2004－DBS－134　(I)　(18)
2004／7／13

XML 文書のアクセス制御ポリシーの簡略化について

王波[1]　チャットウィチェンチャイ　ソムチャイ[2]　岩井原瑞穂[1]

1:　京都大学大学院情報学研究科社会情報学専攻
2: 県立長崎シーボルト大学国際情報学部
e-mail:　wang@db.soc.i.kyoto-u.ac.jp, somchaic@sun.ac.jp, iwaihara@i.kyoto-u.ac.jp

アブストラクト

本稿では，XML 文書に対するアクセス制御について，文書の変換や合成を行った際に，アクセス制御の権限判定が返還前と同じになるように，変換後のアクセス制御ポリシーを求める問題について，新たにできるだけ簡略化されたポリシーを求める問題について考察する．具体的には，部分木全体に同じ権限判定値を与える cascade という設定を最適化することにより，ポリシーを構成するルール数を最小化する問題を検討し，線形時間でルール数最小のポリシー求めるアルゴリズムについて述べる．

On Simplification of Access Authorization Policies for XML Documents

Bo Wang[1]　Somchai Chatvichienchai[2]　Mizuho Iwaihara[1]

1: Dept. Social Informatics, Kyoto University
2: Department of Info-Media, Siebold University of Nagasaki
e-mail:　wang@db.soc.i.kyoto-u.ac.jp, somchaic@sun.ac.jp, iwaihara@i.kyoto-u.ac.jp

Abstract.

XML documents are transformed or synthesized into a new set of XML documents during information exchange.　Access control policies on XML documents must also be translated into new policies so that the same authorization decision is made after the transformation.　Regarding this problem, we discuss simplifying policies after translation for efficient policy processing.　In particular, we discuss optimizing cascade settings on rules, and show a linear-time algorithm finding a　policy having a minimum number of policies.

1. 研究の背景

近年，XML による情報流通が活発になるにつれ，XML 文書の情報を保護するためのアクセス制御が重要となってきている．XML 文書内のデータにおけるきめの細かいアクセス制御を行うアクセス制御は，XML 文書の構造・内容に直接的にアクセスの制約をかける．ゆえに，アクセス権限判定ポリシーは XML 文書の構造・内容に深く関わる．XML 文書の構造は，アプリケーションの拡張や組織間のデータ交換など様々な理由で変わる傾向があるので，新しい構造に適合するように権限判定を修正しなければならない．これまで著者らは XML 文書のアクセス制御ポリシーを変換前と変換後で同じ効果を持つような，変換後のポリシーを求める手法を開発してきた[3][4][5]．しかし変換後のポリシーの品質として，できるだけ簡略化されたポリシーを求める問題は未解決であった．本稿では，文書変換後のポリシーを簡略化する手法として，部分木全体に同じ権限判定値を与える cascade という設定を最適化することにより，できるだけルール数の少ないポリシーを求めるアルゴリズムについて検討する．衝突解消のアルゴリズムとして，Deny-overrides, Permit-overrides, First-applicable のそれぞれが指定された場合に，ルール数を最小化するアルゴリズムを述べる．

以下，本稿は２節において XML のアクセス制御言語である XACML を述べ，３節において XML 文書変換におけるアクセス制御ポリシーの保存を述べる．４節では，アクセス制御ポリシーの簡略化アルゴリズムについて述べ，５節はまとめである．

2. XACML(eXtensible Access Control Markup Language)

XACML[8]は，XML 文書に対するアクセス権を設定するための仕様である．セキュリティ管理者は XACML 言語を利用して，ポリシー（あるリソースに対して，「だれ」がどのような「権限」で「どこ」にアクセスできるのか）を定義することができる．主要なトップレベル要素は PolicySet（ポリシー集合）で，これは他の PolicySet 要素や Policy（ポリシー）要素を 1 つにまとめたものである．Policy 要素は主に Target（対象），Rule （ルール），Obligation（責務）の各要素から構成される．

Target 要素は，要求されたリソースと適用可能な Policy を関連付けるのに使用される． PolicySet，Policy，または Rule をリソースに適用できるためには，要求の Subject（サブジェクト），Resource（リソース），または Action（アクション）が一定の条件を満たす必要があり，Target 要素にはそうした条件が記述される．Target 要素には，Policy のインデックス付け/ルックアップを効率化するための組み込みメカニズムが用意されている．

