるアクションが P と D の二つだけのルールから成るルールリストポリシーの等価判定アルゴリズムを示した．本章では，表 6,7 のように，アクションを P と D だけに制限しないルールから成るルールリストポリシーの等価判定アルゴリズムを提案する．以降，アクションが P と D だけに制限されていないルールリストを多値のルールリストとよぶ．

表 8 表 6,7 が表す函数 $f : \mathcal{P} \to \{A, B, C, D\}$

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

Algorithm 3: make ZDDs for Multiple Actions Rule List $\mathcal{R}$

input : rule list $\mathcal{R} = \langle r_1^{e_1}, r_2^{e_2}, \ldots, r_n^{e_n} \rangle$
output: ZDDs $X_1, X_2, \ldots, X_m$ for rule list $\mathcal{R}$

```
1  i ← 1 ;
   // m is the number of actions {A_1, A_2, ..., A_m};
2  while i ≤ m do
3  |   make zdd X_i for the empty set ;
4  |   j ← n;
5  |   while j > 0 do
6  |   |   make zdd tmp for r_j^{e_j} by Algorithm 1 ;
7  |   |   if e^j = A_i then X_i ← X_i ∪ tmp ;
   |   |   ;
8  |   |   else X_i ← X_i \ tmp ;
9  |   |   i ← i − 1 ;
   |   end
   end
10 return X_1, X_2, ..., X_m ;
```

パケットに適用されるアクションの候補が $A_1, A_2, \ldots, A_m$ と三つ以上あるような多値のルールリストに対しては，アクションの数だけ，前章のように ZDD を構築する．

多値のルールリストに対する ZDD$X_1, X_2, \ldots, X_m$ を構築するアルゴリズムを Algorithm3 に示す．ただし，m はアクションの数である．

Algorithm3 は，2 行目から 9 まで，Algorithm2 とほぼ同じことを繰り返して，アクション i に対する ZDD X_i を構築する．違いは 7 行目において，アクションが D かではなく，アクションが A_i と一致するかを比較していることである．

ルールリスト $\mathcal{R}_1$ と $\mathcal{R}_2$ に対して，Algorithm3 により，$X_{11}, X_{12}, \ldots, X_{1m}$ と $X_{21}, X_{22}, \ldots, X_{2m}$ の $2m$ 個の ZDD を構築し，m 個の各 ZDD が等しいかを判定すれば，$\mathcal{R}_1$ と $\mathcal{R}_2$ のポリシー等価判定が行える．

6. 計算機実験

アクションが P もしくは D であるルールのみから成るルールリストのポリシー等価判定法の有効性を確かめるために C 言語を用いて計算機実験を行った．実験環境は，主記憶容量が 2GB，CPU が Intel Core i5-3470 3.20 GHz の上 CentOS Release 6.10 (Final) である．ルール数が $1000, 2000, 3000, 4000, 5000$ のルールリストを Class

