(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5265378号
(P5265378)
(45)発行日　平成25年8月14日(2013.8.14)　(24)登録日　平成25年5月10日(2013.5.10)

(51)Int.Cl.　F I
G06F 17/30 (2006.01)　G06F 17/30 415
G06F 17/30 170A

請求項の数 38　(全 32 頁)

(21)出願番号	特願2008-543412 (P2008-543412)
(86)(22)出願日	平成18年11月29日 (2006.11.29)
(65)公表番号	特表2009-517782 (P2009-517782A)
(43)公表日	平成21年4月30日 (2009.4.30)
(86)国際出願番号	PCT/US2006/045653
(87)国際公開番号	WO2007/064685
(87)国際公開日	平成19年6月7日 (2007.6.7)
審査請求日	平成21年10月23日 (2009.10.23)
(31)優先権主張番号	11/293,619
(32)優先日	平成17年12月2日 (2005.12.2)
(33)優先権主張国	米国(US)

(73)特許権者　508137947
エクセジー・インコーポレイテッド
アメリカ合衆国、ミズーリ・６３１２７、セント・ルイス、サウス・ガイヤー・ロード・３６６８、スイート・３００
(74)代理人　100103920
弁理士　大崎　勝真
(74)代理人　100114188
弁理士　小野　誠
(74)代理人　100140523
弁理士　渡邊　千尋
(74)代理人　100119253
弁理士　金山　賢教
(74)代理人　100124855
弁理士　坪倉　道明

最終頁に続く

(54)【発明の名称】高性能正規表現パターンマッチングのための方法および装置

(57)【特許請求の範囲】
【請求項１】
決定性有限オートマトン（ＤＦＡ）により入力文字列をパターンにマッチさせるための装置であって、ＤＦＡが、現在状態を含む複数の状態を備え、入力文字列が、複数の入力シンボルを含み、装置が、
ＤＦＡを実施するパターンマッチング回路を備え、パターンマッチング回路が、ｍ個の入力シンボルからなるグループとして入力文字列の入力シンボルを受け取り、連続して処理するように構成され、ここで、ｍは１以上の整数であり、パターンマッチング回路が、（１）複数の保存された遷移を含む遷移テーブルメモリであって、各保存された遷移が、少なくとも１つの入力シンボルに対応するデータによってインデックス付けされ、各遷移が、次状態識別子を含む、遷移テーブルメモリと、（２）受け取られた入力シンボルグループの少なくとも１つの入力シンボルに対応するデータによってインデックス付けされる各遷移を、現在状態を考慮することなく、遷移テーブルメモリから取り出すように構成される遷移取り出しロジックと、（３）各取り出された遷移を受け取り、取り出された遷移のどれが現在状態に対応するかを決定し、決定された遷移の次状態識別子に基づいてＤＦＡについての次状態を決定し、入力文字列とパターンの間にマッチが存在するかどうかを決定するように構成される状態選択ロジックと、を備える、装置。
【請求項２】
入力文字列の各入力シンボルが、アルファベットの要素である複数の入力シンボルの１つであり、パターンマッチング回路が、各受け取られた入力シンボルグループを同値類識

別子（ＥＣＩ）に変換するように構成されるアルファベット符号化ロジックをさらに備え、遷移テーブル内で各遷移がそれによってインデックス付けされる入力シンボルグループに対応するデータが、その入力シンボルグループに対するＥＣＩに対応するデータを含む、請求項１に記載の装置。

【請求項３】

パターンマッチング回路と通信する正規表現コンパイラであって、パターンマッチング回路によって実施されるＤＦＡを決定するためにユーザから正規表現を受け取り、処理し、処理された正規表現に基づいて遷移テーブルにデータ投入し、処理された正規表現に基づいてＥＣＩを定義するように構成される正規表現コンパイラをさらに備え、正規表現は入力文字列がそれに対してマッチさせられるパターンを識別する、請求項２に記載の装置。

【請求項４】

遷移テーブルが、メモリ内で実施され、メモリが、メモリアドレスによってインデックス付けされる複数のワードを含み、各ワードが、ランレングス符号化された複数の遷移を含む、請求項３に記載の装置。

【請求項５】

パターンマッチング回路が、インダイレクションテーブルと、インダイレクションテーブルロジックとをさらに備え、インダイレクションテーブルが、複数のエントリを含み、各エントリが、ＥＣＩによってインデックス付けされ、各エントリが、そのＥＣＩによってインデックスされる少なくとも１つの遷移を含むメモリ内でワードが保存されるメモリアドレスを指すポインタを含み、インダイレクションテーブルロジックが、受け取られた入力シンボルグループによってインデックス付けされるインダイレクションテーブルからポインタを取り出し、取り出されたポインタを遷移取り出しロジックに提供するように構成され、各遷移がそれによってインデックス付けされるその入力シンボルグループに対するＥＣＩに対応するデータが、取り出されたポインタによって識別されるメモリアドレスを含む、請求項４に記載の装置。

【請求項６】

各インダイレクションテーブルエントリが、遷移インデックスと、遷移カウントとをさらに含み、遷移インデックスが、入力シンボルグループに対するＥＣＩによってインデックス付けされる第１の遷移が、そのエントリのポインタによって識別されるワード内のどこに配置されているかを識別し、遷移カウントが、いくつの遷移が、入力シンボルグループに対するＥＣＩによってインデックス付けされるかを識別し、インダイレクションテーブルロジックが、受け取られた入力シンボルグループによってインデックス付けされるインダイレクションテーブルエントリから、遷移インデックスおよび遷移カウントを取り出し、取り出された遷移インデックスおよび遷移カウントを、遷移取り出しロジックに提供するように構成される、請求項５に記載の装置。

【請求項７】

遷移取り出しロジックが、取り出されたポインタおよび取り出された遷移カウントによって識別される各ワードを遷移テーブルから取り出すように構成され、状態選択ロジックが、遷移に対応する各取り出されたワードの、入力シンボルグループに対するＥＣＩによってインデックス付けされないどの部分もマスクアウトするようにさらに構成される、請求項６に記載の装置。

【請求項８】

各インダイレクションテーブルエントリが、先頭遷移インデックスと、末尾遷移インデックスと、ワードカウントとをさらに含み、先頭遷移インデックスが、入力シンボルグループに対するＥＣＩによってインデックス付けされる第１の遷移が、そのエントリのポインタによって識別されるワード内のどこに配置されているかを識別し、ワードカウントが、受け取られた入力シンボルグループに対するＥＣＩによってインデックス付けされるすべての遷移がその中に連続的に保存されるワードの数を識別し、末尾遷移インデックスが、入力シンボルグループに対するＥＣＩによってインデックス付けされる最後の遷移が、

そのエントリのワードカウントによって識別される最後のワード内のどこに配置されているかを識別し、インダイレクションテーブルロジックが、受け取られた入力シンボルグループによってインデックス付けされるインダイレクションテーブルエントリから、先頭遷移インデックス、末尾遷移インデックス、およびワードカウントを取り出し、取り出された先頭遷移インデックス、末尾遷移インデックス、およびワードカウントを、遷移取り出しロジックに提供するように構成される、請求項５に記載の装置。

【請求項９】

遷移取り出しロジックが、取り出されたポインタおよび取り出されたワードカウントによって識別される各ワードを、遷移テーブルから取り出すように構成され、状態選択ロジックが、遷移に対応する各取り出されたワードの、入力シンボルグループに対するＥＣＩによってインデックス付けされないどの部分もマスクアウトするようにさらに構成される、請求項８に記載の装置。

【請求項１０】

各遷移が、事前計算ランレングスプレフィックス合計をさらに含む、請求項５に記載の装置。

【請求項１１】

ｍが、１よりも大きい、請求項５に記載の装置。

【請求項１２】

正規表現コンパイラが更に、処理された正規表現に基づいてインダイレクションテーブルにデータ投入するように構成される、請求項５に記載の装置。

【請求項１３】

パターンマッチング回路が、ＦＰＧＡ上で実施される、請求項５に記載の装置。

【請求項１４】

パターンマッチング回路が、ＡＳＩＣ上で実施される、請求項５に記載の装置。

【請求項１５】

各遷移が、マッチフラグをさらに含み、状態選択ロジックが、決定された遷移のマッチフラグに基づいて、入力文字列とパターンの間にマッチが存在するかどうかを決定するようにさらに構成される、請求項３に記載の装置。

【請求項１６】

ＤＦＡが、位置独立のＤＦＡを含む、請求項１５に記載の装置。

【請求項１７】

各遷移が、マッチリスタートフラグをさらに含み、状態選択ロジックが、決定された遷移のマッチリスタートフラグに基づいて、パターンマッチングプロセスがＤＦＡにおいてリスタートされたかどうかを決定するようにさらに構成される、請求項１６に記載の装置。

【請求項１８】

ｍが、１よりも大きい、請求項１７に記載の装置。

【請求項１９】

パターンマッチング回路と通信する正規表現コンパイラであって、パターンマッチング回路によって実施されるＤＦＡを決定するためにユーザから正規表現を受け取り、処理し、処理された正規表現に基づいて遷移テーブルにデータ投入するように構成される正規表現コンパイラをさらに備え、正規表現は入力文字列がそれに対してマッチさせられるパターンを識別する、請求項１に記載の装置。

【請求項２０】

パターンマッチング回路が、ＦＰＧＡ上で実施される、請求項１９に記載の装置。

【請求項２１】

パターンマッチング回路が、ＡＳＩＣ上で実施される、請求項１９に記載の装置。

【請求項２２】

パターンマッチング回路が、ＦＰＧＡ上で実施される、請求項１に記載の装置。

【請求項２３】

パターンマッチング回路が、ＡＳＩＣ上で実施される、請求項１に記載の装置。
【請求項２４】
パターンマッチング回路が、マルチプロセッサシステム上で実施される、請求項１に記載の装置。
【請求項２５】
遷移取り出しロジックと状態選択ロジックが、並列に動作するパイプラインステージとして構成される、請求項１から２４のいずれか一項に記載の装置。
【請求項２６】
状態機械を使用して入力文字列をパターンにマッチさせる方法であって、入力文字列が、複数の入力シンボルを含み、各入力シンボルが、複数のビットを含み、方法が、
状態機械についての複数の状態を定義するステップであって、状態機械が、現在状態を有し、入力文字列の入力シンボルが受け取られ、処理されるのにつれて、複数の状態を経ながら進行し、それによって、入力文字列の複数の入力シンボルがパターンとマッチするかどうかを検出するように構成される、ステップと、
状態機械の状態間の複数の遷移を定義するステップであって、各遷移が、状態機械の現在状態に対応するデータと、少なくとも１つの入力シンボルに対応するデータとによってインデックス付けされ、各遷移が、状態機械の次状態についての識別子と、複数の入力シンボルとパターンの間にマッチが存在するかどうかを示すマッチフラグとを含む、ステップと、
入力文字列の少なくとも１つの入力シンボルを受け取るステップと、
少なくとも１つの受け取られた入力シンボルに基づいて及び現在状態を考慮することなく、少なくとも１つの定義された遷移を取り出すステップと、
少なくとも１つの取り出された遷移のどれが現在状態によってインデックス付けされているかを決定するステップと、
状態機械の現在状態を、決定された取り出された遷移の次状態識別子によって識別される状態として更新するステップと、
取り出された遷移のマッチフラグに基づいて、複数の入力シンボルとパターンの間にマッチが存在するかどうかを決定するステップと、
入力文字列の追加の入力シンボルが受け取られた場合に、取り出し、遷移決定、更新、およびマッチ決定のステップを繰り返すステップと、を備える、方法。
【請求項２７】
各遷移が、受け取られた少なくとも１つの入力シンボルがパターンに対するマッチングプロセスのリスタートをもたらすかどうかを示すマッチリスタートフラグをさらに含み、方法が、
取り出された遷移のマッチリスタートフラグに基づいて、マッチリスタートが存在するかどうかを決定するステップをさらに備え、
繰り返すステップが、入力文字列の追加の入力シンボルが受け取られた場合に、取り出し、遷移決定、更新、マッチ決定、および部分的マッチ開始決定のステップを繰り返すステップを備える、請求項２６に記載の方法。
【請求項２８】
受け取るステップが、各クロックサイクルについて複数のｍ個の入力シンボルを受け取るステップを備え、取り出すステップが、ｍ個の受け取られた入力シンボルに基づいて、少なくとも１つの定義された遷移を取り出すステップを備える、請求項２７に記載の方法。
【請求項２９】
状態機械によって処理され得る入力シンボルのアルファベットに対して、各入力シンボルが同値類識別子にマッピングされるように、複数の同値類識別子を定義するステップであって、各遷移がそれによってインデックス付けされる少なくとも１つの入力シンボルに対応するデータが、その少なくとも１つの入力シンボルに対する同値類識別子を含む、ステップと、