アクセス決定に適用できるポリシーが複数見つかる場合（また，単一のポリシーに複数の Rule が含まれている場合）があるので，結合アルゴリズムを用いて，複数の判定結果が単一の決定に一本化される．ルール結合アルゴリズムとしては以下の 3 つのアルゴリズムが規定されている．

・Deny-overrides　(拒否優先)

ポリシー内のすべてのルールについて，どれか１つのルールの Effect(判定値)が deny であれば結果は deny とする．すべてのルールが permit，あるいは幾つかのルールが permit で残りのルールすべてが NotApplicable の場合のみ結果を permit とする．

・Permit-overrides (許可優先)

ポリシー内のすべてのルールについて，どれか１つのルールの Effect が permit であれば結果は permit とする．すべてのルールが deny，あるいは幾つかのルールが deny で残りのルールすべてが NotApplicable の場合のみ結果を deny とする．

・First-applicable　(先行優先)

ポリシー内のすべてのルールについて順番に評価して，対象<Target>がマッチした場合，以後の<Rule>の評価を中断し <Target>の評価結果を Match とし，次に Condition を評価し，これが True なら結果は Effect で指定された permit または deny とする．

ルール結合アルゴリズムはルールを評価するとき，上記の permit または deny の決定以外に，要求 Context が用意されたルールにすべてマッチしなければ評価結果は NotApplicable を，評価の過程で構文エラーとなった場合 Indeterminate を返す．

３． XML 文書の変換におけるアクセス制御ポリシーの保存

XML 文書の変換の前後において，アクセス制御ポリシーの等価性を保存する問題についてまとめる．アクセス制御ポリシーの保存問題とは，XML 文書から XSLT 等により新たな XML 文書を導出した場合に，各ノードに対するアクセス制御の可否の情報を保存するように，変換後のアクセス制御ポリシーを設計するというものである．XML 文書は，要素をノードとして持つ木構造で表され，XML 文書の変換とは，ある木構造を別の木構造に変換するものである．アクセス制御ポリシー保存の観点からは，アクセス制御の可否に関する変換にのみ着目すればよい．

ここで文書の要素に含まれるテキストに対する変換も行われ，１つの要素のテキストがコピーや分割等により複数の要素に変換され，また複数の要素が変換後に１つの要素にまとめられることも考えられる．このとき，アクセスの可否が異なる要素どうしを１つの要素にまとめる場合，アクセス制御の衝突がおきる．このように，文書変換で生じる衝突については，**変換時衝突解消アルゴリズム**として Deny-overrides あるいは Permit-overrides が設定されるものとする． アクセス制御ポリシー変換のフレームワークを図１に示す．

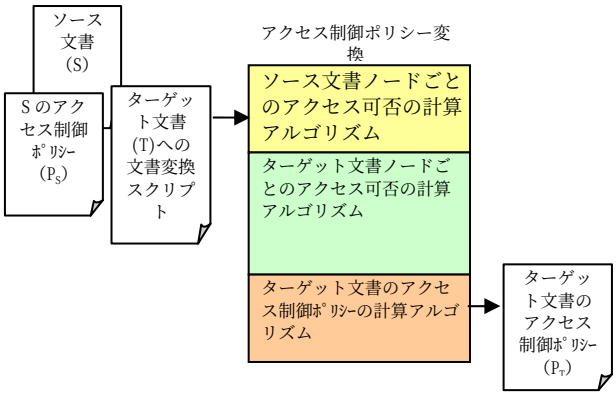

図１アクセス制御ポリシー変換のフレームワーク

XML 文書変換に対する変換を R とする．下記は，XML 文書インスタンスに対するアクセス制御ポリシーの変換を行うアルゴリズムである[3][5].

アルゴリズム１.

