RC-006

ネットワーク侵入検知のためのパターン非依存 NFA とシストリックアルゴリズムを組み合わせた正規表現マッチングエンジン

A Regular Expression Matching Engine Based on a Systolic Algorithm with a Pattern-Independent NFA for Network Intrusion Detection

若葉 陽一　稲木 雅人　永山 忍　若林 真一
Yoichi Wakaba　Masato Inagi　Shinobu Nagayama　Shin'ichi Wakabayashi
広島市立大学大学院　情報科学研究科

あらまし—本稿では，クリーネ演算のネストに対応した，シストリックアルゴリズムに基づくパターン非依存正規表現マッチングハードウェアエンジンを提案する．パターン非依存マッチングエンジンは，ウィルスパターンの即時更新が必要となるネットワーク侵入検知システム（NIDS）に適している．ところが，既存のシストリックアルゴリズムに基づくエンジンはウィルスパターンの表現に使用されるクリーネ演算（繰り返しパターンを表現する演算）のネストを扱うことができない．そこで，我々は任意の正規表現パターンに対応したコンパクトなパターン非依存 NFA 回路を，シストリックアルゴリズムに基づくエンジンに組み込み，クリーネ演算のネストを扱うことのできる効率的な正規表現マッチングエンジンを提案する．実験により，提案エンジンは実装時の面積効率が高く，スループットは 2.17Gbps と高速に動作することを確認した．

1　はじめに

ネットワークセキュリティは今日の情報化社会において非常に重要となっている．有力なセキュリティ対策のひとつであるネットワーク侵入検知システム（NIDS）は，ネットワークを流れるパケットの監視を行い，コンピュータウィルスなどの不正な侵入をユーザに通知するシステムである．NIDS はあらかじめ正規表現で記述されたウィルスパターンと，ネットワークを流れるパケットのペイロードとの間で正規表現マッチングを行うことでウィルスを検知する．ここで，正規表現マッチングとは，文字や文字に対する演算子からなる文字列（正規表現）で表されるパターンに一致する文字列を，入力系列から検索する操作である．

一般的に，NIDS はソフトウェアで実現される．NIDS のソフトウェア実装として Snort [1] がよく知られており，そのウィルスパターン（Snort ルール）は拡張正規表現 [2] で記述されている．しかし，近年ネットワークの伝送速度は高速化しており（例：ギガビットイーサーネット），ネットワークを流れる全データをソフトウェアによる NIDS でリアルタイムに監視することは難しくなりつつある．

近年，NIDS の高速化のため，多くの正規表現マッチングハードウェアエンジンが提案されている [3–10]．正規表現マッチングのハードウェア実装として非決定性有限オートマトン（NFA [11]）を使ったものが主流である [3–8]．この種のハードウェア実装では，まず正規表現で記述されたウィルスパターンを等価な NFA に変換し，その NFA を FPGA などの再構成可能デバイスに実装することで，正規表現マッチングを実現している．ウィルスパターンを NFA として実装するハードウェアエンジンはコンパクトな回路で高速なマッチングを実現することができるが，回路がパターンに依存しており，パターンが更新されるたびに回路の再合成と再構成が必要となる．ウィルスパターンの更新が頻繁に起こる NIDS において，このエンジンは新たなウィルスに瞬時に対応できないという問題がある．

NIDS に適したパターン非依存正規表現マッチングハードウェアエンジンとしては，我々の提案するシストリックアルゴリズムに基づくエンジン [9] と，強い制約付きのパターン非依存 NFA 回路に基づくエンジン [10] が挙げられる．[9] の手法は [10] の手法ほど高速ではないが，サポートする正規表現のクラスが [10] の手法よりも大きく，Snort ルール内のほとんどのパターンを扱うことができるという利点があり，NIDS に特に適している．しかしながら，[9] のエンジンで扱えないウィルスパターンもいくつか存在する．その一つとして，パターンの繰り返しを表現するクリーネ演算のネストを含むパターンがある．

本稿で，我々はクリーネ演算のネストを扱うことができる，シストリックアルゴリズムに基づくパターン非依存マッチングハードウェアエンジンを提案する．提案ハードウェアはコンパクトなパターン非依存 NFA 回路を搭載しており，これによりクリーネ演算のネストの処理を行う．この NFA 回路は最大パターン長の二乗の回路サイズを持つが，パターンの大部分をシストリックエンジンで処理し，クリーネ演算のネスト部分のみを NFA 回路で処理することによって，NFA 回路で処理するパターン長を抑え，コンパクトなエンジンを実現する．実験により，提案エンジンは高面積効率で実装できることを確認した．また，スループットは 2.17Gbps であり，ギガビットイーサネットに対応可能な処理速度であることを確認した．

本稿の構成は以下のとおりである．2 章では準備として，正規表現，NFA，シストリックアルゴリズムに基づく正規表現マッチングハードウェアエンジン，NIDS のシステム構成について説明し，3，4 章では，クリーネ演算のネストに対応可能なハードウェアエンジンを提案する．5 章で実験的評価を述べ，最後に本稿をまとめる．

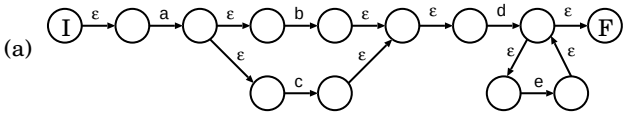

図 1: パターン “$a(b|c)de^*$” に対する NFA

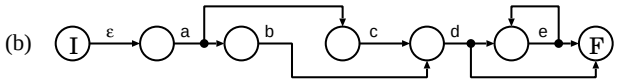

図 2: パターン “$a(b|c)de^*$” に対する UTC NFA

2 準備

2.1 正規表現

正規表現 [11] とは，文字列の集合を一つの文字列で表現する方法，およびそのような文字列自体を言う．正規表現は，ストリングマッチングを行う際のパターンとしてよく用いられる．正規表現では 3 つの基本演算（ユニオン（|），連接（·），クリーネ演算（*））を使い，文字列の集合を表現する．以下にこれらの演算子の定義を示す．