受け取られた少なくとも１つの入力シンボルをその同値類識別子にマッピングするステップと、をさらに備え、
取り出すステップが、少なくとも１つの受け取られた入力シンボルに対する同値類識別子に基づいて、少なくとも１つの定義された遷移を取り出すステップを備え、
繰り返すステップが、入力文字列の追加の入力シンボルが受け取られた場合に、マッピング、取り出し、遷移決定、更新、マッチ決定、および部分的マッチ開始決定のステップを繰り返すステップを備える、請求項２８に記載の方法。
【請求項３０】
状態機械によって処理され得る入力シンボルのアルファベットに対して、各入力シンボルが同値類識別子にマッピングされるように、複数の同値類識別子を定義するステップであって、各遷移がそれによってインデックス付けされる少なくとも１つの入力シンボルに対応するデータが、その少なくとも１つの入力シンボルに対する同値類識別子に対応するデータを含む、ステップと、
受け取られた少なくとも１つの入力シンボルをその同値類識別子にマッピングするステップと、をさらに備え、
取り出すステップが、少なくとも１つの受け取られた入力シンボルに対する同値類識別子に基づいて、少なくとも１つの定義された遷移を取り出すステップを備え、
繰り返すステップが、入力文字列の追加の入力シンボルが受け取られた場合に、マッピング、取り出し、遷移決定、更新、マッチ決定、および部分的マッチ開始決定のステップを繰り返すステップを備える、請求項２６に記載の方法。
【請求項３１】
ランレングス符号化によってメモリ内で遷移を圧縮するステップをさらに備える、請求項３０に記載の方法。
【請求項３２】
圧縮するステップが、事前計算プレフィックス合計を有するランレングス符号化によってメモリ内で遷移を圧縮するステップをさらに備える、請求項３１に記載の方法。
【請求項３３】
メモリが、複数のワードを含み、各ワードが、メモリアドレスによってインデックス付けされ、各ワードが、複数のｘ個の圧縮された遷移を含み、方法が、
少なくとも１つの受け取られた入力シンボルに対する同値類識別子に基づいて、圧縮された遷移にアクセスするためのインダイレクションテーブルを定義するステップであって、インダイレクションテーブルが、複数のエントリを含み、各エントリが、同値類識別子によってインデックス付けされ、各エントリが、少なくとも１つの受け取られた入力シンボルに対する同値類識別子によってインデックス付けされる少なくとも１つの圧縮された遷移を含むワードが保存されるメモリアドレスを指すポインタを含む、ステップをさらに備え、
各遷移がそれによってインデックス付けされる同値類識別子に対応するデータが、その同値類識別子に対するインダイレクションテーブルポインタを含み、方法が、
同値類識別子によってインデックス付けされるポインタを決定するために、少なくとも１つの受け取られた入力シンボルに対する同値類識別子に基づいてインダイレクションテーブルにアクセスするステップをさらに備え
取り出すステップが、少なくとも１つの受け取られた入力シンボルの同値類識別子に対する決定されたポインタ基づいて、少なくとも１つの定義された遷移を取り出すステップを備え、
繰り返すステップが、入力文字列の追加の入力シンボルが受け取られた場合に、マッピング、アクセス、取り出し、遷移決定、更新、マッチ決定、および部分的マッチ開始決定のステップを繰り返すステップを備える、請求項３０に記載の方法。
【請求項３４】
各インダイレクションテーブルエントリが、遷移インデックスと、遷移カウントとをさらに含み、遷移インデックスが、少なくとも１つの受け取られたシンボルの同値類識別子

に対する第１の圧縮された遷移が、ポインタによって識別されるワード内のどこに配置されているかを識別し、遷移カウントが、少なくとも１つの受け取られたシンボルの同値類識別子に対していくつの圧縮された遷移が、メモリ内に連続的に保存されているかを識別し、アクセスするステップが、同値類識別子によってインデックス付けされるポインタ、遷移インデックス、および遷移カウントを決定するために、少なくとも１つの受け取られた入力シンボルに対する同値類識別子に基づいて、インダイレクションテーブルにアクセスするステップを備え、取り出すステップが、少なくとも１つの受け取られた入力シンボルの同値類識別子に対する決定されたポインタ、遷移インデックス、および遷移カウントに基づいて、各定義された遷移を取り出すステップをさらに備える、請求項３３に記載の方法。

【請求項３５】

各インダイレクションテーブルエントリが、先頭遷移インデックスと、末尾遷移インデックスと、ワードカウントとをさらに含み、先頭遷移インデックスが、少なくとも１つの受け取られたシンボルの同値類識別子に対する第１の圧縮された遷移が、ポインタによって識別されるワード内のどこに配置されているかを識別し、ワードカウントが、少なくとも１つの受け取られたシンボルの同値類識別子に対する少なくとも１つの圧縮された遷移がメモリ内で連続的に保存されるワードの数を示し、末尾遷移インデックスが、最後の圧縮された遷移が、ワードカウントによって識別される最後のワード内のどこに配置されているかを識別し、アクセスするステップが、同値類識別子によってインデックス付けされるポインタ、先頭遷移インデックス、末尾遷移インデックス、およびワードカウントを決定するために、少なくとも１つの受け取られた入力シンボルに対する同値類識別子に基づいて、インダイレクションテーブルにアクセスするステップを備える、請求項３３に記載の方法。

【請求項３６】

取り出すステップが、決定されたポインタおよび決定されたワードカウントによって識別される各ワードを、メモリから取り出すステップをさらに備え、遷移決定ステップが、決定された先頭遷移インデックス、決定された末尾遷移インデックス、および決定されたワードカウントによって識別される範囲内にない各取り出されたワード内に配置されるどの圧縮された遷移もマスクアウトするステップを備える、請求項３５に記載の方法。

【請求項３７】

状態定義ステップが、正規表現の集合から状態機械についての複数の状態を定義するステップを備える、請求項３６に記載の方法。

【請求項３８】

取り出すステップと少なくとも１つの取り出された遷移のどれが現在状態によってインデックス付けされているかを決定するステップが、パイプライン化された方法で並列に実行される、請求項２６から３７のいずれか一項に記載の方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、一般に、データシンボルストリームに属するいずれかの文字列がパターンとマッチするかどうかを決定するための、データシンボルからなるストリームの処理の分野に関する。

【背景技術】

【０００２】

ネットワークおよびストレージサブシステム設計における進歩が、コンピュータシステム間およびコンピュータシステム内でデータストリームが処理されなければならない速度を押し上げ続けている。一方で、すべてのレベルのコンポーネントが、時間的制約のある動作をトリガし得るパターンに関してストリームを調べるので、そのようなデータストリームの内容は、ますます増加する精査に委ねられる。パターンは、文字列定数（例えば「ｄｏｇ」および「ｃａｔ」）ばかりでなく、いくつかを挙げれば、クレジットカード番号

、通貨価値、または電話番号などを表す、明細事項も含むことができる。広く使用されているパターン指定言語は、正規表現言語である。正規表現と、決定性有限オートマトン（ＤＦＡ）によるそれらの実施は、よく発達した分野である。ＨｏｐｃｒｏｆｔおよびＵｌｌｍａｎ、「Ｉｎｔｒｏｄｕｃｔｉｏｎ ｔｏ Ａｕｔｏｍａｔａ Ｔｈｅｏｒｙ，Ｌａｎｇｕａｇｅｓ，ａｎｄ Ｃｏｍｐｕｔａｔｉｏｎ」、Ａｄｄｉｓｏｎ Ｗｅｓｌｅｙ、１９７９年を参照されたく、その開示の全体は、参照により本明細書に組み込まれる。以下で説明されるように、ＤＦＡは、状態機械（ｓｔａｔｅ ｍａｃｈｉｎｅ）の動作を定義する論理表現である。しかし、高性能パターンマッチングとの関連において、正規表現の使用を改善する必要性が当技術分野に存在すると、発明者らは本明細書において確信する。

【０００３】

パケットヘッダフィルタリングなど、いくつかのアプリケーションでは、与えられたパターンの位置は、固定されることができ、位置の固定は、パターンがデータストリーム内の１組の定められた位置で開始または終了する場合にのみ、マッチが発生する状況を示す。より一般的に、多くのアプリケーションでは、パターンは、データストリーム（例えば、非構造化データストリーム、パケットペイロードなど）内の任意の場所で開始または終了することができる。いくつかのアプリケーションは、データストリームのあらゆるバイトにおいて、数千のパターンの同時組み付けを必要とする。そのようなアプリケーションの例は、

（約１００００個のパターンからなるルールベースを使用して一般に動作する）ネットワーク侵入検出／防止システム（Ｒｏｅｓｃｈ，Ｍ．、「Ｓｎｏｒｔ－ｌｉｇｈｔｗｅｉｇｈｔ ｉｎｔｒｕｓｉｏｎ ｄｅｔｅｃｔｉｏｎ ｆｏｒ ｎｅｔｗｏｒｋｓ」、ＬＩＳＡ ’９９：１３ｔｈ Ｓｙｓｔｅｍｓ Ａｄｍｉｎｉｓｔｒａｔｉｏｎ Ｃｏｎｆｅｒｅｎｃｅ、２２９－２３８ページ、１９９９年を参照されたく、その開示の全体は、参照により本明細書に組み込まれる）、

不適切または不法な内容を探し出すために発信電子メールをスキャンする電子メール監視システム、

ユーザ指定パターンを課して着信電子メールをフィルタリングするスパムフィルタ、

有害であることが知られているプログラムのシグニチャについてフィルタリングを行うウイルススキャナ、および

海賊版コンテンツを探し出すためにメディアファイルまたはソケットストリームをスキャンする著作権実施プログラム

を含むがこれらに限定されない。上記のようなアプリケーションでは、データストリーム内で探されるパターンの集合は、日々変化し得る。

【０００４】

今日の従来型ハイエンドワークステーションは、高速ネットワークおよびストレージサブシステムから発信されるデータストリームの速度を与えられるパターンマッチングアプリケーションと同じ速度を維持することができない。この性能ギャップに対処するため、発明者らは本明細書において、パイプラインアーキテクチャでのＤＦＡの定式化および実現（例えば、ハードウェアロジック、ネットワーク化プロセッサ、またはその他のパイプライン化処理システム）におけるアーキテクチャ革新に解決策を求める。

【０００５】

正規表現ｒは、正規言語Ｌ（ｒ）を表し、その場合、言語は、（有限）文字列からなる（おそらくは無限）集合である。各文字列は、アルファベットΣから取り出されたシンボルで構成される。正規表現の構文は、以下の基本表現を用いて、帰納的に定義される：

シンボルα∈Σは、｛α｝を表し、

シンボルλは、空（ゼロ幅）文字列を含む単集合を表し、

シンボルφは、空集合を表す。

上記の各々は、正規言語である。より複雑な正規表現は、当技術分野でよく知られているように、合併演算子、連接演算子、およびクリーネ閉包演算子を使用して、構成されるこ

とができる。シンボル範囲指示子および節反復係数が、統語上の便宜のため、一般に提供される。よく知られた正規表現表記および拡張のいずれもが、本発明を実施する際の使用に適しているが、ｐｅｒｌが普及しているため、本明細書の説明および本発明の好ましい実施形態は、正規表現用にｐｅｒｌ表記および拡張を支持する。

【０００６】

上で述べられたように、正規表現は、ファイル検索システムおよびネットワーク侵入検出システムを含むが、これらに限定されない、きわめて多くの検索アプリケーションにおいて、実際的な用途を見出す。ほとんどのテキストエディタおよび検索ユーティリティは、何らかの形式の正規表現構文を使用して、検索ターゲットを指定する。ｐｅｒｌ構文を使用する例示的な一例として、図１に示されるパターンは、米国の通貨価値を表す文字列とマッチするように意図されている。

【０００７】

バックスラッシュ「＼」は、文字通りに解釈されるべき任意の特別なシンボルの前に置かれる、

文字範囲の下限値および上限値は、ダッシュ「－」を使用して指定され、

「＋」符号は、直前の表現が１回以上繰り返され得ることを表し、

中括弧内の単一の数は、直前の表現が指定された回数だけ正確に繰り返され得ることを表し、

中括弧内の対をなす数は、直前の表現についての繰り返しの範囲を表す。

したがって、上記の表現とマッチする文字列は、シンボル「＄」で始まり、それに１０進数字からなる何らかの正の数が続き、その文字列には、任意選択で、１０進小数点「．」と、さらに正確に２つの１０進数字が続く。実際には、そのようなマッチについてのパターンは、マッチが（空白など）何らかの区切り符号によって囲われることも指定することができ、その結果、文字列「＄３４７．１２」は、４つのマッチ（すなわち、「＄３」、「＄３４」、「＄３４７」、「＄３４７．１２」）ではなく、１つのマッチをもたらす。

【０００８】

正規表現を使用して対象とするパターンを指定するアプリケーションは、一般に以下のように動作する。正規表現ｒと、ターゲット文字列ｔ（一般にファイルなど何らかの入力ストリームの内容）を所与として、Ｌ（ｒ）内でｔの部分文字列をすべて見出す。部分文字列は、ｔ内のその位置によって一般に報告される。したがって、別途指摘がない限り、パターンｒがターゲット内のあらゆる位置で適用され、すべてのマッチが報告されることが、一般に意図される。

【０００９】

正規表現を使用して指定されるパターンを認識するための最も簡単で最も実用的なメカニズムは、ＤＦＡであり、ＤＦＡは、形式的に５つ組

（Ｑ，Σ，q_0，δ，Ａ）

として記述され、ここで、

Ｑは、状態の有限集合であり、

Σは、入力シンボルのアルファベットであり、

$q_0 \in Q$は、ＤＦＡの初期状態であり、

δは、遷移関数：

【数１】

$$Q \times \Sigma \mapsto Q$$

であり、

$A \subseteq Q$は、アクセプト状態の集合である。

【００１０】

ＤＦＡは、以下のように動作する。ＤＦＡは、状態q_0で始まる。ＤＦＡが状態ｑにある場合、次の入力シンボルａは、δ（ｑ，ａ）によって決定される遷移を引き起こす。ＤＦＡが状態$q \in A$への遷移をおこす場合、その時点までに処理された文字列はアクセプト

され、ＤＦＡによって認識される言語内にある。例示的な一例として、図１の正規表現は、マッチの開始位置を任意（非位置固定）にするステップを含む一連のよく知られたステップを使用して、図２に示される正準ＤＦＡに翻訳されることができる（上で引用されたＨｏｐｃｒｏｆｔおよびＵｌｌｍａｎの参考文献を参照されたい）。便宜上、図２のＤＦＡは、集合｛０，１，２，３，４，５，６，７，８，９｝を表すために、用語「［０－９］」を使用し、集合｛０，１，２，３，４，５，６，７，８，９，＄，．｝内にないΣのすべてのシンボルを表すために、シンボル「～」を使用する。

【００１１】

ＤＦＡの構成は、非決定性有限オートマトン（ＮＦＡ）が構成される中間ステップを一般に伴う。ＤＦＡが各入力シンボルおよび状態についてたかだか１つの遷移を許容する有限状態機械であるのに対して、ＮＦＡは各入力シンボルおよび状態について２つ以上の遷移を許容する有限状態機械である点で、ＮＦＡはＤＦＡとは異なる。また、あらゆる正規言語は、その言語を認識するのに必要とされる状態の数を最小化することによって取得される、正準（ｃａｎｏｎｉｃａｌ）ＤＦＡを有する。本明細書において別途指摘がない限り、すべてのオートマトンは正準（決定性）形式にあることが仮定されるべきである。

