(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7019589号
(P7019589)

(45)発行日　令和4年2月15日(2022.2.15)

(24)登録日　令和4年2月4日(2022.2.4)

(51)国際特許分類
Ｈ０４Ｌ　45/745 (2022.01)

ＦＩ
Ｈ０４Ｌ　45/745

請求項の数　23　（全65頁）

(21)出願番号　特願2018-549927(P2018-549927)
(86)(22)出願日　平成29年3月23日(2017.3.23)
(65)公表番号　特表2019-512973(P2019-512973A)
(43)公表日　令和1年5月16日(2019.5.16)
(86)国際出願番号　PCT/US2017/023906
(87)国際公開番号　WO2017/165708
(87)国際公開日　平成29年9月28日(2017.9.28)
審査請求日　令和2年3月19日(2020.3.19)
(31)優先権主張番号　62/312,106
(32)優先日　平成28年3月23日(2016.3.23)
(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　62/312,187
(32)優先日　平成28年3月23日(2016.3.23)
最終頁に続く

(73)特許権者　517384903
フォグホーン システムズ，インコーポレイテッド
Ｆｏｇｈｏｒｎ Ｓｙｓｔｅｍｓ，Ｉｎｃ．
アメリカ合衆国・カリフォルニア ９４０４０・マウンテン ヴュー スイート １００・ウエスト エル カミノ リアル ８００
８００ Ｗｅｓｔ Ｅｌ Ｃａｍｉｎｏ Ｒｅａｌ，Ｓｕｉｔｅ １００ Ｍｏｕｎｔａｉｎ Ｖｉｅｗ Ｃａｌｉｆｏｒｎｉａ ９４０４０，Ｕ．Ｓ．Ａ．
(74)代理人　110001195
特許業務法人深見特許事務所
最終頁に続く

(54)【発明の名称】　リアルタイムデータフロープログラミングのための効率的な状態機械

(57)【特許請求の範囲】
【請求項１】
システムであって、
コンピュータメモリにおいて第１のメモリ位置を含む入力キューを備え、前記入力キューは、処理されるべきトークンの先入れ先出しシーケンスを前記第１のメモリ位置に格納し、前記入力キューにおける各トークンはデータを含むかまたはデータを含まず、前記トークンが前記入力キューに入った時間を示すタイムスタンプを含み、データを含む各トークンは、解析されるべきデータおよびタイムスタンプを担持し、データを含まない各トークンは、経過イベントを表し、解析されるべきデータを担持せず、タイムスタンプを担持し、前記トークンはネットワークを介して前記入力キューによって受信され、前記システムはさらに、
前記入力キューに結合されるドライバコンポーネントを備え、前記ドライバコンポーネントは、バックトラッキングなしで前記入力キューにおける前記トークンを処理し、１つ以上の所定の入力パターンにマッチする、前記トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベントを生成し、前記システムはさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第２のメモリ位置を含む出力キューを備え、前記出力キューは、前記第２のメモリ位置に、前記ドライバコンポーネントによって生成された前記出力イベントの先入れ先出しシーケンスを格納し、前記システムはさらに、

前記ドライバコンポーネントに結合される状態表コンポーネントを備え、前記状態表コンポーネントは、前記所定の入力パターンを、行を含む状態表フォーマットで格納し、各行は前記状態表コンポーネントによって記述される複数の状態のうちのある状態を表し、前記システムはさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第３のメモリ位置を含む状態スタックコンポーネントを備え、前記状態スタックコンポーネントは前記第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納し、フレームは、変換状態番号、記号、およびデッドラインを含み、前記変換状態番号は前記状態表コンポーネントによって記述されるある状態に対応する行への参照を示す、システム。
【請求項２】
前記入力キューで受信される前記トークンは、物性をデジタル量に変換するハードウェアセンサによって生成される、請求項１に記載のシステム。
【請求項３】
前記ドライバコンポーネントの動作は、
前記状態スタックコンポーネントの一番上のフレーム上で参照される状態を検査することと、
前記入力キューにおける次の入力トークンを検査し、前記状態スタックコンポーネントに基づいて入力データのカテゴリを判断することと、
前記状態表コンポーネントによって示されるカテゴリのトークンのために前記状態表コンポーネントによって示されるアクションを実行することとを含む、請求項１または２に記載のシステム。
【請求項４】
前記入力キューは、前記入力キューによって受信された時間に基づく順序付けられたリストに前記トークンを格納し、前記入力キューにおける第１の方向は、最も早く受信されたトークンから最後に受信されたトークンまでの時間順序であり、
前記ドライバコンポーネントは、前記入力キューを前記第１の方向において検査することによって前記トークンを処理し、以前に検査されたトークンを検査しない、請求項１～３のいずれか１項に記載のシステム。
【請求項５】
前記入力キューは、前記入力キューによって受信された時間順序で前記トークンをリストに格納し、前記入力キューにおける第１の方向は、最も早く受信されたトークンから最後に受信されたトークンまでの時間順序であり、前記入力キューにおける第２の方向は、前記最後に受信されたトークンから前記最も早く受信されたトークンまでの時間順序であり、前記第２の方向は、前記第１の方向とは反対であり、
前記ドライバコンポーネントは、前記入力キューを前記第１の方向または前記第２の方向のいずれかで検査し、前記第１の方向および前記第２の方向の両方では検査しないことによって、前記トークンを処理する、請求項１～３のいずれか１項に記載のシステム。
【請求項６】
前記ドライバコンポーネントは前記入力キューの各トークンを１回だけ読出し、読出されたトークンをあとで再び読出すためにバッファに保持しない、請求項１～５のいずれか１項に記載のシステム。
【請求項７】
マッチングされる所定の入力パターンは、無限長のある数のトークンを有し得る、請求項１～６のいずれか１項に記載のシステム。
【請求項８】
前記状態表コンポーネントは状態機械を指定し、複数の行を含み、前記状態機械の変換状態を提供する各行は、
入力トークンカテゴリ値または入力ブレークカテゴリ値を、その値の入力トークンが受信されるととられるアクションにマッピングするアクションのセットと、
各種の記号を前記ドライバコンポーネントが変更すべき状態にマッピングする遷移のセッ