1. ソース文書 S について，その全要素に対しアクセス制御ポリシーP_Sを適用し，要素ごとのアクセスの可否を計算し，その結果を要素の新たな属性値として付加する．各属性値に対しても，P_S を適用しその結果を新たな属性値として付加する．（ここで新たに加える属性値は，文書変換 R に影響を与えないようにしなければならない）
2. ソース文書 S に変換 R を適用し，その結果をターゲット文書 t とする．
3. ターゲット文書 T について， その全要素の権限判定を 1.で付加した属性値（これを**継承した権限判定**

とする）から変換時衝突解消アルゴリズムに従って計算する．
4. ターゲット文書 T に新たに加わった要素に対する権限判定を計算する．
5. ターゲット文書 T のある要素 e について，T の根から e までのパスを p とし，e の権限判定を a とする．このとき,<p,a,->を T のアクセス制御ルール r とする．T の全要素 e に対して得られるルール r の集合を，ターゲット文書 T のアクセス制御ポリシーP_Tとする．

上記アルゴリズム１で生成されるアクセス制御ルールは，XPath 式として，文書インスタンスの各要素を特定するものが必要であり，例えば{child, succeeding}の組み合わせが可能である．

アルゴリズム１で計算したアクセス制御ポリシーP_Tでは，ターゲット文書 T の任意の要素 e に対し，以下の権限判定が得られるのは自明である．
1. ソース文書 S から継承した権限判定が１つである（変換時の衝突がない）要素 e の権限判定は，要素 e のもととなったソース文書 S の要素と同じである(ポリシーが保存される).
2. ソース文書 S から継承した権限判定が複数ある（変換時の衝突が存在する）要素 e は，変換時衝突解消アルゴリズムが定める権限判定となる．
3. 継承した権限判定を持たない（新たに加えられた）要素 e は，T で新たに定義される権限判定を持つ．

アルゴリズム１で得られるアクセス制御ポリシーP_Tは，変換 R に対し，権限判定が保存されるという望ましい性質を持つ．しかし P_Tを構成する各ルールは，T の要素１つのみの権限判定を与えるため，P_Tは T の要素数に比例する数のルールを必要とし冗長であるため，P_T による権限判定のコストが増大すると考えられる．次節では P_Tの簡略化について考察する．

4. アクセス制御ポリシーの簡略化

4.1 cascade による簡略化

アクセス制御ルールに用いる XPath 式において，ターゲットとなる要素とそれを根とする部分木に同じ権限判定値を設定することは有用であり，多くのアクセス制御モデルで採用されている[2][6]．本稿ではこれを cascade と呼ぶことにし，cascade の設定を最適化することにより，P_Tをできるだけ簡略化されたものにするアルゴリズムについて検討する．cascade を付加するとは，アルゴリズム１で求まる XPath 式においてターゲットとなる要素に descendant-or-self を設定することに相当する．そして cascade を持たないポリシーP_Tを入力とし，ルール数が最小となるように cascade を設定することを目標とする．さらに以下を仮定する．

(仮定１) 取り扱う文書 D はすべて，アクセス制御ポリシーP を適用することにより，D の全要素について deny または permit の評価結果が得られる（NotApplicable となる要素は存在しない）．

（仮定２）ルール数最小化として，cascade の設定を各ノードで行うかどうかのみを判断するものとする．

XPath 式集合の簡略化としては，他にスキーマを考慮する方法なども考えられるが，今後の課題とする．

ある部分木全体が deny あるいは permit のいずれか片方のみならば，１つのルールですむが，異なる権限判定が子孫に含まれるときは，別のより詳細なルールが必要である．

記号	意味	ルール数
n	ノード e にはルールを設定しない.	0
+	ノード e に権限判定値が a(e)で cascade を行うルールを設定する.	1
-	ノード e に権限判定値が a(e)で cascade を行なわないルールを設定する.	1
±	ノード e に権限判定値が a(e)で cascade を行なわないルール，次に権限判定値が a(e)の反対で cascade を行うルールの２つを設定する（e 自身には前者，e の子孫には後者のルールが適用される）.	2