【００１２】

しかし、パターンマッチングの目的にとって、図２に示されるＤＦＡは以下の点で不十分であると、発明者らは本明細書において確信する。

【００１３】

正規表現のアルファベット内にないシンボルは、ＤＦＡがブロックされる原因となる。パターンマッチングの場合、ＤＦＡがマッチの探索を継続し得るように、そのようなシンボルは、ＤＦＡによって無視されるべきである。この不完全性は、以下のようにＤＦＡを完成することによって、克服されることができる。

ターゲット文字列内で発生し得るどのようなシンボルも含むように、アルファベットが広げられる。本明細書では、Σは、２５６個のシンボルを含むＡＳＣＩＩキャラクタセットであることが仮定される。

新しい状態Ｕを有するように、ＤＦＡが拡大される。Ｑ←Ｑ∪｛Ｕ｝。

それまでδが定義されていなかった、すべてのｑ∈Ｑ、ａ∈Σに対して、δ（ｑ，ａ）＝Ｕと定義することによって、遷移関数δが完成される。

【００１４】

マッチは、ターゲット文字列の先頭文字からマッチが始まる場合にのみ見出される。パターンマッチングアプリケーションは、ターゲット内の任意の位置における示されたパターンの発生をすべて見出すことに関心がある。この不完全性は、ＤＦＡがあらゆる位置でリスタートすることを可能にすることによって、克服されることができる。形式的には、あらゆるｑ∈Ｑからｑ$_0$へのλ遷移が挿入される。

【００１５】

上述の拡大の結果は、ＤＦＡを取得するための知られた技法を介して、正準ＤＦＡに変換され得るＮＦＡである。図３は、正準ＤＦＡの例示的な一例を提供している。

【００１６】

ＤＦＡは、その遷移δをテーブルとして実現することによって、一般に解釈的に実施され、各行は、ＤＦＡの状態に対応し、各列は、入力シンボルに対応する。図３のＤＦＡについての遷移テーブルが、図４に示されている。ＤＦＡのためのアルファベットΣが、（多くのアプリケーションでしばしばそうであるように）ＡＳＣＩＩキャラクタセットである場合、図４の遷移テーブルは、２５６個の列を有する。図４の遷移テーブル内の各エントリは、次状態識別子を含む。したがって、図４の遷移テーブルは、以下のように機能する。ＤＦＡの現在状態がＢであり、次の入力シンボルが２である場合、「Ｄ」が現在状態Ｂと入力シンボル２によってインデックス付けされた次状態識別子であるので、遷移テーブルは、状態Ｄへの遷移を呼び出す。本明細書の説明では、シンボル符号化との混同を避けるため、状態は文字によってラベル付けされる。しかし、実際には、状態は遷移テーブル内で整数インデックスによって一般に表されることに留意されたい。

【発明の開示】
【発明が解決しようとする課題】
【００１７】
パイプライン化アーキテクチャでＤＦＡを実施するためのパターンマッチング技法は、本明細書で開示される新規なパターンマッチングアーキテクチャによって大きく改善され得ると、発明者らは本明細書において確信する。
【課題を解決するための手段】
【００１８】
本発明の一態様によれば、すべての状態依存（反復、フィードバック依存）動作をパイプラインの最終ステージに委ねるパイプライン化戦略が開示される。好ましくは、遷移テーブル検索は、ＤＦＡによって処理される入力シンボルに対応するすべての遷移テーブルエントリを取り出すように動作する。遷移テーブルメモリからの遷移エントリの取り出しは、ＤＦＡの現在状態に基づいていない。代わりに、遷移テーブルメモリからの取り出しは、処理される入力シンボルに対応するデータに基づいて、１組の保存された遷移エントリを取り出すように動作する。
【００１９】
入力データストリームの入力シンボルを同値類（ｅｑｕｉｖａｌｅｎｃｅ　ｃｌａｓｓ）識別子（ＥＣＩ）にマッピングするためにアルファベット符号化が使用される好ましい一実施形態では、これらの遷移テーブルエントリは、ＥＣＩに対応するデータによって、１つまたは複数の入力シンボルに間接的にインデックス付けされる。この改良は、単一サイクル状態遷移決定の実行を可能にし、より複雑な圧縮および符号化技法の使用を可能にし、アーキテクチャのスループットおよびスケーラビリティを向上させる。
【００２０】
本発明の別の態様によれば、遷移テーブルの遷移は、好ましくは、遷移を引き起こした入力シンボルの受け取り時に、入力シンボル文字列のパターンとのマッチが発生したかどうかを表すマッチフラグを含む。同様に、遷移テーブルの遷移は、好ましくは、遷移を引き起こした入力シンボルの受け取り時に、マッチングプロセスがリスタートしたかどうかを表すマッチリスタートフラグを含む。アクセプト状態は取り除かれ、遷移のマッチフラグ内にまとめられ得るので、各遷移におけるマッチフラグの存在は、ＤＦＡ内の状態の数が、従来のＤＦＡに比べて減少されることを可能にする。マッチリスタートフラグの存在は、ＤＦＡが、非位置固定パターンとマッチする入力ストリームの部分文字列を識別することを可能にする。併せて、遷移におけるこれらのフラグの存在は、本発明の別の態様に貢献し、その態様では、好ましいＤＦＡは、サイクル当たりに処理されるバイトの数をスケールアップする能力を有するように構成される。状態遷移は、（クロックサイクル当たり単一の入力シンボルのみを処理するように制限されるのではなく）ｍ個の入力シンボルからなる系列によってトリガされることができ、ｍは１以上である。本明細書で開示されるように、遷移がマッチフラグおよびマッチリスタートフラグを含む含み方のため、ＤＦＡは、ＤＦＡによってグループとして処理されるｍ個の入力シンボルからなる系列に属する先行または中間入力シンボルの結果として、マッチが入力ストリーム内のどこでいつ発生したかを依然として検出することができる。
【００２１】
本発明のまた別の態様によれば、高スループットＤＦＡを実現するのに必要とされる資源を著しく低減するために、増加スケーリング技法、圧縮技法、および文字符号化技法が使用される。例えば、ＤＦＡ遷移テーブルによって消費されるメモリの量を低減（すなわち圧縮）するために、ランレングス符号化が使用されることができる。さらに、その後、ＤＦＡについての次状態を決定するために、ランレングス符号化された遷移に基づいて、状態選択ロジックが動作する。遷移テーブルメモリの検討対象部分から、処理される入力シンボルのＥＣＩに対応する遷移を含まないワードを除去するために、マスキングが状態選択ロジックで使用されることができる。
【００２２】

また、本発明のまた別の態様によれば、ＥＣＩを遷移テーブルメモリ内の遷移にマッピングするために、インダイレクションレイヤが使用されることができる。このインダイレクションレイヤは、遷移テーブルメモリ内の遷移エントリについてのランレングス符号化プロセスを最適化するのに、また遷移テーブルメモリに対して必要なアクセスの回数を最小化し得るように、ランレングス符号化された遷移エントリを遷移テーブルメモリのワードに効果的にパッキングするプロセスを最適化するのに有効な、様々な最適化技法の使用を可能にする。インダイレクションを使用して、インダイレクションテーブルメモリ内のインダイレクションエントリは、遷移テーブルメモリ内の遷移エントリへのＥＣＩのマッピングを、遷移テーブルメモリ内の遷移エントリに対して実行されたいずれの最適化プロセスも考慮に入れて構成するように、データ投入されることができる。

【００２３】

さらに、本発明の別の態様によれば、遷移テーブル内でＤＦＡ状態を順序よく配置し、それによって、ランレングス符号化された遷移の効率を高めてＤＦＡのメモリ要件を改善するための最適化アルゴリズムが、本明細書で開示される。

【００２４】

さらにまた、複数のメモリワードにまたがる共通の対応する入力シンボル（またはＥＣＩなどのその派生物）を共有する遷移テーブルエントリの数が最小化されるように、遷移テーブルエントリをメモリワード内に効率的にパッキングするための最適化アルゴリズムが、本明細書で開示される。メモリの効率的なパッキングは、１つまたは複数の入力シンボルを処理するときに必要とされるメモリアクセスの回数を低減することができるので、このメモリパッキングプロセスは、ＤＦＡのスループットを改善するように動作する。

【００２５】

本発明の別の態様によれば、検索中に適用されるパターンは、パイプラインアーキテクチャ自体のロジックを変更することなく、動的に変更されることができる。正規表現コンパイラは、新しい正規表現に対してパターンマッチングパイプラインを再プログラムするために、遷移テーブルメモリ、インダイレクションテーブル、ＥＣＩマッピングテーブル、および関連レジスタにデータ投入しさえすればよい。

【００２６】

本明細書で提示されるＤＦＡ設計に対する改良に基づくことで、本発明の好ましい実施形態によって達成されるスループットおよび密度は、その他の知られたパターンマッチングソリューションを大きく上回ると、発明者らは本明細書において確信する。

【００２７】

本発明の上記およびその他の特徴が、これ以降で説明され、それらは、以下の明細書および図面を検討することで、当業者には明らかとなろう。

【発明を実施するための最良の形態】

【００２８】

図５は、本発明の好ましい実施形態の概要を示している。好ましい実施形態のアーキテクチャは、正規表現回路５０２内に示されており、正規表現回路５０２は、複数の連続的な入力シンボルを含む入力データストリームに基づいて動作するパターンマッチング回路として機能する。好ましくは、正規表現回路５０２は、ハードウェアロジックで実装される（例えば、ＦＰＧＡなどの再構成可能なハードウェアロジック、またはＡＳＩＣなどの再構成不可能なハードウェアロジック）。本発明の実施者によって望まれるのであれば、正規表現回路５０２ｉの１つまたは複数が同じデバイス上に実装され得ることに留意されたく、図２４には、そのことも反映されている。また、正規表現回路は、マルチプロセッサシステムなど、その他のパイプライン化アーキテクチャで実施されることもでき、その場合、各プロセッサは、正規表現回路のパイプラインステージとして機能する。そのような例においては、パイプラインに属する異なるプロセッサは、互いにネットワーク接続されることができる。

【００２９】

正規表現回路のデータテーブルおよび関連レジスタは、好ましくは、正規表現コンパイ

ラ５００の出力によってデータ投入される。正規表現コンパイラ５００は、本明細書で説明されるような正規表現回路５０２によって実現されるＤＦＡを生成するために、指定された（好ましくはユーザ指定された）正規表現を処理するように動作する。好ましくは、正規表現コンパイラ５００は、パーソナルコンピュータ、ワークステーション、またはサーバのＣＰＵなどの汎用プロセッサによって実行されるソフトウェアで実施される。

【００３０】

正規表現コンパイラ５００は、ネットワークデータ通信、直接インタフェース、およびシステムバスを含むが、これらに限定されない、任意の適切なデータ通信技法によって、正規表現回路５０２と通信する。

【００３１】

正規表現回路５０２は、好ましくは、複数のパイプラインステージを介して、１つまたは複数の指定された正規表現によって定義されるＤＦＡを実現する。第１のパイプラインステージは、好ましくは、ｍ個の入力シンボルからなる入力からＥＣＩ出力を生成するアルファベット符号化ステージ５０４であり、ｍは、１以上の整数とすることができる。第２のパイプラインステージは、好ましくは、インダイレクションテーブルメモリステージ５０６である。インダイレクションテーブルメモリ状態５０６は、様々な方法でアドレシングされることができる。好ましくは、インダイレクションテーブルメモリステージ５０６は、アルファベット符号化ステージ５０４のＥＣＩ出力によって直接的にアドレシングされる。第３のパイプラインステージは、インダイレクションテーブルメモリステージ５０６の出力からインダイレクションテーブルエントリを受け取り、受け取られたインダイレクションエントリを、遷移テーブルメモリステージ５１０内の１つまたは複数のアドレスに変換するように動作する、遷移テーブルロジックステージ５０８である。遷移テーブルロジックステージ５０８は、好ましくは、（マスキング動作に関連して以下で説明されるように）受け取られたインダイレクションテーブルエントリを、状態選択ロジックステージが遷移テーブルメモリステージ５１０からの出力を処理するときに、状態選択ロジックステージ５１２によって使用されるデータにも変換する。

【００３２】

遷移テーブルメモリステージ５１０は、ＤＦＡの次状態を決定し、またマッチが見出されたかどうかを決定するために、ＤＦＡによって使用される遷移を保存する。状態選択ロジックステージ５１２は、遷移テーブルメモリステージ５１０から出力された遷移エントリの１つまたは複数を受け取り、またＤＦＡの現在状態と受け取られた遷移とに基づいてＤＦＡの次状態を決定するように動作する。任意選択で、以下で説明される状態選択ロジックステージ５１２内のマスキング動作５１４および５１６は、独立したマスキングパイプラインステージ、または２つの独立したマスキングパイプラインステージ（先頭マスキングパイプラインステージおよび終端マスキングパイプラインステージ）に分割されることができる。これらのステージの各々についてのさらなる詳細が、本明細書で提示される。

【００３３】

高スループットＤＦＡ

従来のＤＦＡは、一度に１つの入力シンボル（バイト）を処理し、次状態を決定するために各バイトに基づいてテーブル検索を実行する。しかし、現代の通信インタフェースおよび相互接続は、サイクル当たり複数のバイトをしばしば転送し、そのことが、より高いスループットの達成に関して、従来のＤＦＡを「ボトルネック」にする。スループットとは、データストリームが処理され得る速度、すなわち、設計およびその実施によって受け入れられ得る１秒当たりのバイト数のことである。

【００３４】

従来のＤＦＡの拡張は、ｍ個のシンボルからなる文字列に基づいた単一の遷移の実行を可能にするＤＦＡである。ＣｌａｒｋおよびＳｃｈｉｍｍｅｌ、「Ｓｃａｌａｂｌｅ ｐａｔｔｅｒｎ ｍａｔｃｈｉｎｇ ｆｏｒ ｈｉｇｈ ｓｐｅｅｄ ｎｅｔｗｏｒｋｓ」、ＩＥＥＥ Ｓｙｍｐｏｓｉｕｍ ｏｎ Ｆｉｅｌｄ－Ｐｒｏｇｒａｍｍａｂｌｅ Ｃｕ