トと、
前記状態機械が特定の状態にとどまることが許される時間の間隔を示すタイムアウトとを含む、請求項１～７のいずれか１項に記載のシステム。
【請求項９】
前記変換状態番号は、前記状態表コンポーネントの行への参照であり、前記記号は、入力トークンもしくは他の記号から導出されたトークンもしくは中間記号、または組み合わせであり、前記デッドラインは、前記ドライバコンポーネントが次のフレームを前記状態スタックコンポーネントからポップするタイミングである、請求項１～８のいずれか１項に記載のシステム。
【請求項１０】
前記ドライバコンポーネントおよび状態表コンポーネントは、プログラマブルゲートアレイにおいて実施される、請求項１～９のいずれか１項に記載のシステム。
【請求項１１】
前記ドライバコンポーネントおよび状態表コンポーネントは、コンピュータを使用して実施される、請求項１～１０のいずれか１項に記載のシステム。
【請求項１２】
前記状態表コンポーネントは状態機械を指定し、前記ドライバコンポーネントは、ストリームデータを有するトークンが前記入力キューにおいて受信されない時間の経過に基づいて、第１の前記状態機械から第２の前記状態機械に変化する、請求項１～１１のいずれか１項に記載のシステム。
【請求項１３】
前記状態表コンポーネントは状態機械を指定し、前記ドライバコンポーネントは、ストリームデータを有するトークンのシーケンスが第１の期間中に前記入力キューにおいて受信されることに基づいて、第１の前記状態機械から第２の前記状態機械に変化し、
前記ドライバコンポーネントは、ストリームデータを有するトークンのシーケンスが第２の期間中に前記入力キューにおいて受信されないことに基づいて、第３の前記状態機械から第４の前記状態機械に変化する、請求項１～１２のいずれか１項に記載のシステム。
【請求項１４】
経過イベントが生じると、前記ドライバコンポーネントは現在の状態を期限切れとする、請求項１～１３のいずれか１項に記載のシステム。
【請求項１５】
方法であって、
コンピュータメモリにおいて第１のメモリ位置を含む入力キューを提供することを備え、
前記入力キューは、処理されるべきトークンの先入れ先出しシーケンスを前記第１のメモリ位置に格納し、前記入力キューにおける各トークンに関連付けられるタイムスタンプは、関連付けられるトークンが前記入力キューに入った時間を示し、前記トークンはネットワークを介して前記入力キューによって受信され、前記方法はさらに、
前記入力キューに結合されるドライバコンポーネントを提供することを備え、前記ドライバコンポーネントは、バックトラッキングなしで前記入力キューにおける前記トークンを処理し、１つ以上の所定の入力パターンにマッチする、前記トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベントを生成し、前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第２のメモリ位置を含む出力キューを提供することを備え、前記出力キューは、前記第２のメモリ位置に、前記ドライバコンポーネントによって生成された前記出力イベントの先入れ先出しシーケンスを格納し、前記方法はさらに、
前記ドライバコンポーネントに結合される状態表コンポーネントを提供することを備え、前記状態表コンポーネントは前記所定の入力パターンを状態表フォーマットで格納し、前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第３のメモリ