表１　ノードに設定するルールの記号

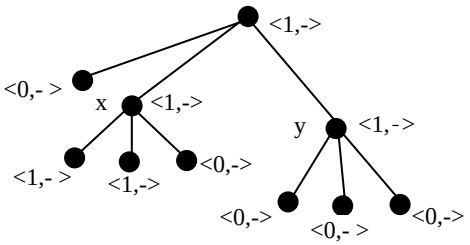

図２ First-applicable, cascade 用いない：ルール数 10

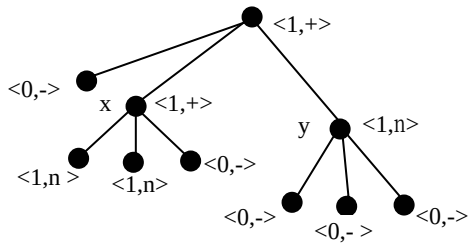

図３　First-applicable, cascade 用いる：ルール数 7

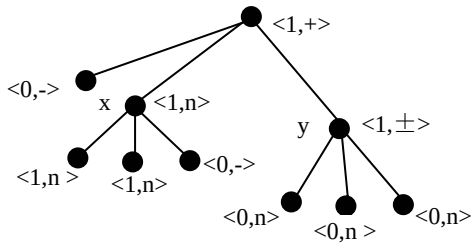

図４　First-applicable, cascade 最適化：ルール数 5

cascade を用いることにより，ルール数がどのように削減されるかを例により示す．表１は，各ノードごとに設定するアクセス制御ルールを４つの場合に分けたものである．ノード e にはあらかじめ権限判定値 a(e) ∈{permit, deny}が与えられているものとする．n は e にルールを設定しない（祖先から cascade されるものを利用する），+は a(e)の値を e に設定し，しかもそれを子孫に cascade する，-は a(e)の値を e に設定するが子孫には cascade しない，±は a(e)の値を e に設定するが，子孫には a(e)とは反転させた権限判定値を cascade させるというものである．１つのノードに対するルールの設定方法は，冗長なものを除くとこの４通りのみである．

図２には，与えられた文書木と各ノードの権限判定値について，cascade を全く使用しないでルールを設定したものであり，計 10 個のルールを必要としている．これに対し，図２は一部 cascade を用いたものであり，たとえばノード x で権限判定値 1 を+で cascade させることにより，２つの子供のルールを省略できている．まだノード y は根から cascade される値を使用できるため，y にルールを設定する必要はない．図４はさらに cascade の最適化を行って，ルール数を 5 まで削減したものである．ノード y では，±を設定することによって，y 自身では値 1 を，y の子には 0 を cascade させることにより，簡略化している．

4.2 First-applicable における cascade の最適化

以下では，衝突解消アルゴリズムとして First-applicable を用いた場合の cascade の最適な設定を行うアルゴリズムを示す．

アルゴリズム２．

入力：各ノード e に 0 または 1 の権限判定値 a(e)が与えられた XML 文書 D

出力：D に対するポリシーP_D

1. D の根を e とする．e が子を持たないときは，rule(e) = “-“として終了する．e が子を持つ場合は，e の子の集合を{e_1,…,e_k}とする．各子 e_i について，アルゴリズム２を再帰的に適用し，e_i を根とする部分木 D_i に対するポリシーP_{Di} を求める．e_i に対するルールは rule(e_i)に格納される．
2. $cost^+ = 0$, $cost^- = 0$, $cost^{\pm} = 1$ とする．
3. for each e_i in {e_1,…,e_k} {
   if (a(e_i) = a(e)) then {
   switch rule(e_i) {
   case “+” : $cost^-$ += 1; $cost^{\pm}$+= 1;
   case “-” : $cost^-$ += 1; $cost^{\pm}$+= 1;
   case “±” : $cost^+$ += 1; $cost^-$ += 2; $cost^{\pm}$+= 1;
   }
   }
   else {
   switch rule(e_i) {
   case “+” : $cost^+$ += 1; $cost^-$ += 1;
   case “-” : $cost^+$ += 1; $cost^-$ += 1;
   case “±” : $cost^+$ += 1; $cost^-$ += 2; $cost^{\pm}$+= 2;
   }
   }
4. if ($cost^+ = \min(cost^+, cost^-, cost^{\pm})$) {
   rule(e) = “+”;
   for each e_i in {e_1,…,e_k} {
   if (a(e_i) = a(e)) then {
   if (rule(e_i) = “+”) OR (rule(e_i) = “-”) then
   rule(e_i) = “n”;
   }
   else {
   if (rule(e_i) = “±”) then
   rule(e_i) = “-”;
   }
   }
   else if ($cost^{\pm} = \min(cost^+, cost^-, cost^{\pm})$) then {
   rule(e) = “±”;
   for each e_i in {e_1,…,e_k} {
   if (a(e_i) ≠ a(e)) then {
   if (rule(e_i) = “+”) OR (rule(e_i) = “-”) then
   rule(e_i) = “n”;
   }
   }
   }
   else {
   rule(e) = “-”;
   }
   }
return;