1. $R_1|R_2 = \mathrm{Q}_1 \cup \mathrm{Q}_2$
2. $R_1R_2 = \mathrm{Q}_1\mathrm{Q}_2 = \{s_1s_2 | s_1 \in \mathrm{Q}_1, s_2 \in \mathrm{Q}_2\}$
3. $R^* = \{\varepsilon\} \cup \mathrm{Q}^1 \cup \mathrm{Q}^2 \cup \cdots$

ここで，ε は文字をもたない文字列（“”）と一致する空語と呼ばれる特殊文字，R, R_1, R_2 は任意の正規表現，Q，Q_1，Q_2 はそれぞれ R, R_1, R_2 が表す文字列の集合である．また正規表現には括弧 (“(”，“)”) が用いられ，演算子の適用される範囲や優先度を指定できる．クリーネ演算のネストとはクリーネ演算 R^* において R の中にさらにクリーネ演算を含む正規表現である．本稿ではこのようなクリーネのネストを含むパターン R^* を KK-項と呼ぶ．KK-項の例として $(ab^*c)^*$ や $(a(bc)^*d)^*$ などがあげられる．

2.2 非決定性有限オートマトン（NFA）

NFA [11] は入力文字列を読み込み，それを受理するかしないかを判定する抽象機械である．NFA は有限個の状態をもっており，初期状態から入力文字に従って状態遷移を行い，受理状態に到達可能かどうかで，入力系列が受理されたかを判定する．NFA の特徴として，ある状態から同一入力文字で複数の状態に遷移可能であり，さらに入力文字を必要とせず状態遷移可能な ε 遷移が許される．正規表現と等価な NFA を正規表現から直接構築する事が出来るため，正規表現マッチングではよく用いられる．正規表現パターン “$a(b|c)de^*$” に対する NFA を図 1 に示す．図 1 において，I は初期状態，F は受理状態，ε は ε 遷移を示す．

初期状態以外の状態からの ε 遷移を許さず，各状態から遷移可能な状態への遷移条件が同一文字である NFA を，本稿では unique transition character NFA (UTC NFA) と呼ぶ．UTC NFA は [4] で提案されているパターン依存 NFA 回路に対応しており，一般的な NFA と等価な受理集合を持っている．図 1 の NFA と等価な UTC NFA を図 2 に示す．文字数 S の正規表現パターンを構築する場合，状態数は S+2 となる．

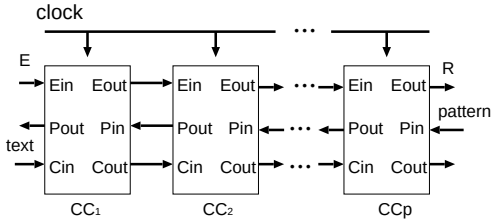

図 3: シストリックアルゴリズムに基づくマッチングエンジン [9] の構成

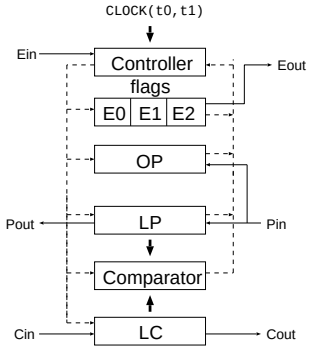

図 4: CC の基本構成

2.3 シストリックアルゴリズムに基づく正規表現マッチングハードウェアエンジン

提案ハードウェアエンジンはここで説明するシストリックアルゴリズムに基づく正規表現マッチングハードウェアエンジン [9] に基づいている．シストリックアルゴリズムとは，セルと呼ばれる基本回路をアレイ状，グリッド状などに接続した回路上で，クロックに同期してデータを流し，計算を行うアルゴリズムの総称である．[9] で提案されているマッチングエンジンは 1 次元アレイ状に比較セル（CC）を接続した構成となっている（図 3）．各 CC は 1 文字単位のマッチングを行う同期回路である．マッチング開始前にパターンを右端の CC から入力し，各 CC にパターン文字をあらかじめセットする．そして，左端の CC から 1 クロックごとに入力系列を 1 文ずつ入力し，各 CC で 1 文字単位のマッチングが並列に行われる．各 CC の比較結果は右隣の CC に伝えられる．右端の CC によるマッチング成功出力は，入力系列の部分文字列がパターンと一致したことを示す．この構成により，各 CC にセットされているパターン文字を書き換える事で，パターンの更新を瞬時に行うことができる．

次に各 CC の基本構成（図 4）について説明する．各 CC は LP，LC，OP，E1，E2，E0 レジスタと制御回路 Controller，比較器からなる．LP と LC は 8 ビットレジスタで，LP はパターンの 1 文字を記憶し，LC は入力系列内の 1 文字を記憶する．OP は正規表現演算（ε や $*$ など）を示すための n ビットレジスタである．ここで n はこのエンジンがサポートする正規表現演算子の種類数となる．全 CC は単一クロック信号に同期して動作する．マッチング開始信号（Ein=true）が入力されると，正規表現演算子を考慮しつつ LP と LC の比較が行われる．Ein=true は CC で比較が行われる入力系列内の文字の 1 つ前までの文字がそこまでのパターンと一致したことを意味する．LP と LC の