位置を含む状態スタックコンポーネントを提供することを備え、前記状態スタックコンポーネントは前記第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納し、フレームは、変換状態番号、記号、およびデッドラインを含み、前記変換状態番号は前記状態表コンポーネントによって記述されるある状態に対応し、前記方法はさらに、
前記状態表コンポーネントのために状態表を生成することを備え、前記状態表コンポーネントのために状態表を生成することは、
入力データのカテゴリを各々が表わす終端記号の集合を識別することと、
終端記号または非終端記号の少なくとも１つのパターンを各々が表わす非終端記号の集合を識別することと、
文法規則のセットを識別することと、
トップレベルの規則を識別することと、
属性のセットを識別することと、
各識別された終端記号および非終端記号について最初の集合を計算することと、
各識別された終端記号および非終端記号について後続集合を計算することと、
対応する後続集合の各要素を有する、前記トップレベルの規則のクロージャから開始状態を生成することとを含み、
前記状態表を生成することは、
前記開始状態から状態変化遷移が存在する各状態のクロージャを帰納的に生成することによって前記開始状態から到達可能なすべての状態を生成することと、
組み合わせおよびリダクションによって状態のセットを最適化することと、
各状態についてのアクション、遷移、およびタイムアウト値を生成することとを含み、
前記属性のセットは、
各属性が、α:T = {A0 → λ0,…An → λn}の形式をとり、αが属性の名前であり、Tが属性の型であり、Aが非終端記号であり、λがその記号の属性の値を計算するために使用される関数であることを含む、方法。
【請求項１６】
方法であって、
コンピュータメモリにおいて第１のメモリ位置を含む入力キューを提供することを備え、前記入力キューは、処理されるべきトークンの先入れ先出しシーケンスを前記第１のメモリ位置に格納し、前記入力キューにおける各トークンに関連付けられるタイムスタンプは、関連付けられるトークンが前記入力キューに入った時間を示し、前記トークンはネットワークを介して前記入力キューによって受信され、前記方法はさらに、
前記入力キューに結合されるドライバコンポーネントを提供することを備え、前記ドライバコンポーネントは、バックトラッキングなしで前記入力キューにおける前記トークンを処理し、１つ以上の所定の入力パターンにマッチする、前記トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベントを生成し、前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第２のメモリ位置を含む出力キューを提供することを備え、前記出力キューは、前記第２のメモリ位置に、前記ドライバコンポーネントによって生成された前記出力イベントの先入れ先出しシーケンスを格納し、前記方法はさらに、
前記ドライバコンポーネントに結合される状態表コンポーネントを提供することを備え、前記状態表コンポーネントは前記所定の入力パターンを状態表フォーマットで格納し、
前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第３のメモリ位置を含む状態スタックコンポーネントを提供することを備え、前記状態スタックコンポーネントは前記第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納し、フレームは、変換状態番号、記号、およびデッドラインを含み、前記変換状態番号は前記状態表コンポーネントによって記述されるある状態に対応し、前記方法はさら

に、
前記状態表コンポーネントのために状態表を生成することを備え、前記状態表コンポーネントのために状態表を生成することは、
入力データのカテゴリを各々が表わす終端記号の集合を識別することと、
終端記号または非終端記号の少なくとも１つのパターンを各々が表わす非終端記号の集合を識別することと、
文法規則のセットを識別することと、
トップレベルの規則を識別することと、
属性のセットを識別することと、
各識別された終端記号および非終端記号について最初の集合を計算することと、
各識別された終端記号および非終端記号について後続集合を計算することと、
対応する後続集合の各要素を有する、前記トップレベルの規則のクロージャから開始状態を生成することとを含み、
前記状態表を生成することは、
前記開始状態から状態変化遷移が存在する各状態のクロージャを帰納的に生成することによって前記開始状態から到達可能なすべての状態を生成することと、
組み合わせおよびリダクションによって状態のセットを最適化することと、
各状態についてのアクション、遷移、およびタイムアウト値を生成することとを含み、
前記文法規則のセットは、
各空でない規則がA → B0…Bnの形を取り、Aが非終端記号であり、B0からBnが各々終端記号または非終端記号のいずれかであることと、
各空の規則がタイムアウトτの形を取り、τがある有益な時間単位で特徴づけられる有限時間量であるか、または無制限な時間量を示す無限大であることとを含む、方法。
【請求項１７】
方法であって、
コンピュータメモリにおいて第１のメモリ位置を含む入力キューを提供することを備え、
前記入力キューは、処理されるべきトークンの先入れ先出しシーケンスを前記第１のメモリ位置に格納し、前記入力キューにおける各トークンに関連付けられるタイムスタンプは、関連付けられるトークンが前記入力キューに入った時間を示し、前記トークンはネットワークを介して前記入力キューによって受信され、前記方法はさらに、
前記入力キューに結合されるドライバコンポーネントを提供することを備え、前記ドライバコンポーネントは、バックトラッキングなしで前記入力キューにおける前記トークンを処理し、１つ以上の所定の入力パターンにマッチする、前記トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベントを生成し、前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第２のメモリ位置を含む出力キューを提供することを備え、前記出力キューは、前記第２のメモリ位置に、前記ドライバコンポーネントによって生成された前記出力イベントの先入れ先出しシーケンスを格納し、前記方法はさらに、
前記ドライバコンポーネントに結合される状態表コンポーネントを提供することを備え、
前記状態表コンポーネントは前記所定の入力パターンを状態表フォーマットで格納し、前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第３のメモリ位置を含む状態スタックコンポーネントを提供することを備え、前記状態スタックコンポーネントは前記第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納し、フレームは、変換状態番号、記号、およびデッドラインを含み、前記変換状態番号は前記状態表コンポーネントによって記述されるある状態に対応し、前記方法はさらに、
前記状態表コンポーネントのために状態表を生成することを備え、前記状態表コンポーネントのために状態表を生成することは、