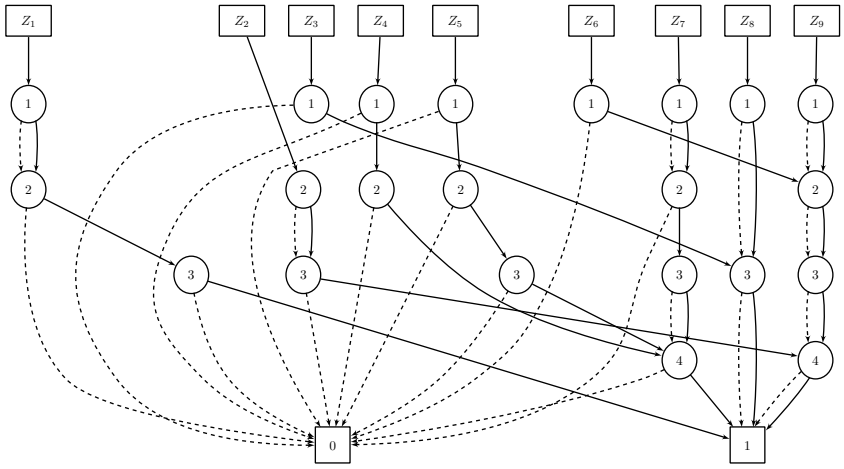

図 12 表 6 のルールリストに対する ZDDs

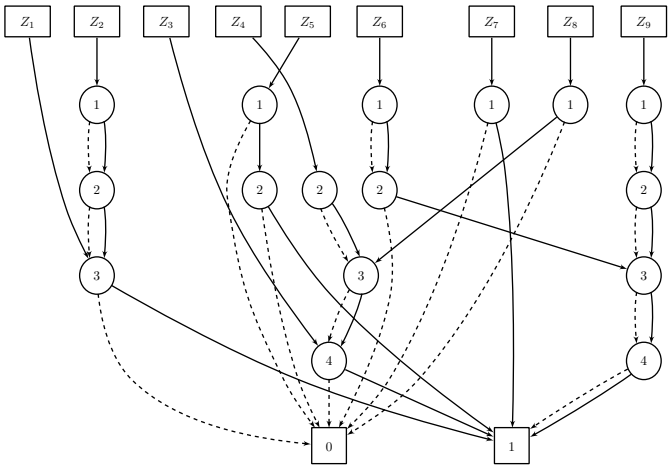

図 13 表 7 のルールリストに対する ZDDs

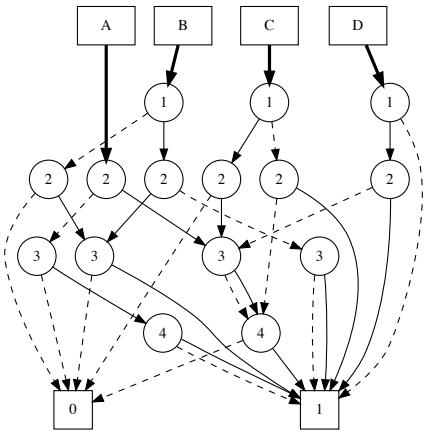

図 14 表 6,7 のポリシーに対応する ZDDs

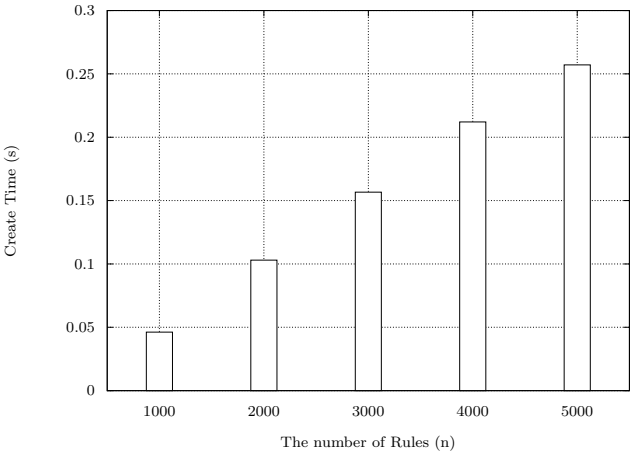

図 15 構築時間（秒）

Bench[16] を用いて生成した．入力として二つのルールリストを与え，それぞれのルールリストのポリシーを表現する ZDD の構築時間と，構築された ZDD の節点数とを計測した．

提案手法による ZDD の構築時間を図 15 に示す．ただし，単位は秒である．さらに，構築された ZDD の節点数を図 16 に示す．単位が 10^{11} であることに注意されたい．図 15,16 より，ルール数が 1000 から 5000 程度の人が目で見て管理するサイズのルールリストに対して，提案手法は 0.3 秒未満でポリシーが等価か判定できる．

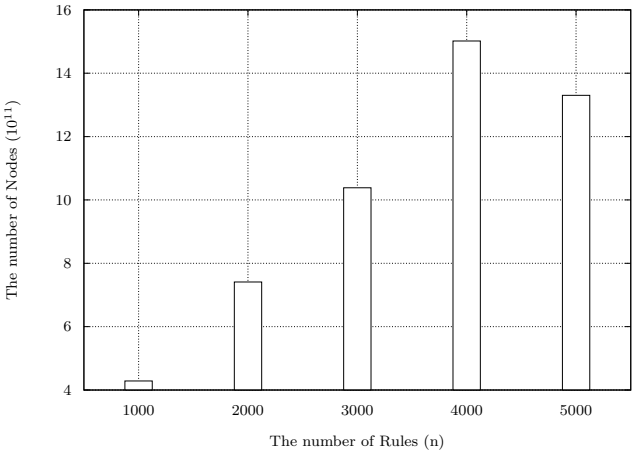

図 16 節点数（10^{11}）

7. まとめ

本稿では，ルールの並び替え，ルールリストの再構築に際して，元のルールリストと新たに生成されたルールリストのポリシーが等しいかを確認することの必要性と，その判定問題が一般には難しいことを示した．そして，ZDD によるルールリストポリシーの等価判定法を提案し，計算機実験によりルール数 5000 のルールリストに対して提案手法が有効であることを確認した．