[定理１] アルゴリズム２は入力の XML 文書 D のノード数|D|について O(|D|)時間で停止し，First-applicable の衝突解消アルゴリズムを用いる場合のルール数最小のポリシーP_Dを求める．

[証明] アルゴリズム２において，各ノード e について高々１回のみ部分木の根として再起呼び出しが行われ，１回の呼び出しでは e とその子の数に比例した計算時間であるため，全体で O(|D|)時間であることがわかる．

ルールの合計数については，表１に示す各ノードのルールの数を合計したものである．以下，ルール数の最小性について D のノード数に関する帰納法を用いる．

(1) D が１つのノードのみからなる場合は，１つのルール“-“がアルゴリズムにより得られ，また実際にこのルールが必要であるため，最小性は成り立つ．

(2) D よりノード数が小さい任意の文書 D’について，アルゴリズム２は最小のルール集合を求めると仮定する．このとき，D の根 e の子 e_1,…,e_k それぞれを根とする部分木を D_1,…,D_k とする．アルゴリズム２は，各部分木のルール集合をステップ 1 で再帰的に求める．ステップ 3 では，e と e_1,…,e_k の間で，e のルールを”+”，”-”，”±”のそれぞれのルールを設定したときに，省略できるルールを除いたルール数の合計

を計算している．ステップ 4 では，ステップ 3 で求めたルールの合計数を最小とするルールを e に設定する．そして部分木の根 $e_1,\ldots,e_k$ それぞれについて省略できるルールを１つ除くかまたはそのままにする．ここで，もしアルゴリズム２よりもルール数が小さいポリシーP'_D が存在したとする．もし，P'_D の各部分木 $D_1,\ldots,D_k$ が P_D と一致するならば，根 e のルール数が P_D より小さいはずであるが，rule(e)は e と $e_1,\ldots,e_k$ の間で最小に選んであるためそれは有り得ない．すると，部分木 $D_1,\ldots,D_k$ のいずれかで P'_D は P_D より小さな解を求めているはずである．ところが帰納法の仮定により P_D は $D_1,\ldots,D_k$ の部分で最小のルール数を求めて，これからさらに部分木の根 $e_1,\ldots,e_k$ のルールを削減しているか同じであるため，これも有り得ない．よって P_D はルール数が最小である．□

4.2 Deny-overrides および Permit-overrides の場合

Deny-overrides または Permit-overrides を衝突解消アルゴリズムとして用いた場合のルール簡略化について考察する． Permit-overrides は Deny-overrides と同様な議論が可能であるため，一般性を失うことなく Deny-overrides について述べる．まず，Deny-overrides を First-applicable で模倣することについて検討する．First-applicable はルールの順序により，先行するルールの判定値が優先するというものであるから，Deny-overrides を模倣するには，Deny を行なうルールを先行させるように，ルールの順序付けを行なえばよいと考えられる．より一般的には，以下の性質が成り立つ.

[定理２] 与えられた XML 文書 D およびポリシー P_D について，ルール数が P_D と等しい次の条件を満たすポリシーP'_D が存在する．すなわち任意のアクセス要求について Deny-overrides (同様に Permit-overrides)アルゴリズムで P_D を評価した権限判定値と，First-applicable で P'_D を評価した権限判定値は常に一致する．

[証明] P_D のルール集合について，D の各ノード e について，permit を与えるルール集合を p(e), deny を与えるルール集合を d(e)とする．このとき，以下のアルゴリズムで出力されるポリシーを P'_D とする．

1. D の文書木を後順(post-order)で走査する．そして各要素 e が走査されたとき，d(e)の各ルールを任意の順で出力する．
2. 再び D の文書木を後順(post-order)で走査する．そして，各要素 e が走査されたとき，p(e)の各ルールを任意の順で出力する．