入力データのカテゴリを各々が表わす終端記号の集合を識別することと、
終端記号または非終端記号の少なくとも１つのパターンを各々が表わす非終端記号の集合を識別することと、
文法規則のセットを識別することと、
トップレベルの規則を識別することと、
属性のセットを識別することと、
各識別された終端記号および非終端記号について最初の集合を計算することと、
各識別された終端記号および非終端記号について後続集合を計算することと、
対応する後続集合の各要素を有する、前記トップレベルの規則のクロージャから開始状態を生成することとを含み、
前記文法規則のセットは、
各空でない規則がA → B0…Bnの形を取り、Aが非終端記号であり、B0からBnが各々終端記号または非終端記号のいずれかであることと、
各空の規則がA → τの形を取り、τがある有益な時間単位で特徴づけられる有限時間量であるか、または無制限な時間量を示す無限大であることとを含む、方法。
【請求項１８】
方法であって、
コンピュータメモリにおいて第１のメモリ位置を含む入力キューを提供することを備え、
前記入力キューは、処理されるべきトークンの先入れ先出しシーケンスを前記第１のメモリ位置に格納し、前記入力キューにおける各トークンに関連付けられるタイムスタンプは、関連付けられるトークンが前記入力キューに入った時間を示し、前記トークンはネットワークを介して前記入力キューによって受信され、前記方法はさらに、
前記入力キューに結合されるドライバコンポーネントを提供することを備え、前記ドライバコンポーネントは、バックトラッキングなしで前記入力キューにおける前記トークンを処理し、１つ以上の所定の入力パターンにマッチする、前記トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベントを生成し、前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第２のメモリ位置を含む出力キューを提供することを備え、前記出力キューは、前記第２のメモリ位置に、前記ドライバコンポーネントによって生成された前記出力イベントの先入れ先出しシーケンスを格納し、前記方法はさらに、
前記ドライバコンポーネントに結合される状態表コンポーネントを提供することを備え、前記状態表コンポーネントは前記所定の入力パターンを状態表フォーマットで格納し、前記方法はさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第３のメモリ位置を含む状態スタックコンポーネントを提供することを備え、前記状態スタックコンポーネントは前記第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納し、フレームは、変換状態番号、記号、およびデッドラインを含み、前記変換状態番号は前記状態表コンポーネントによって記述されるある状態に対応し、前記方法はさらに、
前記状態表コンポーネントのために状態表を生成することを備え、前記状態表コンポーネントのために状態表を生成することは、
入力データのカテゴリを各々が表わす終端記号の集合を識別することと、
終端記号または非終端記号の少なくとも１つのパターンを各々が表わす非終端記号の集合を識別することと、
文法規則のセットを識別することと、
トップレベルの規則を識別することと、
属性のセットを識別することと、
各識別された終端記号および非終端記号について最初の集合を計算することと、
各識別された終端記号および非終端記号について後続集合を計算することと、