ｓｔｏｍ Ｃｏｍｐｕｔｉｎｇ Ｍａｃｈｉｎｅｓ、２００４年４月を参照されたく、その開示の全体は、参照により本明細書に組み込まれる。すなわち、ＤＦＡは、ｍ個の入力シンボルをグループとして、入力ストリームを処理する。形式的には、この適応は、アルファベットΣ^mに基づいたＤＦＡをもたらし、対応する遷移テーブルは、｜Ｑ｜｜Σ｜^{m}のサイズをもつ。この明らかに劇的な資源要件の増大は、本明細書で説明される圧縮技法によって緩和される。便宜上、δ^mは、長さｍの系列に基づいて動作する遷移関数を表すとし、$\delta=\delta^1$である。

【００３５】

例示的な一例として、一度に２バイト（ｍ＝２）を処理することによって、図３に示されたＤＦＡの実効スループットを２倍にすることについて検討する。図４のテーブルに基づいて、現在状態がＥである場合、入力系列「２＄」は、状態Ｂへの遷移をもたらし、すなわち、δ^2（Ｅ，２＄）＝δ（δ（Ｅ，２），＄）＝Ｂである。そのような２文字系列をすべて考慮することによって、以下で説明されるように、完全な遷移テーブルが、このスループットがより高いＤＦＡについて計算されることができる。

【００３６】

一般に、与えられたＤＦＡについてδ^mを構成するアルゴリズムは単純である。状態の集合は、変化せず、遷移関数（テーブル）は、各状態からの進行を長さｍのあらゆる可能な系列についてシミュレーションすることによって計算される。そのアルゴリズムは、サイズがθ（｜Ｑ｜｜Σ｜^{m}）のテーブルを計算するのに、θ（｜Ｑ｜｜Σ｜^{m}ｍ）の時間を要する。より高速なアルゴリズムは、以下の形式の動的プログラミングによって取得されることができる。

【数２】

$$m=2^i, i>0 \text{ かつ文字列 } x_i x_r, |x_l|=|x_r|=\frac{m}{2}$$

について検討する。その場合、

【数３】

$$\forall q \delta^m(q, x)=\delta^{\frac{m}{2}}(\delta^{\frac{m}{2}}(q,x_i),x_r)$$

となる。上述の説に基づいたアルゴリズムが、図６に示されている。

【００３７】

スループットがより高いＤＦＡを取得するため、図６のアルゴリズムは、繰り返して呼び出されることができ、各回のたび、サイクル当たり処理されるシンボルの有効数は２倍になる。これは、δ^mをＯ（｜Ｑ｜｜Σ｜^{m}ｌｏｇ ｍ）まで計算する複雑さを低減する。さらに、冗長な列を識別することによって、各テーブルの有効アルファベットは、以下の「アルファベット符号化」セクションで説明されるような符号化を用いて、実際には著しく減少されることができる。

【００３８】

高スループットＤＦＡ：アクセプト

スループットがより高いＤＦＡは、サイクル当たり複数の遷移を実行するので、遷移中に元のＤＦＡのアクセプト状態を通過することがあり得る。したがって、一連の遷移（ｔｒａｃｅ）中にアクセプト状態が通過されたかどうかを含むように、遷移関数を拡大する。

$\delta^m: Q\times\Sigma^m\rightarrow Q\times\{0,1\}$

値域の第２の成分は、遷移を引き起こしたシンボルの系列が、元のＤＦＡにアクセプト状態を通過させる非空のプレフィックスを含むかどうかを示す。

【００３９】

「アクセプト」（マッチ）情報は、スループットがより高いＤＦＡの辺に関連付けられるので、この形式の遷移関数は、アクセプト状態Ａの集合の必要性を回避する。これは、

以下の「ストライドと符号化の相乗的組み合わせ」セクションで定義される修正ＤＦＡによって形式化される。

【００４０】

好ましいＤＦＡは、どの中間シンボルが実際にアクセプト（マッチ）を発生させたかを追跡していないので、ｍ＞１の場合、アクセプトは不明確である。速度を優先するため、高スループットＤＦＡは、不明確なアクセプトを許容して、アクセプト点の正確な決定をソフトウェア後処理に委ねるように構成されることができる。

【００４１】

高スループットＤＦＡ：リスタート

先に説明されたように、正規表現のためのパターンマッチングＤＦＡは、好ましくは、ターゲット文字列のいたる所でマッチが発生することを可能にする遷移を用いて拡大される。マッチはターゲット内のどこを開始位置としても発生し得るので、アクセプトは、ターゲット内におけるマッチの開始位置を報告すべきである。開始点がいつ設定されるかは、図３のオートマトンでは明らかでない。例えば、状態Ａへのすべての遷移は、開始点を設定するが、「＄」に基づいた状態Ｂへの遷移もそうである。ＥからＡへの遷移を考えると、「～」または「．」はリスタートであるが、数字（ｄｉｇｉｔ）はアクセプトである。

【００４２】

マッチの位置独立を達成するために導入されるλ遷移は、通常の構成を介してＤＦＡに変換され得るＮＦＡをもたらす。以下の「ストライドと符号化の相乗的組み合わせ」セクションは、オートマトンのマッチングプロセスをリスタートするためにのみ役立つ遷移を識別するために、その構成をどのように修正すべきかを説明する。

【００４３】

形式的には、遷移関数は、もう一度、今度はいつリスタートが発生するかを示すために拡大される。

$$\delta^{1} : Q \times \Sigma^{m} \rightarrow Q \times \{0, 1\} \times \{0, 1\}$$

第１のフラグは、リスタート遷移（「マッチリスタート」フラグ）を表し、第２のフラグは、アクセプト遷移（「マッチ」フラグ）を表す。したがって、ＤＦＡ図は、これ以降、緑色の辺を用いてリスタート遷移を、赤色の辺を用いてアクセプト辺を示す。例えば、図７は、一度に１つのシンボルを処理し、図１によって表される言語を認識するオートマトンについての図の例示的な一例を示している。任意選択で、状態Ａへの遷移に関する辺は、黒色として符号化され、しかるべくフラグ設定されることができ、状態Ｂへの遷移に関する辺だけが、緑色として符号化され、しかるべくフラグ設定される。色付けされた辺を有するＤＦＡの動作は、以下のようになる。オートマトンは、マッチの開始および終了を記録するための文脈変数（ｃｏｎｔｅｘｔ　ｖａｒｉａｂｌｅ）ｂおよびｅを含み、最初、ｂ＝ｅ＝０であり、ターゲットの第１のシンボルのインデックスは１である。これらの変数は、マッチング文字列の位置が、図２４に示されるような文脈バッファ（ｃｏｎｔｅｘｔ　ｂｕｆｆｅｒ）内で見出されることを可能にする。その後、以下のように遷移が実行される。

黒色：ｅがｍだけ、すなわち、遷移を引き起こした入力文字列の長さである、オートマトンのストライド（ｓｔｒｉｄｅ）だけ進められる。図６では、ｍ＝１である。マッチは進行中であり、ここまで関与しているターゲット文字列の部分は、両端を含んで、位置ｂで開始し、位置ｅで終了する。

赤色のみ：ｅがｍだけ増やされ、マッチが宣言される。マッチを引き起こしたターゲット部分文字列は、位置ｂで開始し、位置ｅで終了する。

緑色のみ：ｂがｍだけ増やされ、ｅはｂと同じに設定される。オートマトンは、マッチプロセスをリスタートする。

赤色および緑色：緑色動作の前に、赤色動作が実行される。

【００４４】

図９は、図７のＤＦＡに関連する遷移テーブルを示しており、遷移テーブルの各遷移エ

ントリは、次状態識別子に加えて、マッチフラグおよびマッチリスタートフラグを含む。アクセプトについての情報は、辺（δ関数）に関連付けられるので、色付けされた辺を有するＤＦＡは、正準ＤＦＡよりも少ない状態を有することができる。

【００４５】

マッチフラグおよびマッチリスタートフラグの使用は、ＤＦＡを拡大して、サイクル当たり複数の入力シンボルを処理する場合に特に有用である。図１０（ｂ）は、図１の正規表現のための遷移テーブルを示しており、ｍは２に等しい。図１０（ａ）は、この例についてのシンボル符号化を示している。したがって、サイクル当たり複数の入力シンボルを処理する場合であっても、フラグは状態よりもむしろマッチステータス情報を保つので、ＤＦＡは、ｍ個の入力シンボルからなるグループの先行または中間シンボル上でマッチおよびリスタートがいつ発生したかを検出することができる。

【００４６】

アルファベット符号化

上で説明されたように、遷移テーブル（δ）のサイズは、各サイクルで消費される入力系列の長さに伴って指数関数的に増大する。本セクションでは、シンボルアルファベットを符号化するための技法が提示され、その目標は、遷移テーブルのサイズを低減し、ひいてはサイクル当たりに処理されるシンボルの数を最大化することである。

【００４７】

しばしば、与えられた正規表現で使用されるシンボルの集合は、検索ターゲットのアルファベットΣと比べて小さい。ターゲット内に存在するが、パターン内には示されないシンボルは、正規表現に関してＤＦＡにおいて必然的に同じ扱いを与えられる。より一般的には、シンボルのある集合上でのＤＦＡの挙動は、その集合内のすべてのシンボルについて同一となる場合があり得る。例示的な一例として、図１の正規表現は、ＡＳＣＩＩキャラクタセットのわずかな部分のみを使用する。数字についての遷移は、図３に示されたＤＦＡでは同じであり、シンボル「～」は、集合｛０，１，２，３，４，５，６，７，８，９，＄，．｝内にないＡＳＣＩＩセットのすべてのシンボルを表す。

【００４８】

正規表現は、「［０－９］」のように、文字クラスを明示的に表現することができるが、より汎用的な手法は、等価の状態－遷移挙動（ｓｔａｔｅ－ｔｒａｎｓｉｔｉｏｎ ｂｅｈａｖｉｏｒ）についてＤＦＡを分析することによって達成される。形式的には、

（∃ａ∈Σ）（∃ｂ∈Σ）（∀ｑ∈Ｑ）δ（ｑ，ａ）＝δ（ｑ，ｂ）

である場合、ａとｂは「遷移同値（ｔｒａｎｓｉｔｉｏｎ ｅｑｕｉｖａｌｅｎｔ）」であると言われることができる。

【００４９】

遷移テーブル

【数４】

$$\delta : Q \times \Sigma \mapsto Q$$

を所与として、Σを同値類に区分するためのＯ（｜Ｓｉｇｍａ｜[2]｜Ｑ｜）のアルゴリズムが、図８に示されている。図１の例を使用すると、アルゴリズムは、図１で暗示されたシンボルの同値類を生成して、図９（ｂ）に示される４つの列を形成する。区分は、整数からなる集合Ｋと、ΣからＫへのマッピングｋによって表される。アルファベットシンボルの同値性は状態同値によって決定されるので、最良の結果は、状態同値がすでに決定された正準ＤＦＡに、δが対応する場合に得られる。

【００５０】

関連する正規表現を検査する代わりに、ＤＦＡを使用して同値類を計算することは、以下の理由から好ましい。

【００５１】

正規表現は、同値類を暗示し得る範囲またはその他の表記法（ｇｅｓｔｕｒｅ）を使用せずに指定されることができる。例示的な一例として、ａとｂは、正規表現「（ａｃ）｜

（ｂｃ）」に関して、ＤＦＡにおいて同値とされ得るが、それらの同値性は、正規表現では明白ではない。

【００５２】

同値類は、正規表現の局所的検査によってはしばしば決定することができない。例示的な一例として、正規表現「［０－９］ａ｜５ｃ」は、句「［０－９］」を含むが、その範囲の１つの要素（５）は、その他の句のために、その他のシンボルと同じ遷移を共有しない。この場合の適切な区分は、｛｛０，１，２，３，４，６，７，８，９｝，｛５｝｝である。

【００５３】

形式的には、図８の関数は、区分、すなわち集合を生成し、その各要素は、ＤＦＡにおいて同じに扱われるΣ内のシンボルからなる集合である。関数は、その区分のより便利な形式をｋにより生成し、それは、各シンボルをその同値類を表す整数にマッピングする（整数は「同値類識別子」（ＥＣＩ）として機能する）。その後、シンボル符号化に基づいたＤＦＡは、以下の２つのステージで動作する。第１に、次の入力シンボルが、図８のアルゴリズムによって決定されるその同値類を表すＥＣＩに、ｋによってマッピングされる。第２に、ＤＦＡは、その現在状態とＥＣＩを使用して、ＤＦＡの次状態を決定することができる。

【００５４】

これまでに提示されたアイデアに基づいて、図７は、一度に単一のシンボルを処理するＤＦＡの例示的な一例を示しており、赤色の辺は「アクセプト」を表し、緑色の辺は「リスタート」を表す。図８のアルゴリズムを使用する分析は、図９（ａ）に示されるシンボル符号化と、図９（ｂ）に示される遷移テーブルをもたらす。図９（ｂ）に示されるように、遷移テーブルの各エントリは、ＤＦＡ状態に対応するデータと、ｍ個の入力シンボルからなるグループについてのＥＣＩに対応するデータによってインデックス付けされる。図１０（ａ）および図１０（ｂ）は、２つの入力シンボルが一度に処理される、図１の正規表現についての、同等のＥＣＩテーブルおよび遷移テーブルを示している。

【００５５】

遷移テーブル内の各遷移は、次状態識別子と、マッチリスタートフラグと、マッチフラグとを含む３つ組である。例えば、状態ＤとＥＣＩ　０とによってインデックス付けされた遷移は、（Ｂ，１，０）であり、Ｂが次状態識別子、１がマッチリスタートフラグ、０がマッチフラグである。したがって、ＥＣＩ　０によって引き起こされる状態Ｄから状態Ｂへの遷移は、ＥＣＩ　０はマッチを発生させないが、マッチングプロセスをリスタートさせるというように、解釈されることができる。

【００５６】

ストライドと符号化の相乗的組み合わせ

上で説明されたスループット向上とアルファベット符号化のアイデアは、今から組み合わされて、（テーブルを構成する際の）時間および（実行時にテーブルを実現する際の）空間を節約するために、サイクル当たり複数のバイトを消費し、その入力を符号化するアルゴリズムをもたらす。