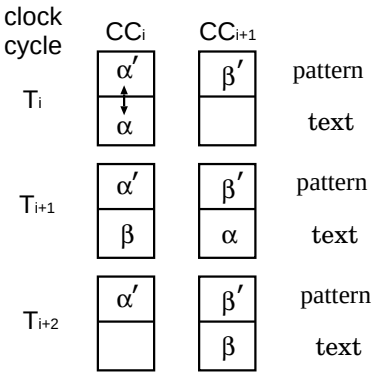

図 5: 各 CC における入力系列の流れ

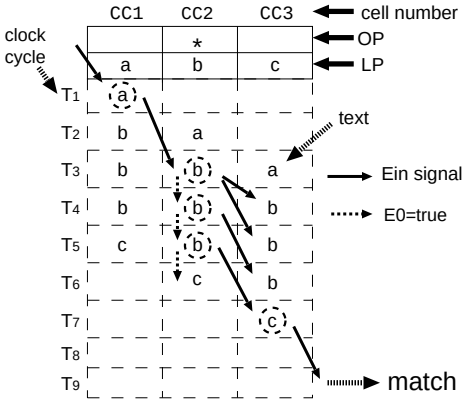

図 6: 各 CC の動作例

比較が成功した場合は 1 クロック後に E1 の値が true となる．さらに 1 クロック後に E1 の値は E2 に伝えられ，E2 の値はそのまま Eout の値となる．以降では，Eout=true を Eout を出力すると言い換えている．アーキテクチャの特性上，入力系列内のある任意の文字 α の次の文字 β が，文字 α が一致したパターン文字 α' の次のパターン文字 β' を持つ CC （i.e., α' を持つ CC の右隣の CC）に到着するまで 2 クロック必要であることから（図 5），各 CC は 2 クロック後に比較結果を右隣の CC に伝える必要があり，出力タイミングを遅らせるために E1 と E2 の 2 つのレジスタを用いている．レジスタ E0 は 1 文字に対するクリーネ演算のために用いられる．E0 は OP に $*$ の演算子が記憶されているとき，マッチング開始信号（Ein=true）が入力されると E0=true となる．これは，その CC での文字の比較が不一致となるまで保持され，E0=true である限りマッチングが繰り返し実行される．比較が不一致となった後は，次の開始信号が入力されるまで，マッチングは実行されない．

図 6 にパターン “ab^*c” と入力系列 “abbbc” に対する各 CC の動作例を示す．図は各クロックサイクルにおける各 CC の出力信号と各 CC が保持している入力系列内の 1 文字を表している（入力系列は左側から 1 クロックごとに入力されていくことに注意）．T1 クロックでは，CC1 に入力系列の 1 番目の文字 ‘a’ と Ein が入力される．このとき，CC1 で比較が行われる．ここでは比較に成功するため，2 クロック後の T3 クロックで CC2 に Eout が出力される．CC2 において T3 クロックでは，入力系列の 2 番目の文字 ‘b’ と Ein が入力される．CC2 は OP に ‘$*$’ が記憶されているため，空語とのマッチング成功として次のクロックで CC3 に Eout を出力する．また CC2 では文字の比較が不一致になるまで，文字の比較が行われるため，T3 から T5 に入力された入力系列の 2 から 4 番目の文字 ‘b’ とのマッチング成功に対して Eout が出力される．CC3 では T4 から T7 クロックに Ein が入力され，比較が行われる．T7 クロックで入力された入力系列の 5 番目の文字 ‘c’ と比較に成功するため T9 クロックで CC3 が Eout を出力する．ここで，右端の CC である CC3 が Eout を出力したことは，入力系列内の部分文字列がパターンと一致した事を示している．

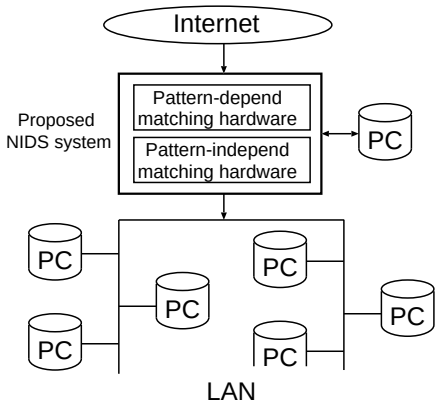

図 7: セキュリティ向上のための NIDS システム構成

CC の動作の詳細は [9] を参照されたい．[9] のエンジンにおいて，クリーネ演算 R^* において R の中にさらにクリーネ演算を含まない正規表現（“$(abc)^*$” や “$(ab|c)^*$” など）も扱う事ができる．以降，シストリックアルゴリズムに基づくハードウェアエンジンをセル回路と呼ぶ．

2.4 セキュリティ向上のための NIDS のシステム構成

NFA に基づいたパターン依存正規表現マッチングハードウェアエンジン [3–8] はコンパクトな回路で高速なマッチングを実現することができるというメリットがあるが，回路構成がウィルスパターンに依存するため，新しいウィルスに瞬時に対応できないという欠点がある．一方，シストリックアルゴリズムに基づくパターン非依存正規表現マッチングハードウェアエンジンは，新しいウィルスに瞬時に対応できるが，パターン依存 NFA 回路と比較して面積効率が悪く，多数のウィルスパターンを同時に扱うことができない．そこで互いの欠点を補うため，両方のエンジンを組み合わせた新たな NIDS のシステム構成が [12] で提案された（図 7）．

[12] で提案されたシステムでは，定常状態においては，ウィルスの検出はパターン依存エンジンで行う．新しいウィルスパターンが追加された場合，パターン非依存エンジンで新しいウィルスの検出を行い，これまでのウィルスの検知はパターン依存エンジンで行っておく．2 つのエンジンがウィルスを検知している間に，新たなウィルスパターンを追加した回路の再構成データの生成を行い，生成が終わり次第，パターン依存エンジンの更新を行う．

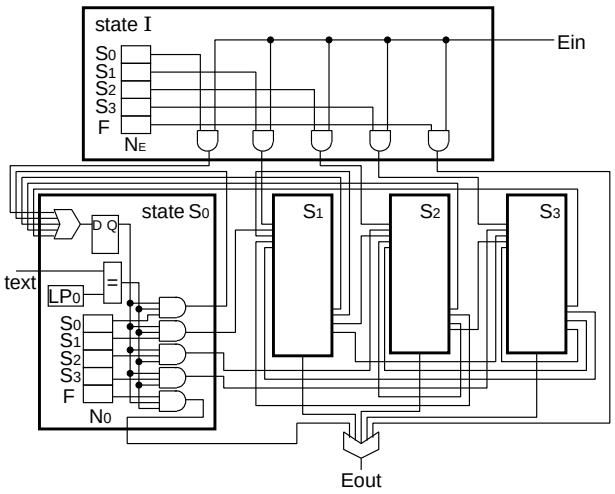

図 8: パターン非依存単文字遷移 NFA 回路

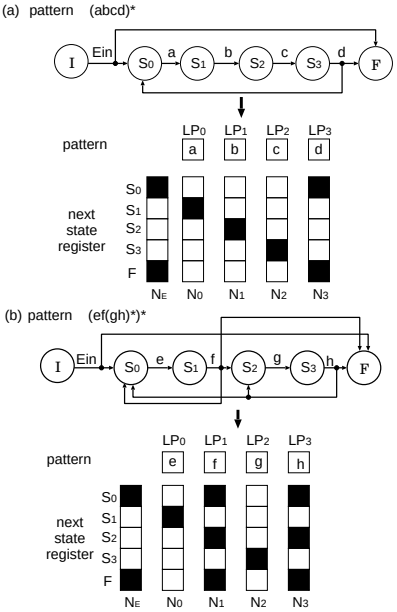

図 9: UTC NFA と次状態遷移レジスタ

このシステム構成により多くのウィルスを同時に検知することができ，かつ新しいウィルスにもすぐに対応できる．本稿で提案するパターン非依存エンジンはこのシステム構成に適用可能である．