対応する後続集合の各要素を有する、前記トップレベルの規則のクロージャから開始状態を生成することとを含み、
前記属性のセットは、
各属性が、α:T = {A0 → λ0,…An → λn}の形式をとり、αが属性の名前であり、Tが属性の型であり、Aが非終端記号であり、λがその記号の属性の値を計算するために使用される関数であることを含む、方法。
【請求項１９】
前記トップレベルの規則は、Ω→Bの形を有し、Bは終端記号または非終端記号のいずれかである、請求項１５に記載の方法。
【請求項２０】
前記属性のセットは、
各属性が、α:T = {A0 → λ0,…An → λn}の形式をとり、αが属性の名前であり、Tが属性の型であり、Aが非終端記号であり、λがその記号の属性の値を計算するために使用される関数であることを含む、請求項１６または１７に記載の方法。
【請求項２１】
システムであって、
コンピュータメモリにおいて第１のメモリ位置を含む入力キューを備え、前記入力キューは、処理されるべきトークンの先入れ先出しシーケンスを前記第１のメモリ位置に格納し、前記入力キューにおける各トークンはデータを含むかまたはデータを含まず、前記トークンが前記入力キューに入った時間を示すタイムスタンプを含み、データを含む各トークンは、解析されるべきデータおよびタイムスタンプを担持し、データを含まない各トークンは、経過イベントを表し、解析されるべきデータを担持せず、タイムスタンプを担持し、前記トークンはネットワークを介して前記入力キューによって受信され、前記システムはさらに、
前記入力キューに結合されるドライバコンポーネントを備え、前記ドライバコンポーネントは、前記入力キューにおける前記トークンを処理し、１つ以上の所定の入力パターンにマッチする、前記トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベントを生成し、前記システムはさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第２のメモリ位置を含む出力キューを備え、前記出力キューは、前記第２のメモリ位置に、前記ドライバコンポーネントによって生成された前記出力イベントの先入れ先出しシーケンスを格納し、前記システムはさらに、
前記ドライバコンポーネントに結合される状態表コンポーネントを備え、前記状態表コンポーネントは、前記所定の入力パターンを、行を含む状態表フォーマットで格納し、各行は前記状態表コンポーネントによって記述される複数の状態のうちのある状態を表し、前記システムはさらに、
前記ドライバコンポーネントに結合され、前記コンピュータメモリにおいて第３のメモリ位置を含む状態スタックコンポーネントを備え、前記状態スタックコンポーネントは前記第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納し、フレームは、変換状態番号、記号、およびデッドラインを含み、前記変換状態番号は前記状態表コンポーネントによって記述されるある状態に対応する、システム。
【請求項２２】
前記状態表コンポーネントは状態機械を指定し、前記ドライバコンポーネントは、ストリームデータを有するトークンのシーケンスが第１の期間中に前記入力キューにおいて受信されることに基づいて、前記状態機械における第１の状態から前記状態機械の第２の状に変化する、請求項２１に記載のシステム。
【請求項２３】
前記状態表コンポーネントは状態機械を指定し、前記ドライバコンポーネントは、ストリームデータを有するトークンのシーケンスが第１の期間中に前記入力キューにおいて受信されないことに基づいて、前記状態機械における第１の状態から前記状態機械の第２の状

態に変化する、請求項２１に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
説明
関連出願との相互参照
この特許出願は、２０１６年３月２３日に出願された米国特許出願第６２／３１２，１０６号、第６２／３１２，１８７号、第６２／３１２，２２３号、および第６２／３１２，２５５号の利益を主張し、それらは、本願で引用されたすべての他の引例とともに、引用により援用される。
【背景技術】
【０００２】
発明の背景
この発明はコンピューティングの分野に関し、より具体的には、産業用機械によって生成される大量のデータを取扱うためのエッジコンピューティングにおいて用いられてもよいリアルタイムデータフロープログラミングのための効率的な状態機械に関する。
【０００３】
従来の企業ソフトウェアアプリケーションホスティングは、規模の経済性およびシステム効率を活用するために、データセンターまたは「クラウド」インフラストラクチャに依拠してきた。しかしながら、これらのデータセンターは、企業がそのビジネス活動のほとんどを行なう物理的活動地点（たとえば工場、倉庫、小売店など）から任意に離れている場合がある。産業用インターネット・オブ・シングス（industrial Internet of things：ＩＩｏＴ）とは、非常に高頻度でイベントを追跡するセンサを用いた物理的活動の計測に依拠するデバイスまたは使用事例の集合を指す。
【０００４】
多くのセクターにおける産業用機械が、製造、石油およびガス、採鉱、運輸、電力および水、再生可能エネルギー、ヘルスケア、小売、スマートビルディング、スマートシティ、および接続された車両を含むこのインターネット・オブ・シングス（ＩｏＴ）の下に入る。クラウドコンピューティングの成功にもかかわらず、多くの欠点が存在する。そのデータのすべてをクラウドストレージへ送信することは実用的ではない。なぜなら、接続性が常にそこにあるとは限らず、帯域幅が十分ではなく、レイテンシにおける変動が高すぎ、または、帯域幅が存在していたとしても、それは法外なコストがかかるためである。接続性、帯域幅、およびコストが問題ではなかったとしても、リアルタイムの意思決定および予測保守はなく、それは機械への著しい損傷をもたらし得る。
【０００５】
したがって、改良されたエッジ解析を含む改良されたコンピューティングシステム、アーキテクチャ、および手法、ならびにリアルタイムデータフロープログラミングのためのより効率的な状態機械が、産業用機械によって生成される大量のデータを取扱うために必要とされる。
【発明の概要】
【課題を解決するための手段】
【０００６】
発明の簡単な概要
効率的な状態機械に基づくパターンマッチング技術は、入力キューにおけるトークンを処理し、バックトラッキングなしに、１つ以上の所定の入力パターンにマッチする、トークンのシーケンスにおけるパターンを識別する。トークンには、データを含むこと、またはデータを含まないこと、および時間コンポーネントを含むことが可能である。トークンは、物性をデジタル量に変換するセンサによって生成されたデータのストリームであることができる。パターンマッチング技術は、入力キューを単一の方向において処理し、以前に検査されたトークンを検査しない。ある実現例では、マッチングされるべき特定のパター