上記で出力されるルールの順序は，最初に deny を与えるルールが葉から根の順序で出力され，次に permit を与えるルールが葉から根の順序で出力される．また P_D を並べ替えただけであるからルール数は変わらない．ここでもしある権限判定が P_D において deny であるならば，P_D の中のあるルール r が deny の値を与えるはずである．すると，P'_D においても r は deny の値を持つ．P'_D においては，r よりも前に permit を与えるルールは存在しないため，P'_D においても First-applicable のアルゴリズムにより，deny の判定が行なわれる．P'_D は後順でルールが出力されているため，cascade の処理も正しく行なわれる．P'_D における判定値が permit の場合も同様な議論で P'_D も permit を出力することが確かめられる．Permit-overrides についても，上記の d(e)と p(e)，deny と permit をそれぞれ入れ替えればよい． □

定理２より，ルール数の観点からは，First-applicable を用いた方が Deny-overrides や Permit-overrides よりも常に等しいか少ないルール数が得られることが分かった．しかし，権限判定を行なうコストの観点からは，例えば Deny-overrides の場合に文書木を根から判定対象のノードに降下しながら各ルールの評価値を求めていった場合，途中でも deny の値を与えるルールが見つかれば，そこで木の探索を終了して，権限判定は deny であると結論することができる．このため祖先で deny を行なうルールが多い場合など， Deny-overrides が有利な局面も考えられる．以下では，Deny-overrides（および Permit-override）でのルールの最小化を行なうアルゴリズムについて考察する.

Deny-overrides では，deny をあるノードから cascade させた場合，そのノードからの部分木全体が deny になる．そのため，もしその部分木中に permit のノードが１つでもあるならば，cascade は使用できないことになる．また，アルゴリズム２で用いた±については，a(e)=permit のノード e については，deny が優先されるため±では deny に判定されてしまい，用いることができない．以上を考慮すると以下のアルゴリズム 3 を構成することができる．ここでは permit と deny を交換すれば Permit-overrides 用に変更できる.

アルゴリズム 3.

入力：各ノード e に 0 または 1 の権限判定値 a(e)が与えられた XML 文書 D

出力：D に対するポリシーP_D

1. D の根を e とする．e が子を持たないときは，rule(e) = “-“とする．さらに a(e)= deny のとき，部分木全体が deny であることを示すフラグとして uniform(e)=”yes”とし，a(e)=permit のときは uniform(e)=”no”と設定して終了する．e が子を持つ場合は，e の子の集合を$\{e_1,\ldots,e_k\}$とする．各子 e_i について，アルゴリズム３を再帰的に適用し，e_i を根とする部分木 D_i に対するポリシーP_{Di} を求める．e_i に対するルールは rule(e_i)に格納される．
2. if a(e) = deny AND
   uniform(e_i)=”yes” holds for every e_i in $\{e_1,\ldots,e_k\}$
   then {
   rule(e) = “+”; uniform(e)=”yes”;
   return;
   }
   }
3. uniform(e) = “no”;
   $cost^+ = 0$; $cost^- = 0$; $cost^{\pm} = 1$;
4. for each e_i in $\{e_1,\ldots,e_k\}$ {
   if (a(e_i) = a(e)) then {
   switch rule(e_i) {
   case “+” : $cost^-$ += 1; $cost^{\pm}$ += 1;

```
      case “-” :  cost- += 1;  cost±+= 1;
      case “±” :  cost+ += 1; cost- += 2; cost±+= 1;
     }
    }
   else {
     switch  rule(ei)  {
      case “+” :  cost+ += 1;  cost- += 1;
      case “-” :  cost+ += 1;  cost- += 1;
      case “±” :  cost+ += 1; cost- += 2; cost±+= 2;
     }
    }
5. if (cost- = min(cost+ , cost- , cost±))  OR
    (a(e)=deny AND cost+ = min(cost+ , cost- , cost±))  OR
    (a(e)=permit AND cost± = min(cost+ , cost- , cost±))
   then {
     rule(e) = “-”;
     return;
  }
6. if (a(e)=deny then {/* ±使用不可，+使用可*/
    rule(e) = “+”;
    for each ei in {e1,…,ek}  {
     if  (a(ei) = a(e))  then {
      if  ( rule(ei)  =  “+” ) OR ( rule(ei)  =  “-” )  then
          rule(ei)  = “n”;
     }
    else {
      if  (rule(ei)  =  “±” )  then
         rule(ei)  = “-”;
     }
   return;
  }
7.    /* ±使用可，+使用不可*/
   rule(e) = “±”;
   for each ei in {e1,…,ek}  {
    if  (a(ei) ≠ a(e)) {
      if  ( rule(ei)  =  “+” ) OR ( rule(ei)  =  “-” ) then
          rule(ei)  = “n”;
      }
    }
   }
   return;
```