3 パターン非依存単文字遷移 NFA 回路を導入した提案エンジン

この章ではセル回路 [9] に，以降の節で説明するパターン非依存単文字遷移 NFA 回路を導入し KK-項のマッチングを実現する．

3.1 パターン非依存単文字遷移 NFA 回路

パターン非依存単文字遷移 NFA 回路の構造を図 8 に示す．この回路は，ε 遷移を許さず，ある状態から同一入力文字のみで状態遷移を行う UTC NFA に基づく同期回路である．図 8 の回路は，演算子を除き，最大 4 文字までの任意の正規表現に対するマッチングを行う事ができる．最大 n 文字までの任意の正規表現に対するマッチングを行うことが可能な NFA 回路において，state S_i （$0 \leq i \leq n$-1） は NFA の初期状態と受理状態を除く状態に対応するモジュールである．state I は UTC NFA の初期状態に対応するモジュールである．state I への信号線 Ein は入力系列におけるマッチングを開始する位置 (文字）を伝えるもので，マッチングを開始する際に Ein=true となる．次状態遷移レジスタ N_i は状態 S_i における次状態を示す．同様に N_E は，初期状態における次状態を示すためのレジスタである（UTC NFA では，ある状態の次状態の候補が複数あったとしても遷移条件が同一なため，遷移文字が入力されたときは全ての次状態候補に遷移することに注意）．LP_i は状態 S_i から次状態に遷移するために必要な文字を記憶するためのレジスタである．LP_i と N_i，N_E は書き換え可能であり，パターン設定時にセットする．S_i 内のフリップフロップは S_i が現状態かどうかを示す．例えば正規表現パターン $(abcd)^*$ と $(ef(gh)^*)^*$ は，図 9 のように LP_i と N_E および N_i をセットする事で実現出来る．

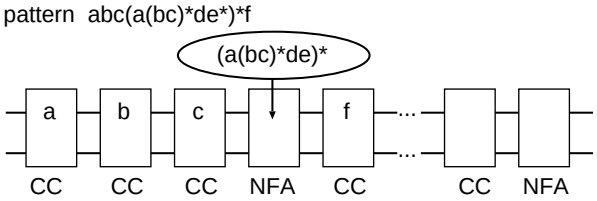

図 10: 単文字遷移 NFA 回路を導入したハードウェア構成

次に NFA 回路の動作を説明する．マッチング中は，入力系列が 1 文字ずつ入力される．モジュール S_i は入力系列内の 1 文字と LP_i にセットされたパターン文字の比較を行う．S_i で文字の比較が成功し，かつ S_i が現状態（モジュール S_i のフリップフロップが true）だった場合は，次状態遷移レジスタ N_i を参照し，N_i が示す状態に遷移する．Ein が true の場合は，N_E を参照し，初期状態からの状態遷移が行われる．次状態遷移レジスタ N_i が参照されたとき，N_i が受理状態（F）を示していた場合，NFA 回路はマッチング成功信号 Eout を出力する（Eout=true となる）．

この回路は，任意の正規表現を実現でき，柔軟性が高いという利点がある．しかし全ての状態の次状態遷移レジスタの総 bit 数は，状態数に対して二乗オーダーとなる．配線数についても同様である．そのため，対応パターン文字数を増加させると，回路が非常に複雑化し，かつ動作速度の低下を招く．

3.2 単文字遷移 NFA 回路によるセル回路拡張

効率的に KK-項を含む正規表現のマッチングを行うため，セル回路にパターン非依存単文字状態遷移 NFA 回路を導入した構成を提案する（図 10）．提案回路は比較セル CC と単文字状態遷移 NFA 回路からなり，NFA 回路は一定の割合で挿入される．NFA 回路において状態モジュール数は対象とするパターンの KK-項内の最大文字数に制限する．基本的にパターンの KK-項以外の部分に対するマッチングは従来のセル回路部分（CC）が行い，パターン非依存 NFA 回路は KK-項のみに対するマッチングを行う（図 10）．

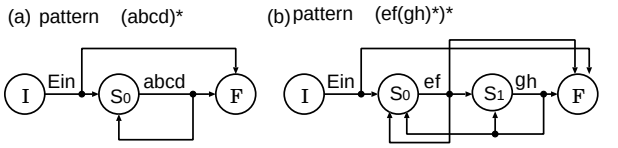

図 11: 多文字遷移 NFA

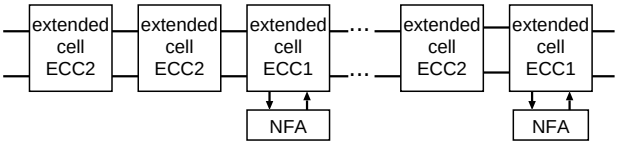

図 12: 多文字遷移のためのハードウェア構成

4 パターン非依存多文字遷移 NFA 回路を導入した提案エンジン

本章では，3 章で説明したパターン非依存単文字遷移 NFA 回路を拡張する事で高速化と効率化を図る．単文字遷移 NFA 回路の全ての状態の次状態遷移レジスタの総 bit 数は，状態数に対して二乗オーダとなる．そこで必要な状態数を削減するため多文字遷移を NFA 回路に適用する．