ンを、状態機械を使用して指定し、状態機械は状態表において指定され、状態スタックを使用して動作する。

【０００７】

具体的な実現例では、パターンマッチング技術は、エッジコンピューティングシステムにおいて用いられ、エッジでのインテリジェンスを可能にするための方法である。特徴は、以下を含む。ゲートウェイデバイスまたは組込まれたシステム上でホストされるソフトウェア層においてセンサデータによってトリガすること。ソフトウェア層はローカルエリアネットワークに接続される。サービス、アプリケーション、およびデータ処理エンジンのリポジトリが、ソフトウェア層によってアクセス可能にされる。ソフトウェア層によって利用可能にされた式言語を通して、センサデータを特定条件の発生の意味論的記述と整合させること。式を連続的に実行することによってパターンイベントを自動的に発見すること。アプリケーションおよび解析式をつなぐために、ソフトウェア層によって管理されるネットワーク全体のゲートウェイデバイスおよび組込まれたシステムを越えて、サービスおよびアプリケーションをインテリジェントに構成すること。リソース利用可能性に基づいてアプリケーションおよび解析のレイアウトを最適化すること。ソフトウェア層の健全性を監視すること。未加工のセンサデータまたは式の結果を、ローカル時系列データベースまたはクラウドストレージに格納すること。サービスおよびコンポーネントは、どのゲートウェイ環境でも円滑な動作を保証するために、コンテナ化され得る。

【０００８】

エッジインテリジェンスは、インターネット・オブ・シングス（ＩｏＴ）データのソースでイネーブルにされる。システムは、リアルタイムのエッジ解析およびアプリケーションのために、ＩｏＴデバイスセンサデータへの強化（enrich）されたアクセス（ストリームモードまたはバッチモード、または双方）を提供する。システムは、低メモリフットプリント機械で動作する高性能解析エンジンを通して解析関数および解析式を実行するための高効率で表現的なコンピュータ言語を含む。システムは、さらなる機械学習のために収集データをクラウドへ発行することを可能にする。システムは、エッジアプリを開発するためのソフトウェア開発キットを含む。クラウドベースの管理コンソールが、エッジデプロイメント、構成、アプリケーション、および解析式の管理を可能にする。

【０００９】

エッジインフラストラクチャおよびプラットフォームの特定の一実現化例は、フォグホーン・システムズ・インコーポレイテッド（FogHorn Systems, Inc）（フォグホーン）による。フォグホーンのウェブサイト、www.foghorn-systems.com、出版物（白書、ユーザガイド、指導書、ビデオなどを含む）、ならびに、フォグホーンの技術および製品についての他の出版物が、引用により援用される。

【００１０】

フォグホーンは、産業用および商業用インターネット・オブ・シングス（ＩｏＴ）データのためのエッジインテリジェンスを可能にするためのプラットフォームを提供する。何百億もの産業用および商業用ＩｏＴデバイスによって生成されるデータの量は、インターネット全体を圧倒するのに十分莫大になるであろう。フォグホーンのプラットフォームはＩｏＴデータを、それがまさしく生じるところ、すなわちネットワークのエッジで処理し、解析し、それに応答する。フォグホーンの「インテリジェントエッジ」ソフトウェアプラットホームは、前例がないレベルの自動化、業務効率、コスト削減などを可能にする。

【００１１】

産業用インターネット・オブ・シングス（ＩＩｏＴ）は、センサ、機械類、およびコンピュータといった、相互接続された産業用および商業用デバイスからなる。ＩＩｏＴの目標は、分散された企業全体にわたって、より優れたデバイス制御、データ管理、機械自動化、および業務効率を可能にすることである。会社は、システム全体の管理および最適化のためにクラウドコンピューティングを活用しながら、リアルタイムの解析および自動化応答を使用してグリーンフィールドＩＩｏＴ機会を捕らえるためにエッジでフォグコンピューティングを適用することができる。フォグホーンのエッジコンピューティングプラット

フォームはまた、追加のコンピューティングリソースを追加することが実行できない場合に、既存のプログラマブルロジックコントローラ（programmable logic controller：ＰＬＣ）（たとえば、ブラウンフィールド機会）において動作するように設計される。ブラウンフィールドとは、確立されたシステムを勘案しながら情報技術（information technology：ＩＴ）問題領域を解決するための新しいシステムの実現を指す。新しいソフトウェアアーキテクチャは、既存のおよび動作中のソフトウェアを考慮する。

【００１２】

エッジインテリジェンスプラットフォームとは、ＩＩｏＴデバイスが存在するエッジのより近くまでデータ処理および解析を拡張する、フォグコンピューティング概念に基づいたソフトウェアベースのソリューションである。遠くの集中型クラウドへすべてのデータを送信する代わりに、エッジデバイスとの近接性を維持することは、待ち時間を最小化し、最大性能、より速い応答時間、ならびにより効果的な保守および業務戦略を可能にする。それはまた、全体的な帯域幅要件と、広範に分散されたネットワークを管理するコストとを著しく減少させる。