今後の課題は，アクションが三つ以上から成るルールリストのポリシー等価判定法の有効性を確かめることと，より多くのルール数から成るルールリストに対して，提案手法の有効性を確かめることである．

参考文献

[1] Rivest, R. L.: Learning Decision Lists, *Mach. Learn.*, Vol. 2, No. 3, pp. 229–246 (online), DOI: 10.1023/A:1022607331053 (1987).

[2] Hamed, H. and Al-Shaer, E.: On Autonomic Optimization of Firewall Policy Organization, *J. High Speed Netw.*, Vol. 15, No. 3, pp. 209–227 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2692141.2692143⟩ (2006).

[3] Misherghi, G., Yuan, L., Su, Z., Chuah, C. N. and Chen, H.: A general framework for benchmarking firewall optimization techniques, *IEEE Transactions on Network and Service Management*, Vol. 5, No. 4, pp. 227–238 (online), DOI: 10.1109/TNSM.2009.041104 (2008).

[4] Tapdiya, A. and Fulp, E.: Towards Optimal Firewall Rule Ordering Utilizing Directed Acyclical Graphs, *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, pp. 1–6 (online), DOI: 10.1109/ICCCN.2009.5235232 (2009).

[5] TANAKA, K., MIKAWA, K. and HIKIN, M.: A Heuristic Algorithm for Reconstructing a Packet Filter with Dependent Rules, *IEICE Trans. Commun.*, Vol. 96, No. 1, pp. 155–162 (online), DOI: 10.1587/transcom.E96.B.155 (2013).

[6] Tanaka, K., Mikawa, K. and Takeyama, K.: Optimization of packet filter with maintenance of rule dependencies, *IEICE Communications Express*, Vol. 2, No. 2, pp. 80–85 (online), DOI: 10.1587/comex.2.80 (2013).

[7] Mohan, R., Yazidi, A., Feng, B. and Oommen, B. J.: Dynamic Ordering of Firewall Rules Using a Novel Swapping Window-based Paradigm, *Proceedings of the 6th International Conference on Communication and Network Security*, ICCNS '16, New York, NY, USA, ACM, pp. 11–20 (online), DOI: 10.1145/3017971.3017975 (2016).

[8] 日景喬一，山田敏規：D-1-6 ルール間の依存関係を保持したファイアウォールの負荷最小化のためのアルゴリズム (D-1. コンピュテーション, 一般セッション)，電子情報通信学会総合大会講演論文集，Vol. 2016, No. 1, p. 6 (2016).

[9] 渕野　敬，原田崇司，田中　賢，三河賢治：重み平均に基づくペアリングによるルール並び替え法 (回路とシステム)，電子情報通信学会技術研究報告 = IEICE technical report : 信学技報， Vol. 118, No. 295, pp. 31–36 (2018).

[10] 原田崇司，田中　賢，三河賢治：包含関係に限定したルールリスト再構築 (回路とシステム)，電子情報通信学会技術研究報告 = IEICE technical report : 信学技報， Vol. 118, No. 82, pp. 93–98（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/40021586334/⟩ (2018).

[11] Guijarro, D., Lavn, V. and Raghavan, V.: Monotone term decision lists, *Theoretical Computer Science*, Vol. 259, No. 1, pp. 549 – 575 (online), DOI: https://doi.org/10.1016/S0304-3975(00)00043-8 (2001).

[12] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *Design Automation, 1993. 30th Conference on*, pp. 272–277 (1993).

[13] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677–691 (online), DOI: 10.1109/TC.1986.1676819 (1986).

[14] Akers, S. B.: Binary Decision Diagrams, *IEEE Transactions on Computers*, Vol. C-27, No. 6, pp. 509–516 (online), DOI: 10.1109/TC.1978.1675141 (1978).

[15] Somenzi, F.: CUDD Package, `http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf`.

[16] Taylor, D. E. and Turner, J. S.: ClassBench: A Packet Classification Benchmark, *IEEE/ACM Trans. Netw.*, Vol. 15, No. 3, pp. 499–511 (online), DOI: 10.1109/TNET.2007.893156 (2007).