単文字遷移 NFA 回路の基となる UTC NFA において，次状態が 1 つのみの状態が存在する．例として，図 9（a）の S_0，S_1，S_2 や図 9（b）の S_0，S_2 は次状態が 1 つだけである．このような単純な状態遷移については，文字の単純な連接（文字列）を遷移条件とすることで，遷移元と遷移先の状態をまとめることができる（図 11）．拡張した NFA 回路では，状態遷移が単一文字に対してだけでなく，文字列に対しても行われる．そのため，本稿では，これを多文字遷移と呼ぶ．

多文字遷移を導入することで，NFA 回路が持つ状態モジュール数を減らすことができる．多文字遷移の実現のため，セル回路がパターンの KK-項以外の部分のマッチングだけでなく，KK-項部分の文字や文字列のマッチング（即ち，多文字遷移 NFA の状態遷移条件の判定）も行う．多文字遷移 NFA 回路は状態遷移に必要な文字や文字列のマッチング結果をセル回路から受け取る．その結果をもとに，多文字遷移を行い，受理状態に達したどうかで KK-項のマッチング成功を判定する．セル回路における文字列比較は数クロックを必要とするため、NFA 回路において状態遷移のタイミングをシフトレジスタで制御する．

パターン依存ハードウェアマッチングエンジンにおいては，[7] や [8] で同様なアイディアが採用されており，効果を上げている．我々の採用する NFA 回路はパターン非依存であり，状態数削減による効果がさらに大きく，より効果的な拡張と言える．

4.1 多文字遷移 NFA 回路によるセル回路拡張

より効率的に KK-項を含む正規表現のマッチングを行うため，セル回路にパターン非依存多文字遷移 NFA 回路を導入したハードウェア構成を提案する（図 12）．提案ハードウェアエンジンは CC を拡張した ECC1 と ECC2 からなる．本章での提案回路では，各 ECC1 はそれぞれ一つの NFA 回路と接続されている．KK-項の文字および文字列の比較は ECC1 および ECC2 で行われ，比較結果が ECC1 に接続された NFA 回路に送られる．NFA 回路では比較結果を元に状態遷移を行い，KK-項全体でのマッチングが行われる．ECC1 と ECC2 については次の節で詳細を説明する．

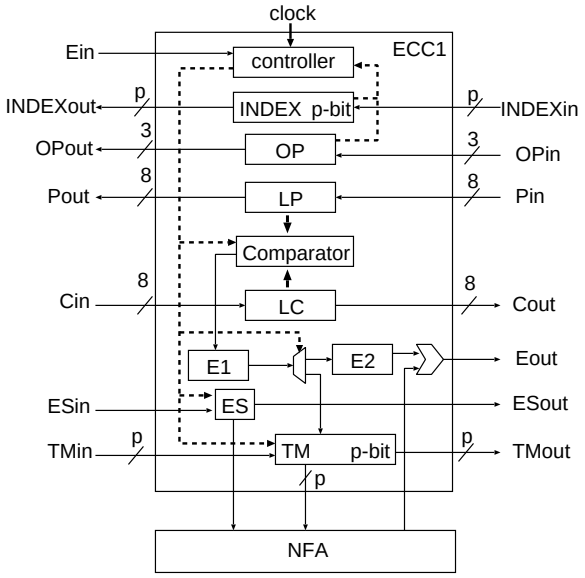

図 13: ECC1 の構造

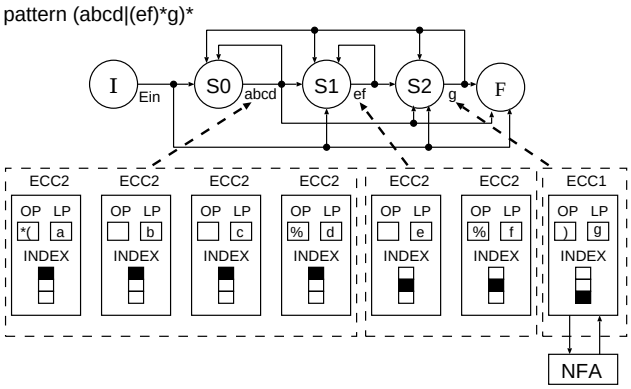

図 14: 拡張 CC へのパターン設定

4.2 拡張比較セル（CC），ECC1, ECC2

ここではまず拡張セル（ECC1 および ECC2）の構造について説明し，次に拡張セルの動作について説明する．

ECC1 の構造を図 13 に示す．ECC1 と ECC2 の構造や動作はほぼ同じであるが，ECC1 は多文字遷移 NFA 回路と接続されているが，ECC2 は NFA 回路と接続されていないという点が異なる．多文字遷移を実現するため，拡張 CC（ECC1 と ECC2）は状態遷移に必要な文字列や文字のマッチングを行う．そのため，KK-項内の各パターン文字は拡張 CC の LP に設定される．また各 CC にはパターン文字の他に 2 つの情報が設定される．各 CC の OP レジスタには，必要に応じて，KK-項の先頭文字と末尾文字，各状態に対応する文字列の末尾文字を表す値がセットされる．本稿では，それぞれの値を “*(”，“)”，”%” としている．各 CC の p ビット幅の INDEX レジスタには，設定されている文字がどの状態の遷移文字列に対応しているかを示すため，状態の番号がセットされる．p は多文字遷移 NFA 回路

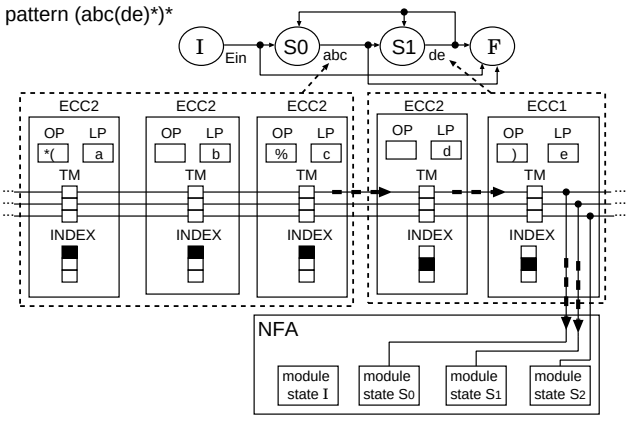

図 15: 遷移文字列のマッチング結果の伝達

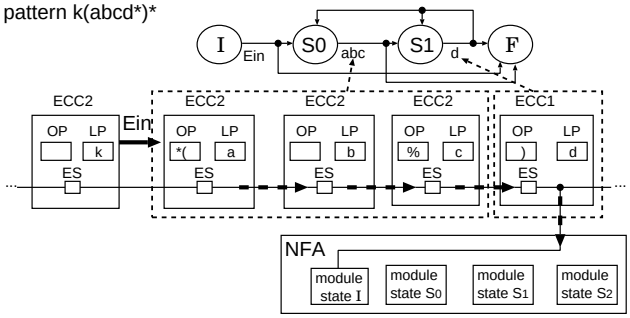

図 16: KK-項のマッチング開始の伝達

内の状態モジュール数である．LP，OP，INDEX の値は外部から入力される．パターン “$(abcd|(ef)^*g)^*$” の場合，図 14 のようになる．