【００１３】

エッジでのＩＩｏＴ操作に焦点を合わせることは、全体的な帯域幅要件を減少させ、時間依存の条件への即時の自動化応答を可能にする。産業界は何十億もの新しいＩＩｏＴデバイスを追加しており、これらのデバイスは集団で、毎日多くのペタバイトのデータを生成する。このデータのすべてをクラウドに送信することは、法外なコストがかかるだけでなく、より大きいセキュリティリスクも生み出す。エッジで動作することは、はるかにより速い応答時間、リスクの減少、および全体的コストの低下を保証する。

【００１４】

２０１５年８月２７日に出願された米国特許出願第６２／２１０，９８１号、および２０１６年８月２９日に出願された米国特許出願第１５／２５０，７２０号は、ここに引用により援用され、エッジコンピューティング環境およびプラットフォームを記載する。２０１７年３月２３日に出願された米国特許出願第１５／４６７，３１３号は、ここに引用により援用され、リアルタイムデータフロープログラミングにおけるパターン駆動型反応の構成を記載する。２０１７年３月２３日に出願された米国特許出願第１５／４６７，３１８号は、ここに引用により援用され、リアルタイムデータフロープログラミング言語のためのツールおよび方法を記載する。

【００１５】

一実装形態では、方法は、データストリームをセンサ（たとえば物理量を監視し、監視された物理量をデジタル形式のデータストリームに変換するハードウェアデバイス）から受信することと、データストリームを入力キュー（たとえばＦＩＦＯメモリ）に格納することとを備え、各トークンは、データが受信されるときのタイムスタンプとともに格納されるデータストリームのデータを含み（またはデータを含まず）、前記方法はさらに、単一の方向に入力キューを通ってトークンを読出し、以前に読出されたトークンを再読出ししないことによって、１つ以上の所定のパターンにマッチする、データストリームにおけるパターンを識別することと、データストリーム内において所定のパターンを識別すると、所定のパターンが識別されたという肯定的な表示を出力キューにおいて出力することとを備える。

【００１６】

別の実装形態では、システムは以下を備える：入力キューは、ＦＩＦＯメモリとして編成されるコンピュータメモリにおいてメモリ位置を含み、 入力キューは処理されるべきトークンを格納し、各トークンは、データを含み（またはデータを含まず）、トークンが入力キューに入ったときのタイムスタンプを含む。ドライバコンポーネントは、バックトラッキングなしで入力キューにおけるトークンを処理し、１つ以上の所定の入力パターンにマッチする、トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベント出力を生成する。出力キューは、ＦＩＦＯメモリとして編成されるコンピュータメモリにおいて第２のメモリ位置を含み、出力キューは、ドライ

バコンポーネントによって生成された出力イベントのシーケンスを格納する。状態表コンポーネントは、所定の入力パターンを状態表フォーマットで格納する。状態スタックコンポーネントは、ＬＩＦＯメモリとして編成されるコンピュータメモリにおいて第３のメモリ位置を含み、スタックフレームには、変換状態番号、記号、もしくはデッドライン、または任意の組み合わせが含まれる。

【００１７】

別の実装形態では、方法は以下を備える：ＦＩＦＯメモリとして編成されるコンピュータメモリにおいて第１のメモリ位置を含む入力キューを提供することを備え、入力キューは処理されるべきトークンを格納し、各トークンは、データを含み（またはデータを含まず）、トークンが入力キューに入ったときのタイムスタンプを含む。ドライバコンポーネントを提供することを備え、ドライバコンポーネントは、バックトラッキングなしで入力キューにおけるトークンを処理し、１つ以上の所定の入力パターンにマッチする、トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベント出力を生成する。ＦＩＦＯメモリとして編成されるコンピュータメモリにおいて第２のメモリ位置を含む出力キューを提供することを備え、出力キューは、ドライバコンポーネントによって生成された出力イベントを格納する。所定の入力パターンを状態表フォーマットで格納する状態表コンポーネントを提供することを備える。スタックフレームを格納する状態スタックコンポーネント（たとえばＬＩＦＯメモリ）を提供することを備える。

【００１８】

本発明の他の目的、特徴、および利点は、以下の詳細な説明および添付図面を考慮すれば明らかになるであろう。図面では、図全体を通し、同じ参照記号は同じ特徴を表わす。

【図面の簡単な説明】

【００１９】

【図１】クライアント－サーバシステムおよびネットワークのブロック図である。

【図２】クライアントまたはサーバのより詳細な図である。

【図３】コンピュータシステムのシステムブロック図である。

【図４】センサストリームとクラウドとの間にあるエッジコンピューティングプラットフォームのブロック図である。

【図５】エッジ解析を含むエッジコンピューティングプラットフォームのより詳細なブロック図である。

【図６】エッジインフラストラクチャとクラウドインフラストラクチャとの間の動作フローを示す図である。

【図７】決定性有限オートマトン（ＤＦＡ）に変換された拡張非決定性有限オートマトン（ＮＦＡ）および状態縮小機械（state-reduced machine）を示す。