【００５７】

そのような新しい高スループットＤＦＡ^{m}は、形式的に６つ組（Ｑ，Σ，q_0，Ｋ，ｋ，δ）として今は記述されることができ、ここで、

Ｑは、状態の有限集合であり、

Σは、ターゲットの入力シンボルのアルファベットであり、

$q_0 \in Q$は、初期状態であり、

Ｋは、サイズがＱ（｜Σ｜^{m}）である（しかし実際にはより小さいことが予想される）整数からなる集合であり、

ｋは、一度にｍ個の入力シンボルをそれらの符号化にマッピングする、関数

【数５】

$$\Sigma^m \mapsto K$$

であり、

δは、現在状態およびｍ個のシンボルからなる次の部分文字列を、次状態、リスタートの可能性、およびアクセプトの可能性にマッピングする、関数

【数６】

$$Q \times K \mapsto Q \times \{0, 1\} \times \{0, 1\}$$

である。

【００５８】

一連の変換は、正規表現ｒで開始し、ＤＦＡ^{m}を取得するために以下のステップを実行する。

１．１つまたは複数の正規表現ｒについて通常の方法（上で引用されたＨｏｐｃｒｏｆｔおよびＵｌｌｍａｎの参考文献を参照されたい）でＤＦＡ　ｄが構成される。例えば、上で説明されたように、図１の正規表現は、図３に示されたＤＦＡをもたらす。

２．ターゲット内の任意の位置での開始に基づいて、オートマトンが受け入れることが可能な遷移ｐの集合が計算される。これは、各状態についてｑ$_{0}$へのλ遷移をシミュレーションすることによって達成される。具体的には、ρは以下のように計算される。

【数７】

$$\rho \leftarrow \phi$$

$$\textbf{foreach } \phi \in Q \textbf{ do}$$

$$\textbf{foreach } a \in \Sigma \textbf{ do } \rho \leftarrow \rho \cup \{(p, a) \mapsto \delta(go, a)\}$$

３．ｄおよびｐから、基本ＤＦＡ^{1}が、図１１のアルゴリズムによって構成される。

４．状態最小化が、標準アルゴリズム（上で引用されたＨｏｐｃｒｏｆｔおよびＵｌｌｍａｎの参考文献を参照されたい）によって、高スループットＤＦＡに対して実行されるが、ただし、状態は、従来のＤＦＡでのように最終状態かそれとも非最終状態かによって分割される代わりに、入来する辺の色（黒、緑、赤）によって最初に分割される。

５．ＤＦＡ^{k}を所与として、アルファベット符号化を伴う、スループットがより高いＤＦＡ$^{2^{k}}$が、図６に示されたアルゴリズムによって構成される。

【００５９】

ランレングス符号化による遷移テーブル圧縮

好ましい高スループットＤＦＡのための遷移テーブルは、｜Ｋ｜×｜Ｑ｜個のエントリを含むことができる。状態最小化は、｜Ｑ｜を最小化することを試み、より高いスループットとアルファベット符号化の組み合わせに関する先の説明は、｜Ｋ｜を最小化することを試みる。それにも関わらず、ストレージ資源は一般に制限され、したがって、できるだけ多くのテーブルを収容するための技法が、検討されるべきである。以下では、テーブル自体をどのように圧縮するかを説明することによって、この問題を検討する。

【００６０】

上述の説明に基づいて、遷移テーブルセルは、３つ組（次状態、スタートフラグ、アクセプトフラグ）を含む。ランレングス符号化は、十分な冗長性を見せるシンボル系列についてストレージ要件を低減することができる単純な技法である。そのアイデアは、文字列ａ^{n}を、ランレングスｎおよびシンボルａとして符号化するというものであり、表記ｎ（ａ）が使用されることができる。したがって、文字列ａａａａｂｂｂｃｂｂａａａは、４（ａ）３（ｂ）１（ｃ）２（ｂ）３（ａ）としてランレングス符号化される。各シンボルおよび各ランレングスが１バイトの記憶域を必要とする場合、ランレングス符号化は、この例では、ストレージ要件を３バイトだけ低減する（１３バイトから１０バイト）。

【００６１】

図９（ｂ）の例を調べると、遷移テーブルの列にはランレングス符号化のための好条件が豊富に存在する。符号化ＥＣＩシンボル０および３の場合、テーブルは、直前の状態といずれも同じ３つ組を指定しており、そのため、ＥＣＩ　０および３によってインデックス付けされるテーブル内の遷移に対するランレングス符号化プレフィックスは、ともに「５」である。一般に、遷移テーブルにおいて期待されることは、共通の「次状態」挙動である。したがって、遷移テーブルの各列内にただ１つしかないエントリの数は、ＤＦＡにおける状態の数よりも一般に少ない。図１２は、図９の遷移テーブルのランレングス符号化バージョンを含んでいる。

【００６２】

列圧縮は記憶域を節約するが、所望のエントリを取得するために遷移テーブルにアクセスする際のコストを増大させるように思える。圧縮に先立ち、行は、現在状態によってインデックス付けされ、列は、ＥＣＩによってインデックス付けされる。図１２に示されるように、列がランレングス符号化されると、各列の圧縮された内容は、図１３に示されるように、隣接して保存される。この例では、状態自体も、整数として符号化されている（Ａは０、Ｂは１によって表されるなど）。ＤＦＡの次状態および動作を決定するために、今は２つのステップが存在する。

１．入力から符号化された次のＥＣＩに基づいて、図１３に示される記憶域レイアウト内で、エントリの関連範囲が見出される。この検索は、ＥＣＩから記憶域レイアウトにおけるオフセットへのマッピングを使用して実行されることができる。エントリの範囲は、図１２からの圧縮された列である。図１３では、下線が引かれたエントリが、図１２からのＥＣＩ　１に対応する。

２．現在状態に基づいて、次状態および動作フラグが、エントリの関連範囲内で見出されなければならない。状態選択と呼ばれるこのロジックは、どのエントリが現在状態のインデックスに対応するかを見出すために、エントリを必然的に伸張する必要がある。

【００６３】

圧縮列全体が利用可能である場合、図１４に示される回路は、適切な組エントリを決定するために圧縮形式および現在状態を解釈することによって、状態選択がどのように実現され得るかの一例を示している。ランレングス符号化列内の各エントリについて、そのランレングスプレフィックスと先行するエントリのランレングスプレフィックスの合計が計算される。その後、現在状態インデックスが、各合計と比較され、その合計が現在状態インデックスよりも大きい最初（図１４の例では最も左）のエントリが、プライオリティ符号器によって選択される。その後、プライオリティ符号器は、ＤＦＡのための次状態を決定することができる。

【００６４】

図１４は、図１２のＥＣＩ　１に対して進行中の状態選択の例示的な一例を示している。各「＜」ノードは、現在状態インデックス（この例では３、すなわち状態Ｄ）を、圧縮列内のランレングスプレフィックスの合計と比較する。状態インデックス０（状態Ａ）が現在状態として供給された場合、３つの比較器すべてが、「１」を出力し、プライオリティ符号器は、最も左のものを取って、内容として（Ａ，１，０）を選択する。図１４では、現在状態は３であり、それは、第２の合計（３＋１）および第３の合計（３＋１＋１）より小さいので、右２つの比較器が、「１」を出力する。プライオリティ符号器は、最も左のものを取り、その結果、ＥＣＩ　１および状態３の検索として、（Ｃ，０，０）が選択される。

【００６５】

メモリにおける可変長列のサポート

図１３に示された記憶域レイアウトは、物理メモリにマッピングされなければならず、物理メモリにおいて、テーブル全体は、１回のメモリアクセスでフェッチされるには大き過ぎる。フィールドプログラマブルゲートアレイ（ＦＰＧＡ）および同様のデバイスは、サイズおよびワード長に関して構成可能なメモリバンクをしばしばサポートする。さらに、与えられたエントリのサイズは、各フィールド（ランレングス、次状態識別子、マッチ

リスタートフラグ、マッチフラグ）のために割り当てられたビットの数に依存する。以下の分析は、サイクル当たりｘ個の遷移テーブルエントリが取り出され得るという全般的な仮定に基づいている。シングルポートメモリでは、これは、ｘがワード当たりの遷移エントリ数に一致することを意味する。マルチポートメモリでは、これは、ｘがポート数にワード当たりの遷移エントリ数を乗じた値に一致することを意味する。一例として、サイクル当たり５回のアクセスをサポートし、ワード当たり３つのエントリを保有する物理メモリは、好ましい実施形態では、ｘ＝５×３＝１５と設定することによって適合される。しかし、サイクル当たり１回のみのアクセスをサポートし、ワード当たり３つのエントリを保有する物理メモリは、好ましい実施形態では、ｘ＝３と設定することによって適合される。

【００６６】

いくつかのアーキテクチャは、ｘの選択可能性に関して、その他のアーキテクチャよりも大きな柔軟性を提供する。例えば、ＦＰＧＡブロックＲＡＭのビットは、時には、ワードの数および各ワードの長さに関して構成可能である。以下の考察は、ｘの最良の選択に対して一般に適用される。

【００６７】

メモリアクセスは、ｘをできるだけ高く押し上げる（ｄｒｉｖｉｎｇ）ことによって、一般に低減される。

【００６８】

図１４の論理回路は、一度に処理されなければならないエントリの数に伴って成長する。回路の全体的サイズに及ぼすその成長の影響は、ターゲットプラットフォームおよび実装に依存する。図１４の論理回路を実現するためには、相当なＦＰＧＡ資源が必要とされる。

【００６９】

メモリにおける可変長列のサポート：遷移テーブルメモリ

ｘが選択されると、圧縮列は、できるだけコンパクトに物理メモリ内に配置される。図１５（ｂ）は、遷移テーブルの列が、ｘ＝３のメモリ内にパッキングされた一例を示している。メモリ内の各ワードは、メモリアドレスによってインデックス付けされる。例えば、メモリアドレス０によってインデックス付けされたワードは、以下の遷移、すなわち、５（Ｂ，１，０）、３（Ａ，１，０）、および１（Ｃ，０，０）を含む。各列の変化する長さのため、与えられた列は、行内のいずれのエントリからでも開始することがある。

【００７０】

遷移テーブルにインダイレクションレイヤを導入することによって、ランレングス符号化と、遷移テーブルメモリ（ＴＴＭ）内へのエントリのコンパクト配置とによって提供される、メモリ効率性を利用することが可能である。図１５（ａ）は、各ＥＣＩについて１つのエントリを含む、そのようなインダイレクションテーブルの一例を示している。ＥＣＩは連続的に割り当てられ得るので、インダイレクションテーブルは、与えられた入力文字列についてのＥＣＩ値を使用して、直接的にアドレシングされることができる。各インダイレクションテーブルエントリは、以下のものから構成されることができる。

ポインタ：ランレングス符号化された遷移テーブル列の第１のエントリを含む、遷移テーブルメモリ内のメモリワードのアドレス。

遷移インデックス：ランレングス符号化された遷移テーブル列に対する第１のメモリワードにおける、その列の第１のエントリのインデックス。

遷移カウント：（または略して「カウント」）ランレングス符号化された遷移テーブル列内のエントリの数。

【００７１】

入力シンボルのＥＣＩを使用してインダイレクションテーブルエントリが取り出されると、ＴＴＭから第１のメモリワードを読み取るために、取り出されたエントリ内のポインタが使用される。ｘがＴＴＭ内のメモリワード当たりのエントリの数であることを思い出されたい。アドレス遷移インデックスから開始して、ＴＴＭからｗ個の連続するワードを

読み取ることによって、列全体がアクセスされ、ここで、ｗは、

【数８】

$$w = \left\lceil \frac{transition.count + transition.index}{x} \right\rceil \quad (1)$$

によって与えられる。遷移インデックス値および遷移カウント値は、最初のメモリワードのどのエントリから最後のメモリワードのどのエントリまでが列に関与するかを決定する。図１５の例では、各ＴＴＭワードは、３つのエントリを保存することが可能であり、エントリは、ランレングス符号化された遷移組である。図１５の場合、網掛け部分から分かるように、ＥＣＩ　１の列をフェッチするには、遷移テーブルメモリを２回読み取ることが必要であることが分かる。ＥＣＩ　１の遷移インデックスおよび遷移カウントの具体的な値は、列が、フェッチされた最初のワードの第２のエントリから始まり、フェッチされた最後のワードの第１のエントリまで続くことを示している。各列がインデックス０から始まるように配置することによって、ＴＴＭエントリが浪費される場合、アクセスの回数は、

【数９】

$$\left\lceil \frac{transition.count}{x} \right\rceil$$

まで減少させることができる。０≦遷移インデックス＜ｘであるので、ＴＴＭにおけるコンパクト記憶は、アクセス回数をたかだか１だけ増大させるに過ぎない。

【００７２】

以下で説明され、また図２５に示されるように、インダイレクションテーブルおよびＴＴＭへのアクセスは、互いにパイプライン化されることができ、また本発明の設計のその他のコンポーネントともパイプライン化されることができる。マルチポートメモリが利用可能である場合、両テーブルは、性能を悪化させることなく、同じ物理メモリ内に保存されることができる。しかし、インダイレクションレイヤは、ＴＴＭにおけるランレングス符号化列の内容の配置に応じて、状態遷移当たり様々な回数の遷移テーブルメモリアクセスをもたらす。特定のＥＣＩについて、メモリから列を取り出すためのメモリアクセスの回数は、直接アドレシング手法で必要とされる回数を超えることはできない。平均で、状態遷移当たりのメモリアクセスの回数は、かなり小さい。ランレングス符号化およびインダイレクションによって達成されるメモリ効率性は、インダイレクションテーブルおよび追加のランレングスフィールドを保存することによって背負い込むオーバヘッドを補償して余りあると、発明者らは一般に確信する。

【００７３】

さらに、各エントリは同じサイズであり、エントリ数は入力シンボル同値類の最大数に等しいので、インダイレクションテーブルのためのメモリの割り当ては、相対的に単純である。

【００７４】

メモリにおける可変長列のサポート：状態選択

好ましくはＴＴＭの効果的な記憶域レイアウトを考慮し、その他の最適化も提供する、状態選択論理回路の実施が、今から説明される。ＴＴＭは圧縮列のコンパクトな記憶を提供するが、状態選択ロジックは、より複雑になる。図１４に示されるロジックは、圧縮列が、無関係なエントリなしに同時にロジックに提示され得ることを仮定する。そのロジックは、ＴＴＭを使用して状態選択を実行するためには、以下の理由で、準最適である。