次に拡張 CC の動作を説明する．拡張 CC は 2 つの動作を行う．

1 つ目は，KK-項のマッチング開始を NFA 回路に伝える動作である．KK-項のマッチングが開始されるのは KK-項の先頭文字がセットされた拡張 CC にマッチング開始信号が入力された（Ein=true）場合である．KK-項の先頭文字がセットされた拡張 CC で Ein=true となると，ES レジスタを使い，クロックサイクル毎に各拡張 CC の ES レジスタを通り，NFA 回路に伝えられる（図 16）．

2 つ目は，各状態の遷移文字や文字列のマッチングを行い，NFA 回路内の対応する状態モジュールにマッチング結果を伝達する動作である．まず各状態の遷移文字列のマッチングを行うため，遷移文字列の先頭文字がセットされている拡張 CC は，マッチング開始信号が入力されてない（Ein=false）場合も，常に文字のマッチングを行う．遷移文字列内の各文字の比較結果は E1 と E2 を用いて伝えられる．遷移文字列の末尾文字がセットされている拡張 CC で比較に成功すると，p ビットの TM レジスタを用いて，NFA 回路内の対応する状態モジュールに遷移文字列のマッチング成功を伝える．対応する状態モジュールにマッチング成功を伝えるため，INDEX レジスタが示す TM レジスタの 1 ビットを使い，クロックサイクルごとに各拡張 CC の TM レジスタを通り，対応する状態モジュールにマッチング結果を伝達する（図 15）．

```
begin
  MOVE_INPUT_STRING;
  E1:= if (LP==LC) then begin
    if(Ein) true; else if (OP==“∗(”) true; else false end;
  else false;
  E2:= if(OP=“%”) true; else if(OP=“)”) false; else E1;
  Eout:= if(OP==“)”) NFA(ES,TM); else E2;
  ES:= if(OP=“∗(”) Ein; else ESin;
  ES_out:= if (OP=“)”) false; else ES;
  TM:= if (E1) then begin
    if(OP==“%”) TMin | INDEX;
    else if (OP==“)”) TMin | INDEX; else TMin end;
  else TMin;
  TM_out:= if (OP=“)”) false; else TM;
end;
```

図 17: 拡張 CC の模擬コード

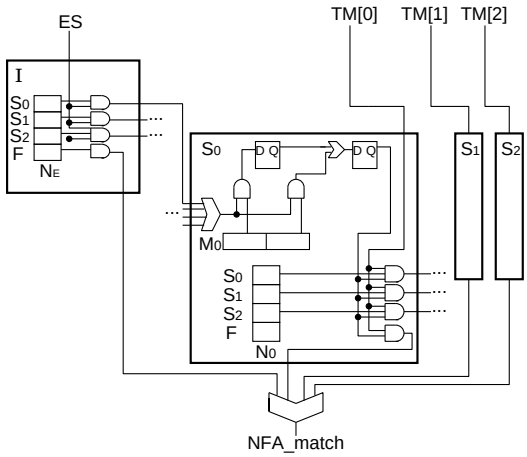

図 18: 多文字遷移 NFA 回路

拡張 CC の模擬コードを図 17 に示す．図 17 において，“A|B” は A と B の間のビットワイズ OR の結果を意味し，“NFA(ES,TM)” は ES と TM の値に対する多文字遷移 NFA 回路の出力を意味する．

4.3 多文字遷移 NFA 回路

次に多文字遷移 NFA 回路について説明する．多文字遷移 NFA 回路の構成を図 18 に示す．この図は状態間の配線を省略している．この NFA 回路の各状態 S_i は次状態遷移レジスタ N_i の他に，h ビットのシフトレジスタ，遷移文字列の比較にかかるクロック数を記憶する h ビットのレジスタ M_i からなる．ここで h は多文字遷移 NFA 回路において，許容する最大遷移文字数（遷移文字列長）である．図 18 では h=2 としている．パターン設定時に N_i と M_i がセットされる．ECC1 とつながった NFA 回路は，マッチング開始信号 ES と状態 S_i の遷移文字列のマッチング成功信号 TM[i] の値をもとに状態遷移を行い，受理状態に到達したとき，KK-項のマッチング成功として ECC1 への出力信号 NFA_match を true とする．NFA 回路の処理の流れは以下である．

動作 1. [**初期状態からの遷移**] ES が入力されると（ES=true），NFA 回路は次状態レジスタ N_E によって指定された状態モジュールへの信号（状態遷移信号）を true とする．

動作 2. [**受理状態を除く状態への遷移**] 状態モジュール S_i に入力される状態遷移信号が true の場合，M_i レ

ジスタが示すシフトレジスタのビットの値を true にする．シフトレジスタの値はクロックごとに右シフトされ，状態遷移信号を受け取った時点から n_i クロック後にシフトレジスタの最右 bit へ到達する．ここで n_i は状態 S_i に対応する文字や文字列の比較にかかるクロック数で，文字列の長さと等しい．

動作 3. [**初期状態を除く状態からの遷移**] S_i 内のシフトレジスタの最右 bit が true かつ TM の i ビット目が true のとき，次状態遷移レジスタ N_i に示された状態モジュール S_j への状態遷移信号を true とする．

動作 4. [**受理状態への遷移**] 動作 3. において次状態レジスタ N_i が受理状態を示している場合，ECC1 への出力信号 NFA_match を true とする．