図５は，図２に対しアルゴリズム３を適用し Deny-overrides についてルール数を最小化した結果であり，ルール数は６になっている．以下の定理３は，定理１とほぼ同様な議論で証明することができる．

[定理３] アルゴリズム３は入力の XML 文書 D のノード数|D|について O(|D|)時間で停止し，Deny-overrides の衝突解消アルゴリズムを用いる場合のルール数最小のポリシーP_Dを求める． □

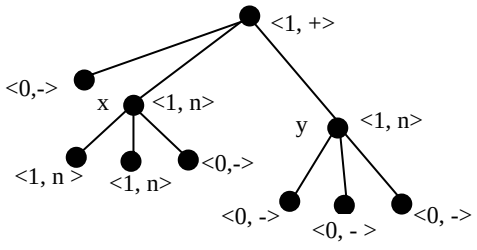

図５　Deny-overrides，cascade 最適化：ルール数 6

５．まとめ

本稿では，XML 文書集合を別のフォーマットの文書に変換したり，文書の合成を行った際に，元の文書に設定されていたアクセス制御ポリシーを保存するポリシーを変換後の文書に設定するうえで，できるだけ簡略化されたポリシーを求める最適化について考察し，cascade の最適化は線形時間で可能であることを示した．今後は，スキーマを考慮した場合など，より広い XPath のクラスの簡略化について考察する予定である．

参考文献

[1] Anutariya, C., Chatvichienchai, S., Iwaihara, M., Wuwongse, V., and Kambayashi, Y. A rule-based XML access control model. In *Int. Workshop on Rules and Rule Markup Languages for the Semantic Web* (RuleML 2003), LNCS2876, pp.35–48, Florida, USA, Oct. 2003.

[2] Bertino, E., Castano, S., Ferrari, S., and Mesiti, M. Specifying and enforcing access control policies for XML document sources. *World Wide Web*, Baltzer Science Publishers, Netherlands, 3, 2000.

[3] Chatvichienchai, S., Iwaihara, M., and Kambayashi, Y. Towards translating authorizations for transformed XML documents. In *Proc. of the 3rd International Conference on Web Information Systems Engineering* (WISE 2002), IEEE CS, pp. 291–300, Dec. 2002.

[4] Chatvichienchai, S., Iwaihara, M., and Kambayashi, Y. Translating content-based authorizations for XML documents. In *Proc. of the 4th Int. Conference on Web Information Systems Engineering (WISE 2003)*, IEEE CS, pp.103–112, Roma, Italy, Dec. 2003.

[5] Chatvichienchai, S., Iwaihara, M., and Kambayashi, Y. Authorization translation for XML document transformation. *Int. Journal of World Wide Web: Internet and Web Information Systems*, Kluwer, Vol. 7, No.1, pp. 111-138, Mar 2004.

[6] Hitchens, M. and Varadharajan, V. RBAC for XML document stores. In *Information and Communications Security, 3rd Int. Conference*, ICICS 2001, LNCS2229, pp.121–135, Nov. 2001.

[7] Kudo, M., Hada, S., Paraboschi, S., and Samarati, P. XML document security based on provisional authorization. In *Proc. of the 7th ACM Conference on Computer and Communications Security*, pp.87–96, Nov. 2000.

[8] OASIS eXtensible Access Control Markup Language Technical Committee. XACML 1.0 Committee Specification Set. http://www.oasis-open.org/committees/xacml/, Nov. 2002.

[9] Su, H., Kuno, H., and Rundensteiner, E.A. Automating the transformation of XML documents. In *Proc. of the 3rd Int. Workshop on Web Information and Data Management (WIDM 2001)*, pp.68–75, Jul 2001.