圧縮列がＴＴＭの複数のワードにまたがることがある。

圧縮列の先頭がＴＴＭワードの中間から始まることがある。したがって、先頭前のエントリは状態選択のために抑制されなければならない。

圧縮列の末尾がＴＴＭワードの末尾前に生じることがある。したがって、末尾後のエントリは状態選択のために抑制されなければならない。

【００７５】
図１４に示されるロジックは、各エントリの前のすべてのランレングスの合計を累算するために、加算器を使用する。ランレングスは各エントリ内で固定であるので、プレフィックス合計を事前計算し、ランレングスそのものの代わりに、それらを各組の「係数」として保存することによって、加算器は除去されることができる。合計を事前計算することによって、図１５に示されるテーブルは、図１６に示されるテーブルに変換される。
【００７６】
圧縮列の先頭および末尾を決定するために使用されるロジックの量も、削減されることができる。各列の先頭は、第１のエントリを含むＴＴＭワードと、第１のエントリのそのワード内でのインデックスとを提供する、ポインタフィールドおよび遷移インデックスフィールドを使用して、インダイレクションテーブルにおいて指定される。その後、圧縮列によって占められるワード数ｗが、式（１）によって与えられる。完全に占められた各ワードは、圧縮列のｘ個のエントリを含む。最後のワードでは、圧縮列によって占められる最大インデックスは、
（カウント＋インデックス－１） ｍｏｄ ｘ （２）
によって与えられる。
【００７７】
式２を計算するために、状態選択回路内にロジックが配備されることができる。しかし、ｘは、設計時パラメータである。ハードウェア定義言語（ＨＤＬ）コードの適切なパラメータ化によって、式２は、インダイレクションテーブルおよびＴＴＭテーブルが生成されるときに、計算されることができる。
【００７８】
したがって、計算ロジックの量は、各エントリについて以下の変数をインダイレクションテーブルに保存することによって、削減されることができる。
ポインタ：遷移テーブル列の第１のエントリを含むＴＴＭワードのアドレス。
先頭遷移インデックス：遷移テーブル列がまたがる第１のＴＴＭワードにおける、（遷移テーブル列の）第１のエントリのインデックス。
末尾遷移インデックス：遷移テーブル列がまたがる最後のＴＴＭワードにおける、（遷移テーブル列の）最後のエントリのインデックス。
［付加的な］ワードカウント：ｗ－１、ここで、ｗは式１によって計算される。
この例を続けると、図１７は、図１６に示される遷移テーブルに関する、インダイレクションテーブルおよびＴＴＭエントリを示している。基本的に、遷移カウント値は、ワードカウント値と、末尾遷移インデックス値とに変換される。この変換は、ＴＴＭの内容に影響を与えないが、これらのテーブルを処理するために必要とされるロジックを削減する。
【００７９】
図１８に示される状態選択ブロックロジックは、１つのＴＴＭワードにまたがる、２つのエントリを含む遷移テーブル列に基づいて動作する。各ワードは、４つのエントリを保存する。図１８に示されるワードカウントの出力は、現在の遷移テーブル行について検査されたメモリワードの数を反映する。現在のメモリワードが、その行がまたがる第１のワードである場合、先頭遷移インデックスマスクＲＯＭから１組のマスクビットを取り出すために、先頭遷移インデックスが使用される。これらのビットは、遷移テーブル列の部分ではない、メモリワード内の先行エントリをマスクオフするために使用される。マスキングは、ランレングス合計を強制的にゼロにすることによって達成される。現在のメモリワードが、その列がまたがる第１のワードでない場合、このステージでは、どのエントリもマスクされない。
【００８０】
次のステージは、遷移テーブル列の部分ではない、最後のメモリワード内のエントリを「マスク」する。遷移テーブル列の部分ではないエントリについてのランレングス合計は、強制的に最大ランレングスレジスタの値にされる。この値は、遷移テーブル列内のエントリの最大数（すなわち、符号化されていない遷移テーブルの列数であり、各符号化遷移

テーブル列内の最終エントリのランレングス合計の値でもある）を記録する。現在のメモリワードが、遷移テーブル列がまたがる最後のメモリワードである（ワードカウンタの値がワードカウントに等しい）場合、末尾遷移インデックスが、末尾遷移インデックスマスクＲＯＭに対するアドレスとして使用される。最後のメモリワードでない場合、このステージ中、どのエントリもマスクされない。後続エントリのランレングス合計を強制的に最大ランレングス合計値にすることは、次状態を選択するマルチプレクサのための選択ビットを生成するプライオリティ符号器を簡略化する。このマスキングプロセスは、小なり（ｌｅｓｓ－ｔｈａｎ）比較から以下の特性を有する出力ベクトルを生成し、その特性とは、最も左の「１」ビットのインデックスが次状態エントリのインデックスであり、このビットより右のビットはすべて「１」に設定される、というものである。先に言及されたように、スループットを向上させるために、マスキングステージはパイプライン化されることができる。一代替実施形態では、小なり比較、プライオリティ符号器、および次状態選択ロジックのみが、最終パイプラインステージで発生する必要がある。

【００８１】

最適化

高スループットの正規表現パターンマッチングエンジンを達成することは、本明細書で開示される、高スループットＤＦＡ、文字符号化、および遷移テーブル圧縮技法を開発する主要な動機である。以下では、何らかのメモリ効率性を犠牲にしてシステムのスループットを最適化する技法が検討され、したがって、以下の技法の各々は、ＴＴＭによって制約される。具体的には、ＴＴＭは、以下の制約を課す。

ワード当たりのエントリの数。

テーブル内のメモリワードの数。

テーブル内のエントリの総数。

【００８２】

本セクションで説明される最適化問題は、ビンパッキング問題（ｂｉｎ　ｐａｃｋｉｎｇ　ｐｒｏｂｌｅｍ）またはナップザック問題（ｋｎａｐｓａｃｋ　ｐｒｏｂｌｅｍ）の類に含まれる。Ｃｏｒｍｅｎ他、「Ｉｎｔｒｏｄｕｃｔｉｏｎ　ｔｏ　Ａｌｇｏｒｉｔｈｍｓ」、Ｃａｍｂｒｉｄｇｅ、ＭＡ、Ｔｈｅ　ＭＩＴ　Ｐｒｅｓｓ、１９９０年を参照されたく、その開示の全体は、参照により本明細書に組み込まれる。ワード当たりのエントリの数は、パッキング問題のビン（またはナップザック）サイズを定義する。符号化遷移テーブルの構造は、エントリの総数またはワード（ビンまたはナップザック）の総数が遷移テーブルメモリによって課される限界を超えない限り、テーブルを表すのに必要とされるエントリおよび／またはワードの総数を増やすことによってメモリアクセスの回数を最小化するために、変更されることができる。

【００８３】

最適化：スループットの最適化

検索のために必要とされるメモリアクセスの回数は、ＴＴＭ内における圧縮列の配置と、それらの列がアクセスされるパターンとによって決定される。パターンは、エンジン内における１組の正規表現と、エンジンによって処理される具体的な入力データとに依存する。ＤＦＡ^{m}の場合、ｍ個の入力シンボルは、１列の検索を引き起こすＥＣＩに変換される。メモリアクセスの回数は、符号化遷移テーブル内の列の長さと、列アクセスパターンとに依存する。列アクセスパターンは、エンジン内における正規表現（または１組の正規表現）と、入力データとに依存する。与えられた検索についてのメモリアクセスの総数は、

【数１０】

$$W = N\sum_{i=1}^{|K|} f_i w_i \qquad (3)$$

のように表現されることができ、ここで、w_iは、遷移テーブルメモリにおいて行ｉがま

たがるワードの数、f_iは、行ｉがアクセスされる相対頻度、Ｎは、入力データによって生成される同値類識別子の数である。

【００８４】

入力データについての事前知識は存在しないが、符号化遷移テーブルの構造を変更する能力が存在する。直接アドレス遷移テーブル内の行を並べ替えることによって、符号化遷移テーブル内の列の長さに影響を与えることができる。最適化問題は、メモリアクセスＷの総数を最小化する列配列を選択することである。遷移テーブル列アクセスパターンは一様分布$f_i = N / |K|$に従うと仮定する。この場合、

【数１１】

$$W = \frac{N^2}{|K|}\sum_{i=1}^{|K|} w_i \qquad (4)$$

となる。

【００８５】

これらの条件において、最適化問題は、量

【数１２】

$$v = \sum_{i=1}^{|K|} w_l = \sum_{i=1}^{|K|} \left\lceil \frac{count_i}{x} \right\rceil \qquad (5)$$

を最小化することである。$count_i$は、遷移カウントｉ、またはランレングス符号化遷移テーブルの行ｉ内のエントリの数であり、ｘは、ＴＴＭにおけるワード当たりのエントリの数である。

【００８６】

最適化問題を簡略化するため、ｘ＝１と仮定することができ、その結果、今最小化される必要がある量は、

【数１３】

$$v = \sum_{i=1}^{|K|} count_i \qquad (6)$$

となる。これは、任意のｘを有する関数を最小化するのと同様の結果を一般にもたらす。

【００８７】

図１９は、本明細書で提示された実行例に対する状態並べ替えの利点を示している。

【００８８】

状態並べ替えには多くの手法が存在する。１つの手法は、行内のエントリのソートされた順序に従って、直接アドレス遷移テーブルの行を整列することによって、符号化遷移テーブルの単一の列の長さを最小化することである。これは、その１つの列に対するランレングス符号化の効率を最大化する。しかし、並べ替えは、その他の列に対するランレングス符号化の効率を低下させることもある。

【００８９】

好ましい手法は、欲張りな手法であり、好ましくは、大多数の列についてランの長さを最大化し、それによって、各符号化列の長さを最小化することが望まれる。

【００９０】

２つの状態を所与として、相違しており、そのためランを継続しないＥＣＩの数を示す、相違行列（ｄｉｆｆｅｒｅｎｃｅ　ｍａｔｒｉｘ）を生成することによって開始することができる。このアルゴリズムが、図２０に示されている。

【００９１】

次に、相違行列内のエントリに基づいて、何らかの開始点から、状態を整列する。好ま

しくは、次のラベルを取得するのに大多数のランレングスを保つ状態を選択する。選択される開始状態は、好ましくは、相違行列において最大の列合計を有する状態である。その状態を最初に選んだ主意は、その他のすべてから最も異なるのがその状態だからである。その状態を（中間ではなく）末尾まで移動することによって、最長のランを保つことができる。このアルゴリズムの概要が、図２１に示されている。図２０および図２１のアルゴリズムは、一緒になって、「遷移テーブル状態並べ替え」アルゴリズムとして機能する。

【００９２】

最適化：メモリパッキング

インダイレクションレイヤは、符号化遷移テーブルの列がＴＴＭ内の任意の位置で開始および終了することを可能にすることを思い出されたい。符号化テーブル列の物理メモリへの愚直なパッキングは、各テーブル列について余計なメモリアクセスを招くことによって、上述の最適化を妨害することがあり得る。図１５において、入力シンボル「．」（ＥＣＩ　１）に関連付けられたランレングス符号化遷移テーブル列は、３つのエントリを含むが、ＴＴＭにおいて２つのメモリワードにまたがることに留意されたい。列を単一のメモリワードに保存することは可能であるが、この列へのアクセスは、図１５においてレイアウトされているように、２回のメモリアクセスを必要とする。符号化遷移テーブル行がたかだかｗ個のワードにまたがるに過ぎないことを保証することによって、インダイレクションレイヤによって提供される柔軟性を利用することができ、ここで、

【数１４】

$$\omega \leq \left\lceil \frac{count}{x} \right\rceil \qquad (7)$$

である。図２０は、符号化遷移テーブルをＴＴＭにパッキングするためにこの制約を使用した一例を示している。ＥＣＩ　１に関連付けられた列の取り出しは、ただ１回のメモリアクセスを必要とすることが分かる。この例では、メモリ効率性における損失はないが、常にそうであるとは限らない。例えば、メモリワードが３つのエントリを保有し、どの符号化遷移テーブル列も２つのエントリを含む場合を考えられたい。

【００９３】

このメモリパッキング問題は、ｗが制約関数または目的関数である、古典的な分数ナップザック問題（ｆｒａｃｔｉｏｎａｌ　ｋｎａｐｓａｃｋ　ｐｒｏｂｌｅｍ）の変形である。Ｂｌａｃｋ，Ｐ．Ｅ．、「Ｄｉｃｔｉｏｎａｒｙ　ｏｆ　Ａｌｇｏｒｉｔｈｍｓ　ａｎｄ　Ｄａｔａ　Ｓｔｒｕｃｔｕｒｅｓ」、ＮＩＳＴ、２００４年を参照されたく、その開示の全体は、参照により本明細書に組み込まれる。ここでの好ましい実施形態における相違は、符号化遷移テーブル列の連続記憶域を必要とすることである。これは、古典的問題において対象（符号化遷移テーブル列）を複数のナップザック（メモリワード）に区分するときに、追加の制約を課す。

【００９４】

この問題に対する１つの解決策は、部分集合和に基づいている。これは、一般的な場合におけるＮＰ完全問題であるが（ＧａｒｅｙおよびＪｏｈｎｓｏｎ、「Ｃｏｍｐｕｔｅｒｓ　ａｎｄ　Ｉｎｔｒａｃｔａｂｉｌｉｔｙ：Ａ　Ｇｕｉｄｅ　ｔｏ　ｔｈｅ　Ｔｈｅｏｒｙ　ｏｆ　ＮＰ－Ｃｏｍｐｌｅｔｅｎｅｓｓ」、Ｗ．Ｈ．Ｆｒｅｅｍａｎ　ａｎｄ　Ｃｏ．、１９７９年を参照されたく、その開示の全体は、参照により本明細書に組み込まれる）、それが多項式時間で実行されるある条件が存在し、すなわち、それは、和を生成するためにそこから要素が選択される要素の数よりも和がはるかに小さいという条件である。好ましい実施形態の和は、常にメモリワードの幅であり、そのため、好ましいアルゴリズムも、多項式時間で実行される。