4.4 提案ハードウェアエンジンへのパターン設定

提案ハードウェアエンジンにおいて，拡張 CC にパターンをセットする際，KK-項の末尾文字は ECC1（NFA 回路と接続した CC）にセットする必要がある．そのためパターン設定時に空語を適宜挿入して，KK-項の末尾文字が ECC1 に来るよう調整する（図 19）．

4.5 NIDS システムへの提案ハードウェアエンジンの適用

2.4 節で説明した NIDS のシステム構成において，従来のパターン非依存ハードウェアエンジンを用いた場合，クリーネ演算のネストを用いたウィルスパターンに対しては更新している間は対応できず，セキュリティホールが存在した．本稿で提案したパターン非依存エンジンを用いることで，そのセキュリティホールを埋めることができ，従来よりもネットワークのセキュリティが向上すると考えられる．

4.6 動作例

図 20 にパターン “$(ab(cd)^*e)^*$” と入力系列 “abeabcdcdefg” に対する提案ハードウェアエンジンの動作例を示す．紙面の都合上，(a) にはセル回路，(b) には NFA 回路の動作を別々に示している．(a) は各クロックサイクルにおける各 CC の出力信号と保持している入力系列内の 1 文字を示している．CC1 と CC2 は状態 S_0 の遷移文字が設定されているため INDEX の 0 ビットが true に設定される．CC3 と CC4 は 1 ビット目が，CC5 は 2 ビット目が true に設定される．T1 クロックでは，入力系列の 1 番目の文字 ‘a’ とマッチング開始信号 Ein が CC1 に入力される．CC1 は Ein 信号が入力されたため，KK-項のマッチング開始を ES 信号として T2 クロックで右隣の CC に伝える．この ES 信号は毎クロックサイクル，各 CC の ES レジスタを通り，T6 クロックで NFA 回路に伝えられる．CC1 は遷移文字列の先頭文字を記憶しているため，T1 クロックで入力系列の 1 番目の文字 ‘a’ の比較を行う．CC1 でのマッチング成功は従来 CC 同様 2 クロック後に CC2 に伝えられる．T3 クロックで CC2 に 2 文字目の ‘b’ と Ein が入力され，‘b’ のマッチングが行われる．‘b’ のマッチング成功は，遷移文字列 “ab” のマッチングが成功したことを意味する．そのため，T5 クロックで TM レジスタの 0 ビット目が true となり，各 CC を通り，T8 クロックで “ab” のマッチング成功が NFA 回路に伝わる．他の遷移文字列に対するマッチング成功の伝達については

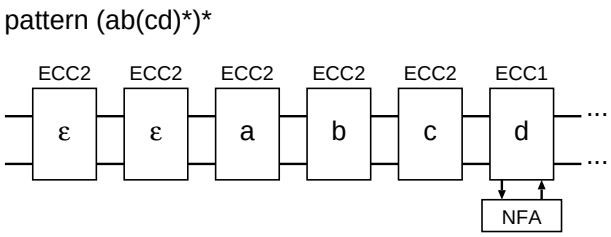

図 19: 提案ハードウェアエンジンへのパターン設定

の説明は省略する．

(b) は CC5 につながった NFA 回路の動作を示している．各クロックサイクルにおける各状態モジュール内のシフトレジスタの値をタプルで表現している．T1 から T5 クロックにおいて NFA 回路に CC5 からの入力がないため，状態遷移は起こらず，シフトレジスタの値は変わらない．T6 クロックで ES 信号が CC5 から伝わる．このとき，N_E が受理状態 F を示しているため，CC5 への出力信号 NFA_match が true となる．これは空語に対するマッチング成功である．また N_E は状態 S_0 も示しているため，T7 クロックで状態モジュール S_0 内のシフトレジスタの最下位ビットから 2 ビット目が true となる．T8 クロックで S_0 内のシフトレジスタの最下位ビットが true となり，かつ TM[0]=true となるため状態 S_1 と S_2 に遷移する．T9 クロックで TM[2]=true となり，状態モジュール S_2 のシフトレジスタの最下位ビットの値が true であるため，受理状態と状態 S_0 に遷移する．受理状態に遷移するため，T9 クロックで NFA_match が true となる．これは，入力系列内の部分文字列 “abe” に対するマッチング成功を示す．以降同様に遷移し，T16 クロックで NFA_match が true となる．これは，部分文字列 “abeabcdcde” に対するマッチング成功を示す．

5 実験的評価

本稿で提案した多文字遷移 NFA 回路と拡張 CC を組み合わせた構成（図 12）と従来ハードウェアエンジン（図 3）に対して面積と速度の評価を行った．統合開発環境は Xilinx 社の ISE11.1，FPGA は Vertex-4（XC4VLX100-11F1513）を用いた．扱うパターンは Snort ルール v2.7 とした．ルールを解析した結果，提案ハードウェアエンジンの各パラメータを以下のように設定した．

多文字遷移 NFA 回路の状態モジュール数：5
各状態の最大遷移文字数：3
ECC1 と ECC2 の割合: 1：4

この条件下で面積と速度の評価を行った．面積は実験で用いた FPGA に実装できるセル数で評価した．従来エンジンの最高動作周波数は 301.8MHz で 1250 セル実装可能である．提案エンジンは 271.7MHz で 1000 セル実装可能であることが分かった．

これらの結果から，提案エンジンは，正規表現クラス拡張前の従来エンジンと比較して，処理速度と面積効率をそれほど落とさずに KK-項を扱えることが確認できた．また提案エンジンは 1 クロックサイクルで 1 文字（8 ビット）の処理が可能なため，スループットは 2.17Gbps（8×271.7Mbps）であり，ギガビットイーサネットにも対応可能な性能であることが確認できた．