【００９５】

基本的なアイデアは、最長のランレングス符号化列を見出し、それを最初に選択することである。その後、可能な限り最良のパッキングを達成することを保証しながら、それをメモリワード内にパッキングする。その後、直前の列で残されたエントリの数を取り、残

りのランレングス符号化列を用いてそれに部分集合和を適用することができる。これは、追加のメモリアクセスを引き起こすことなく、できるだけ満杯になるようメモリをパッキングする。このプロセスは、残りの符号化列がなくなるまで繰り返される。本明細書で「最適メモリワードパッキング」アルゴリズムと呼ばれる、このアルゴリズムの概要が、図２１に示されており、図において、Ｒは、ランレングス符号化列の集合であり、ｗは、メモリワードの幅である。

【００９６】

実装アーキテクチャ

このセクションでは、好ましい高スループットＤＦＡおよびパイプライン化遷移テーブル技法に基づいた、高性能正規表現検索システムの実施が説明される。この実施の主眼は、高速ディスクのアレイへの高帯域幅相互接続を有する複数のスーパースカラマイクロプロセッサおよび再構成可能なハードウェア装置を含む、ハイブリッド処理プラットフォームである。図２４は、システムレベルアーキテクチャの一例を示しており、ユーザインタフェースと、正規表現コンパイラと、ファイルＩ／Ｏコントローラと、正規表現ファームウェアモジュールと、結果プロセッサとを含む。

【００９７】

アーキテクチャの目的は、正規表現エンジンのアレイを再構成可能なハードウェア装置（例えばＦＰＧＡ）に埋め込むことによって、スループットを最大化することである。制御ロジックおよび文脈バッファをサポートするエンジンのアレイは、論理的に単一のファームウェアモジュールと見なされることができる。正規表現回路／エンジンがＦＰＧＡ上に実現される一実施形態では、これらのエンジンは、ハードウェア定義言語（ＨＤＬ）表現に統合され、知られた技法を使用してＦＰＧＡにロードされることができる。

【００９８】

各正規表現エンジンの主要なタスクは、高速ディスクからの入力ファイルストリーム内の正規表現を認識することである。正規表現の集合は、好ましくは、ユーザインタフェースを介してユーザによって指定され、高スループットＤＦＡにコンパイルされ、ソフトウェアコンポーネントの集まりによって１組のテーブルおよびレジスタ値に変換される。１組のテーブルおよびレジスタ値は、検索を開始する前に、ファームウェアモジュールに書き込まれる。正規表現エンジンは、パターンを認識した場合、文脈（パターンを含むファイルの部分）と、ファイル内のパターンの開始および終了インデックスと、アクセプト状態ラベルとを含むメッセージを、結果プロセッサに送る。動作環境および統合レベルに応じて、ユーザインタフェースは、簡単なコマンドラインインタフェース、グラフィカルユーザインタフェース（ＧＵＩ）、または特定言語向けＡＰＩとすることができる。以下のサブセクションは、残りのコンポーネントについての詳細な説明を提供する。

【００９９】

実装アーキテクチャ：正規表現コンパイラ

図５に詳しく示されるように、正規表現コンパイラは、ユーザによって指定された正規表現の集合をユーザインタフェースから受け取る。その後、当技術分野の標準アルゴリズムが、指定された正規表現を解析して、正規表現からＮＦＡを生成し、ＮＦＡを位置独立なＤＦＡに変換し、位置独立なＤＦＡを最小ＤＦＡに縮小する。正規表現コンパイラによって実行される次のステップは、従来の最小ＤＦＡを、本発明の好ましい高スループットＤＦＡに変換することである。このステップは、（遷移テーブルについての適切なマッチフラグおよびマッチリスタートフラグの決定と、入力シンボルのＥＣＩへの符号化とを含む）サイクル当たりｍ個の入力シンボルからなるストライドに対応するためのＤＦＡのスケーリングと、遷移テーブルのランレングス符号化とに関して上で説明されたプロセスを含む。次に、図２０および図２１のアルゴリズムが、遷移テーブルメモリへのランレングス符号化遷移の保存を最適化するために実行されることができ、図２３のアルゴリズムが、遷移テーブルメモリのワードへのランレングス符号化遷移エントリのパッキングを最適化するために実行されることができる。その後、インダイレクションテーブル内のエントリが決定されることができ、コンパイラは、入力シンボル対ＥＣＩテーブル、インダイレ

クションメモリテーブル、および遷移テーブルのための回路メモリにデータ投入するように動作する正規表現回路５０２に、コマンドを発行する準備が整う。

【０１００】

実装アーキテクチャ：結果プロセッサ

結果プロセッサを介して検索結果を報告するために、様々な技法のいずれもが使用され得ることが予想される。好ましい結果プロセッサは、正規表現回路（エンジン）によって生成された結果を使用して、各マッチに対する正しい表現および入力文字列セグメントを決定するように構成されることができる。図２４に示されるもののような好ましい一実施形態では、正規表現回路（エンジン）によって生成される結果は、一意のエンジン識別子（ＩＤ）と、開始状態と、アクセプト状態と、アクセプト遷移をトリガするｍ個の入力シンボルに対するＥＣＩとを含む。結果は、パターンにマッチした文字列を含む入力ストリームの一部分である、マッチについての文脈も含む。結果プロセッサは、マッチング文字列および関連する正規表現（パターン）をユーザインタフェースに報告する。

【０１０１】

実装アーキテクチャ：ファイルＩ／Ｏコントローラ

ファイルＩ／Ｏコントローラは、入力ストリームを制御する、システムのコンポーネントである。図２４の例示的なシステムでは、ファイルＩ／Ｏコントローラは、データストアから正規表現回路に流れる、ファイルのストリームを制御する。当技術分野で知られているように、入力ストリームも、ネットワークインタフェース（またはその他のデータインタフェース）によって供給され得ることに留意されたい。

【０１０２】

実装アーキテクチャ：正規表現ファームウェア

正規表現ファームウェアモジュールは、図２４に示されるシステムアーキテクチャにおける主要なデータパスコンポーネントである。正規表現ファームウェアモジュールは、好ましくは、正規表現エンジン（またはパターンマッチング回路）のアレイと、少量の制御ロジックとを含む。アレイ内のエンジンの数は、システム内の再構成可能ハードウェア装置の能力に依存する。制御ロジックは、入力ファイルストリームを各正規表現エンジンにブロードキャストし、したがって、エンジンは、同じ入力データに基づいて並列に動作する。制御ロジックはまた、入力のコピーを文脈バッファに送る。文脈バッファのサイズは、エンジンがパターンを認識したときに、結果プロセッサに送られる文脈の量に依存する。このパラメータは、ファームウェア／ソフトウェアインタフェースに過剰負荷をかけない間は、返され得る文脈の量を最大化するように調整されることができる。

【０１０３】

先に言及されたように、正規表現エンジンのスループットは、正規表現エンジンが決定性有限オートマトンについての状態遷移を計算することができる速度によって基本的に制限される。現在状態および入力シンボルに基づいた次状態の決定は、本来的には直列動作である。好ましい実施プラットフォーム上で利用可能な再構成可能ロジック資源を利用するために、並列性を最大化することが望まれる。パイプライン化は、直列計算において並列動作の数を増加させるための一般的な技法であるが、処理パイプラインにフィードバックループがないことが必要とされる。パイプラインの与えられたステージにおける動作の出力は、パイプライン内の後続ステージの結果に依存することができない。図２５に示されるように、正規表現エンジンは、一連のパイプライン化された計算ブロックである。ブロック間にフィードバックループがなく、各ブロックは、先行ブロックの結果にのみ基づいて動作することに留意されたい。これは、本発明の好ましい実施形態の著しい特徴である。アルファベット符号化は、１組の入力シンボルにのみ基づいて動作する状態独立な動作である。インダイレクションテーブル検索は、エントリを見つけるために、結果の入力シンボルＥＣＩを使用する。遷移テーブル検索は、インダイレクションテーブルエントリ内に含まれるポインタおよびインデックスにのみ依存する。現在状態に依存する唯一の動作は、状態選択ブロック内の最後の計算である。この唯一のフィードバックループをパイプラインの最終ステージに効果的に「押し込む」ことによって、好ましい実施形態は、並

列性を、ひいては正規表現エンジンのスループットを最大化する。以下のサブセクションは、正規表現エンジン内の各ブロックの設計について説明する。

【０１０４】

正規表現ファームウェア：アルファベット符号化

アルファベット符号化ブロックは、ｍ個の入力シンボルからなる組に同値類識別子（ＥＣＩ）を割り当てる。各入力シンボルがｉ個のビットを使用して指定され、ＥＣＩがｐ個のビットを使用して指定される場合、アルファベット符号化ブロックは、入力を基本的にｍｉビットからｐビットに短縮する。この動作を実行するための単純な方法は、直接アドレステーブルを使用して対組み合わせを実行することである。図２６に示されるように、テーブルの第１の組は、１つのｉビット入力シンボルをｊビットのＥＣＩに変換する。このステップは、単一の入力シンボルを同値類にマッピングする。テーブルの次のステージは、単一の入力シンボルに対応するｊビットＥＣＩを２つ単に連結し、ＥＣＩテーブルを直接アドレスすることによって、２つの入力シンボルに対してｋビットＥＣＩを生成する。第２のステージにおいてＥＣＩテーブルを実施するために使用されるメモリのアドレス可能性によって、ｊの上限が定められることに留意されたい。具体的には、２ｊは、メモリによってサポートされるアドレスビットの数より小さいか、または等しくなければならない。同様に、次のステージは、２つのシンボルに対応するｋビットＥＣＩを２つ連結し、ＥＣＩテーブルを直接アドレスすることによって、４つの入力シンボルに対してｐビットＥＣＩを生成する。同様に、ＥＣＩテーブルによってサポートされるアドレス空間が、ｋに対して上限を定める。理論的には、この技法は、任意の数の入力シンボルにＥＣＩを割り当てるために使用されることができるが、実際には、最終の同値類によってカバーされるシンボルの数が増加するにつれて、メモリ効率は著しく悪化する。

【０１０５】

図２５のパイプラインの後続ステージを実施するために必要とされるロジックは、すでに上で説明された。

【０１０６】

正規表現ファームウェア：バッファ

各正規表現エンジンは、好ましくは、小規模な入力バッファおよび出力バッファを含む。入力バッファは、単一のエンジンが、ＴＴＭ内の複数のワードにまたがる遷移テーブル列を取り出さなければならない場合に、アレイ内のすべてのエンジンを機能停止にさせることを防止する。いずれかのエンジンの入力バッファが満杯である場合、アレイ全体は機能停止しなければならないが、入力バッファは、ファイル入力速度の瞬間的変動を分離するのに役立つ。出力バッファは、正規表現エンジンがマッチを見出した後、そのマッチが結果プロセッサに送られる以前に、正規表現エンジンが処理を継続することを可能にする。文脈バッファは、好ましくは、ラウンドロビン方式で正規表現エンジンの出力バッファにサービスを行う。いずれかのエンジンの出力バッファが満杯である場合、別の結果を出力バッファに送る以前に、エンジンは機能停止しなければならない。エンジンの入力バッファが満杯である場合、好ましくは、アレイが機能停止しなければならない。

【０１０７】

本発明がその好ましい実施形態に関連して上で説明されたが、本発明の範囲内に依然として収まる様々な修正が、本発明に施されることができる。本発明に対するそのような修正は、本明細書の教示を検討することで認識可能であろう。例えば、遷移テーブルは、行が状態に対応し、列がＥＣＩに対応するものとして本明細書では説明されたが、本明細書で説明されたどのテーブルの行と列も反対にされ得ることは容易に理解されよう。そのため、本発明の完全な範囲は、添付の特許請求の範囲とその法的均等物によってのみ確定されるものとする。

【図面の簡単な説明】

【０１０８】

【図１】例示的な正規表現を示す図である。

【図２】図１の正規表現についてのＤＦＡを示す図である。

【図３】図１の正規表現についての改良ＤＦＡを示す図である。
【図４】図３のＤＦＡについての従来の遷移テーブルを示す図である。
【図５】本発明の好ましい一実施形態の概略ブロック図である。
【図６】ＤＦＡのストライドを２倍にするための好ましいアルゴリズムを示す図である。
【図７】状態の数が削減され、マッチおよびマッチリスタート用のフラグを遷移内に有する、本発明の好ましい実施形態によるＤＦＡを示す図である。
【図８】入力シンボルを同値類識別子（ＥＣＩ）に符号化するための好ましいアルゴリズムを示す図である。
【図９】図７のＤＦＡのための好ましい一遷移テーブルを示す図である。
【図１０】ＤＦＡのストライドがサイクル当たり２つの入力シンボルに等しい、図１の正規表現のための遷移テーブルを示す図である。
【図１１】ｄおよびｐから基本ＤＦＡ^{1}を構成するための好ましいアルゴリズムを示す図である。
【図１２】例示的なランレングス符号化遷移テーブルを示す図である。
【図１３】図１２のランレングス符号化遷移テーブルの隣接保存バージョンを示す図である。
【図１４】与えられたＥＣＩに対応する取り出された１組のランレングス符号化遷移に基づいて次状態を決定するための例示的な状態選択論理回路を示す図である。
【図１５】図１２および図１３のランレングス符号化遷移テーブルが保存される（ａ）インダイレクションテーブルおよび（ｂ）メモリを示す図である。
【図１６】図１５のランレングス符号化遷移テーブルが事前計算ランレングスプレフィックス合計を含む（ａ）インダイレクションテーブルおよび（ｂ）メモリを示す図である。
【図１７】図１４のランレングス符号化遷移テーブルが事前計算ランレングスプレフィックス合計を含む（ａ）インダイレクションテーブルおよび（ｂ）ＴＴＭの代替定式化を示す図である。
【図１８】例示的な状態選択論理回路を示す図である。
【図１９】（ａ）状態並べ替えによって最適化された例示的な遷移テーブルおよび（ｂ）状態並べ替え遷移テーブルのランレングス符号化バージョンを示す図である。
【図２０】ＴＴＭのためのランレングス符号化を最適化するのに使用される相違行列を計算するための好ましいアルゴリズムを示す図である。
【図２１】ＴＴＭにおけるランレングス符号化の最適化のための好ましいアルゴリズムを示す図である。
【図２２】ＴＴＭにパッキングされた符号化遷移テーブルの一例を示す図である。
【図２３】メモリワードの最適パッキングのための好ましいアルゴリズムを示す図である。
【図２４】好ましい正規表現システムアーキテクチャを示す図である。
【図２５】一連のパイプライン化された計算ブロックとしての好ましい実施形態の正規表現エンジンを示す図である。
【図２６】直接アドレステーブルおよび対組み合わせを使用してＥＣＩを入力シンボルに割り当てるための好ましいプロセスを示す図である。