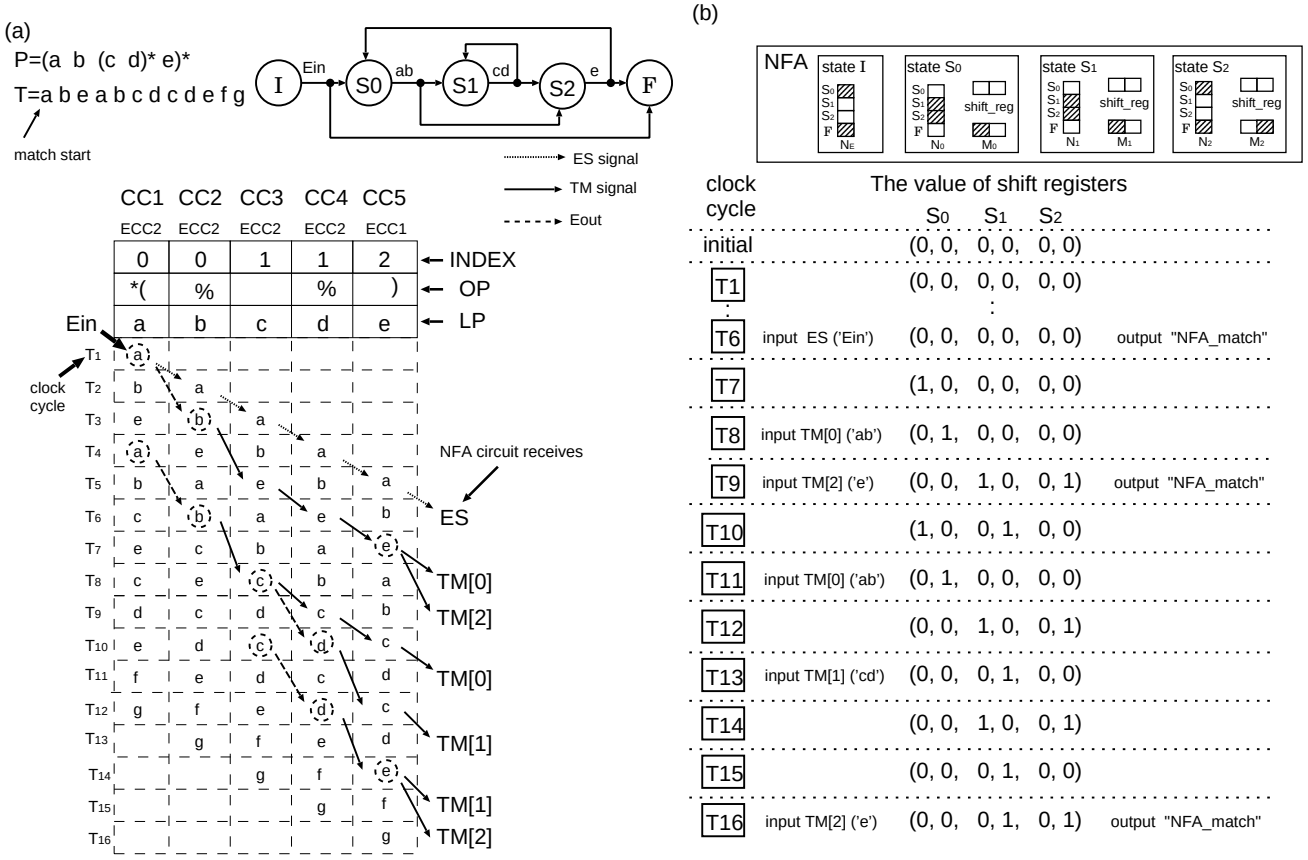

図 20: 動作例

6 おわりに

本稿では，クリーネ演算のネストを扱うことができるシストリックアルゴリズムに基づく正規表現マッチングハードウェアエンジンを提案した．提案エンジンでは，クリーネ演算のネストを扱うため，小型のパターン非依存 NFA 回路を導入した．実験により，提案エンジンは効率的にクリーネ演算のネストを扱える事が確認でき，またギガビットイーサネット上でもウィルスを検知できることが確認できた．

今後の課題としては，後方参照への対応などがあげられる．

参考文献

[1] Sourcefire, Inc., “SNORT Network Intrusion Detection System,” http://www.snort.org/.

[2] Jon Allen (JJ),“perldoc.perl.org - Official documentation for the Perl programming language,” http://perldoc.perl.org/perlre.html.

[3] J. Bispo, I. Sourdis, J. M. P. Cardoso, S. Vassiliadis, “Regular expression matching for reconfigurable packet inspection,” Proc. 2006 IEEE ICFPT, pp.119-126, 2006.

[4] R. Sidhu, V. K. Prasanna, “Fast Regular Expression Matching Using FPGAs,” Proc. 2001 IEEE FCCM, pp.227-238, 2001.

[5] C. R. Clark, D. E. Schimmel, “Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns,” Proc. 2003 IEEE ICFPL, pp.956-959, 2003.

[6] Y. K. Chang, C. R. Chang, C. C. Su, “The Cost Effective Pre-processing Based NFA Pattern Matching Architecture for NIDS,” Proc. 2010 IEEE ICAINA, pp.385-391, 2010.

[7] N. Yamagaki, R. Sidhu, S. Kamiya, “High-Speed Regular Expression Matching Engine Using Multi-Character NFA,” Proc. 2008 IEEE ICFPL, pp.131-136, 2008.

[8] H. Nakahara, T. Sasao, M. Matuura, “A Regular Expression Matching Circuit Based on a Modular Non-Deterministic Finite Automaton with Multi-Character Transition,” Proc. 2010 SASIMI, pp.359-364, 2010.

[9] Y. Kawanaka, S. Wakabayashi, S. Nagayama, “A Systolic Regular Expression Pattern Matching Engine and its Application to Network Intrusion Detection,” Proc. 2008 IEEE ICFPT, pp.297-300, 2008.

[10] J. Divyasree, H. Rajashekar, Kuruvilla Varghese, “Dynamically reconfigurable regular expression matching architecture,” Proc. 2008 ASAP, pp.120-125, 2008.

[11] J. E. Hopcroft, R. Motwani, J. D. Ullman, “Introduction to Automata Theory, Languages, and Computation (2nd Edition),” Pearson Education, 2000.

[12] Y. Kawanaka, S. Wakabayashi, S. Nagayama, “A Fast Regular Expression Matching Engine for an FPGA-based Network Intrusion Detection System,” Proc. 2009 IEEE SASIMI, pp.88-93, 2009.