【図１】

$$\$[0-9]+(\backslash.[0-9]\{2\})\{0,1\}$$

Figure 1

【図２】

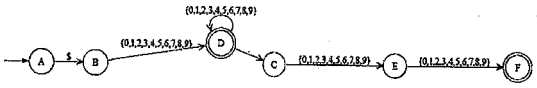

Figure 2

【図３】

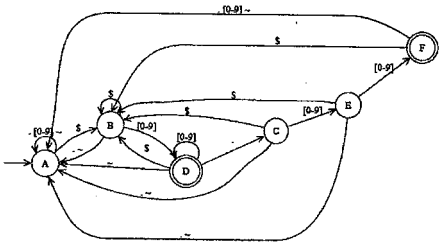

Figure 3

【図４】

状態	シンボル 0…9	.	$	~…
A	A…A	A	B	A…
B	D…D	A	B	A…
C	E…E	A	B	A…
D	D…D	C	B	A…
E	F…F	A	B	A…
F	A…A	A	B	A…

Figure 4

【図５】

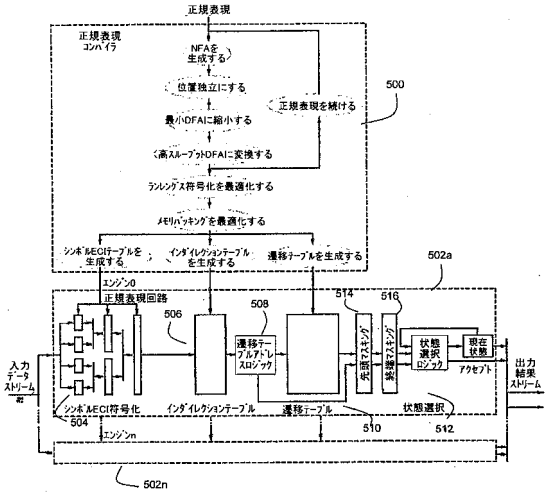

Figure 5

【図６】

function DOUBLESTRIDE(δ^k) : δ^{2k}
  foreach $q \in Q$ do
    foreach $a \in \Sigma^k$ do
      foreach $b \in \Sigma^k$ do
        $\delta^{2k}(q, ab) \leftarrow \delta^k(\delta^k(q, a), b)$
end

Figure 6

【図７】

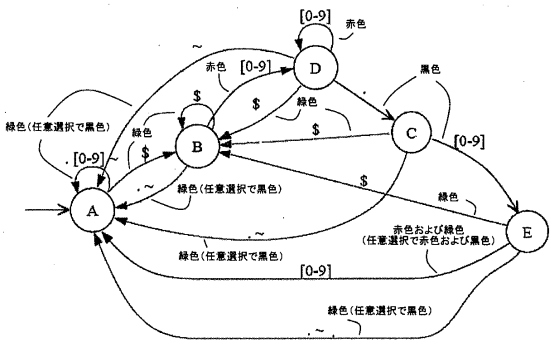

Figure 7

【図８】

function PARTITION($Q, \Sigma, \delta : Q \times \Sigma \mapsto Q$) : ($result, K, \kappa : \Sigma \mapsto K$)
  $WorkList \leftarrow \Sigma$
  $result \leftarrow \emptyset$
  $i \leftarrow -1$
  while $WorkList \neq \emptyset$ do
    $w \leftarrow$ WorkListに属する何らかの要素
    $equiv \leftarrow \{ a \in \Sigma \mid \forall q \in Q\ \delta(q, a) = \delta(q, w) \}$
    $i \leftarrow i + 1$
    $K \leftarrow K \cup \{i\}$
    foreach $sym \in equiv$ do $\kappa(sym) \leftarrow i$
    $result \leftarrow result \cup equiv$
    $WorkList \leftarrow WorkList - equiv$
end

Figure 8

【図９】

シンボル	ECI
$	0
.	1
0–9	2
その他すべて	3

(a)

状態	ECI 0	1	2	3
A	(B,1,0)	(A,1,0)	(A,1,0)	(A,1,0)
B	(B,1,0)	(A,1,0)	(D,0,1)	(A,1,0)
C	(B,1,0)	(A,1,0)	(E,0,0)	(A,1,0)
D	(B,1,0)	(C,0,0)	(D,0,1)	(A,1,0)
E	(B,1,0)	(A,1,0)	(A,0,1)	(A,1,0)

(b)

Figure 9

【図１０】

シンボル		ECI
0	0	0
0	1	1
0	2	2
0	3	1
1	0	0
1	1	1
1	2	3
1	3	1
2	0	4
2	1	5
2	2	6
2	3	7
3	0	0
3	1	1
3	2	1
3	3	1

(a)

状態	ECI 0	1	2	3	4	5	6	7
A	(B,1,0)	(A,1,0)	(D,1,1)	(A,1,0)	(B,1,0)	(A,1,0)	(A,1,0)	(A,1,0)
B	(B,1,0)	(A,1,0)	(D,1,1)	(A,1,0)	(B,1,1)	(C,0,1)	(D,0,1)	(A,1,1)
C	(B,1,0)	(A,1,0)	(D,1,1)	(A,1,0)	(B,1,0)	(A,1,0)	(A,0,1)	(A,1,0)
D	(B,1,0)	(A,1,0)	(D,1,1)	(E,0,0)	(B,1,1)	(C,0,1)	(D,0,1)	(A,1,1)
E	(B,1,0)	(A,1,0)	(D,1,1)	(A,1,0)	(B,1,1)	(A,1,1)	(A,1,1)	(A,1,1)

(b)

Figure 10

【図１１】

```
function BASEDFA(Q, Σ, q0, δ, A, ρ)) : DFA^1 = (Q', Σ', q0', K, κ, δ')
    K ← {0}
    foreach a ∈ Σ do κ(a) ← 0
    WorkList ← {q0}
    while WorkList ≠ ∅ do
        w ← WorkListに属する何らかの要素
        WorkList ← WorkList − {w}
        /*                          w 既にλ閉方された状態の集合
        /*                            q0 ∈w 構造による
        Q' ← Q' ∪ {w}
        foreach a ∈ Σ do
            targs ← {t | (∃s ∈ w)δ(s, a) = t}
            targs ← LAMBDACLOSURE(targs)
            red ← targs ∩ A ≠ ∅
            green ← (∀ s ∈ w − {q0})(∀ t ∈ Q − {q0}) δ(s, a) ≠ t
            δ' ← δ' ∪ {w × a ↦ (targs, green, red)}
            if targs ∉ Q'
            then WorkList ← WorkList ∪ {targs}
end
```

Figure 11

【図１２】

状態	ECI 0	1	2	3
A	5(B,1,0)	3(A,1,0)	1(A,1,0)	5(A,1,0)
B			1(D,0,1)	
C			1(E,0,0)	
D		1(C,0,0)	1(D,0,1)	
E		1(A,1,0)	1(A,0,1)	

Figure 12

【図１３】

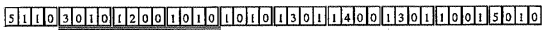

Figure 13

【図１４】

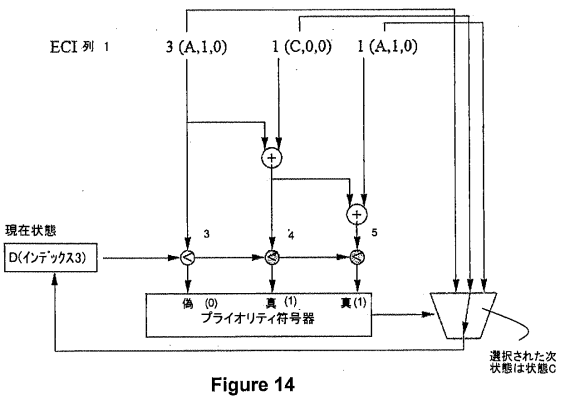

Figure 14

【図１５】

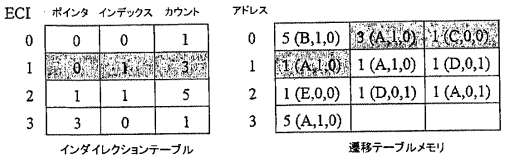

Figure 15

【図１６】

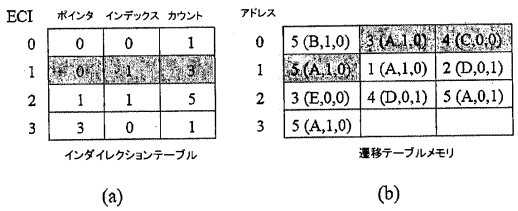

Figure 16

【図１７】

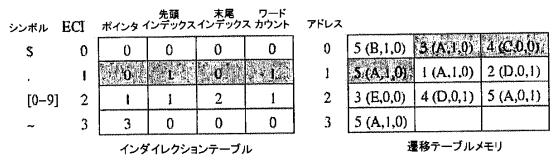

Figure 17

【図１８】

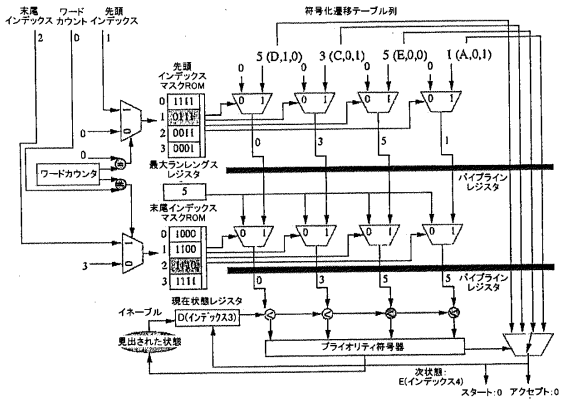

Figure 18

【図１９】

状態	ECI 0	1	2	3
A	(B,1,0)	(A,1,0)	(A,1,0)	(A,1,0)
C	(B,1,0)	(A,1,0)	(E,0,0)	(A,1,0)
E	(B,1,0)	(A,1,0)	(A,0,1)	(A,1,0)
B	(B,1,0)	(A,1,0)	(D,0,1)	(A,1,0)
D	(B,1,0)	(C,0,0)	(D,0,1)	(A,1,0)

(a)

状態	ECI 0	1	2	3
A	(5,B,1)0	(4,A,1)0	(1,A,1)0	(5,A,1)0
C			(1,E,0)0	
E			(1,A,0)1	
B			(2,D,0)1	
D		(1,C,0)0		

(b)

Figure 19

【図２０】

```
function DIFFMATRIX(Q, Σ, δ) : D
    foreach q ∈ Q do
        foreach p ∈ Q do
            foreach s ∈ Σ do  D[q][p] ← D[q][p] + δ(q, s) ≠ δ(p, s)
end
```

Figure 20

【図２１】

```
function RLOPTIM(Q, q0, Σ, δ, D) : DFA' = (Q', Σ, q0', δ')
    Used ← ∅
    cur ← MAX( Σ_{i∈Q} D[i])
    while Used ≠ Q do
        next ← MAX( D[curr] − Used)
        Used ← Used ∪ {next}
        cur ← next
end
```

Figure 21

【図２２】

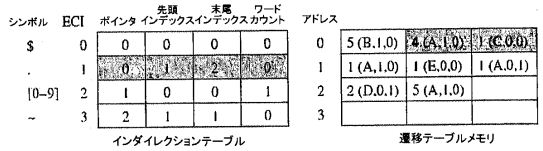

Figure 22

【図２３】

```
procedure COLPACK(R, x)
    while R ≠ ∅ do
        longest ← MAXLENGTH(R)
        R ← R − {longest}
        PACK(longest)
        sum ← x − (longest mod x)
        L0 ← {0}
        foreach col ∈ R do  Li ← MERGELISTS(Li−1, Li−1 + LENGTH(col))
        sumより大きいすべての要素をLi から取り除く
        S ← Max(Ln)
        foreach col ∈ S do
            R ← R − {col}
            call PACK(col)
end
```

Figure 23

【図２６】

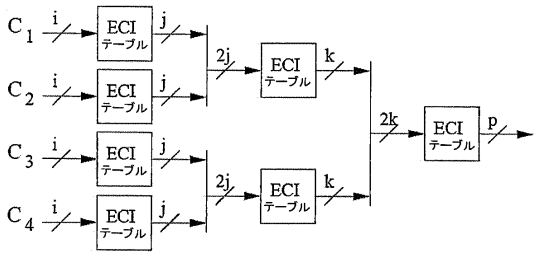

Figure 26

【図２４】

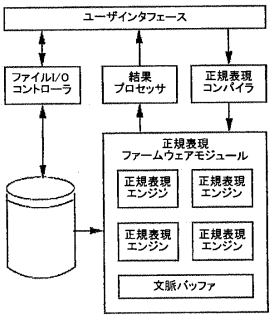

Figure 24

【図２５】

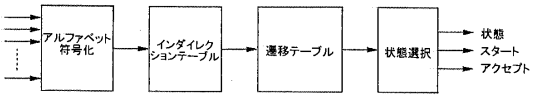

Figure 25

フロントページの続き

(72)発明者　サイトロン，ロン・ケイ
アメリカ合衆国、ミズーリ・６３１２４、セント・ルイス、グラナダ・ウエイ・４８
(72)発明者　テイラー，デイビツド・エドワード
アメリカ合衆国、ミズーリ・６３１１８、セント・ルイス、ミズーリ・アベニユー・３４４８
(72)発明者　ブロデイ，ベンジヤミン・カリー
アメリカ合衆国、ミズーリ・６３１３０、ユニバーシテイ・シテイ、リーランド・アベニユー・１
・ダブリユ・７２６

審査官　吉田　誠

(56)参考文献　特開２００５－２４２９９７（ＪＰ，Ａ）
特開平１０－２０７９１２（ＪＰ，Ａ）
特開２００５－２４２６７２（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１７／３０