【図８】トークンアルファの受信で状態ＡからＢへの遷移を示す。

【図９】別途の状態遷移、状態Ｘを経た、状態Ａから状態Ｂへの遷移を示す。

【図１０】構文解析によって形成された抽象構文木の例を示す。

【図１１】離接のための部分グラフを示す。

【図１２】合接のための部分グラフを示す。

【図１３】構造を有する閉包（closure）を示す。

【図１４】時限順方向右端（timed forward right most）（１）またはＴＦＲ（１）パターンマッチングと呼ばれる技術を使用するパターンマッチャーの構成要素を示す。

【図１５Ａ】「ペット」の例の状態機械図を示す。

【図１５Ｂ】「ペット」の例の状態機械図を示す。

【発明を実施するための形態】

【００２０】

発明の詳細な説明

図１は、本発明の一実施形態を取入れる分散型コンピュータネットワーク１００の簡略ブロック図である。コンピュータネットワーク１００は、複数の通信リンク１２８を介して

通信ネットワーク１２４に結合された、複数のクライアントシステム１１３、１１６、および１１９と、サーバシステム１２２とを含む。通信ネットワーク１２４は、分散型ネットワーク１００のさまざまなコンポーネントが互いに通信して情報を交換することを可能にするためのメカニズムを提供する。

【００２１】

通信ネットワーク１２４はそれ自体、多くの相互接続されたコンピュータシステムおよび通信リンクで構成されてもよい。通信リンク１２８は、配線リンク、光リンク、衛星または他の無線通信リンク、波動伝搬リンク、もしくは、情報の通信のための他のメカニズムであってもよい。通信リンク１２８は、ＤＳＬ、ケーブル、イーサネット（登録商標）または他の配線リンク、受動または能動光リンク、３Ｇ、３．５Ｇ、４Ｇおよび他のモビリティ、衛星または他の無線通信リンク、波動伝搬リンク、もしくは、情報の通信のための任意の他のメカニズムであってもよい。

【００２２】

図１に示すさまざまなシステム間の通信を容易にするために、さまざまな通信プロトコルが使用されてもよい。これらの通信プロトコルは、ＶＬＡＮ、ＭＰＬＳ、ＴＣＰ／ＩＰ、トンネリング、ＨＴＴＰプロトコル、無線アプリケーションプロトコル（wireless application protocol：ＷＡＰ）、ベンダー固有プロトコル、カスタマイズされたプロトコルなどを含んでいてもよい。一実施形態では、通信ネットワーク１２４はインターネットであり、他の実施形態では、通信ネットワーク１２４は、ローカルエリアネットワーク（local area network：ＬＡＮ）、ワイドエリアネットワーク（wide area network：ＷＡＮ）、無線ネットワーク、イントラネット、私的ネットワーク、公的ネットワーク、スイッチドネットワーク、およびこれらの組合せなどを含む、任意の好適な通信ネットワークであってもよい。

【００２３】

図１の分散型コンピュータネットワーク１００は、本発明を取入れる実施形態の単なる例示であり、請求項に記載されるようなこの発明の範囲を限定しない。当業者であれば、他の変形、変更、および代替物を認識するであろう。たとえば、２つ以上のサーバシステム１２２が通信ネットワーク１２４に接続されてもよい。別の例として、複数のクライアントシステム１１３、１１６、および１１９は、アクセスプロバイダ（図示せず）を介して、または何らかの他のサーバシステムを介して、通信ネットワーク１２４に結合されてもよい。

【００２４】

クライアントシステム１１３、１１６、および１１９は典型的には、情報を提供するサーバシステムから情報を要求する。この理由により、サーバシステムは典型的には、クライアントシステムよりも多いコンピューティングおよびストレージ容量を有する。しかしながら、特定のコンピュータシステムは、そのコンピュータシステムが情報を要求しているか、または提供しているかに依存して、クライアントまたはサーバとして、双方として機能してもよい。加えて、この発明の局面はクライアント－サーバ環境を使用して説明されてきたが、この発明はスタンドアロンコンピュータシステムにおいても具現化され得ることが明らかであるはずである。

【００２５】

サーバ１２２は、クライアントシステム１１３、１１６、および１１９から情報要求を受信し、要求を満たすために必要とされる処理を行ない、要求に対応する結果を要求元クライアントシステムへ送り返す役割を担う。要求を満たすために必要とされる処理は、サーバシステム１２２によって行なわれてもよく、またはそれに代えて、通信ネットワーク１２４に接続された他のサーバに委任されてもよい。

【００２６】

クライアントシステム１１３、１１６、および１１９は、ユーザが、サーバシステム１２２によって格納された情報にアクセスし、情報をクエリすることを可能にする。特定の一実施形態では、クライアントシステムは、デスクトップアプリケーション、もしくはモバ

イルスマートフォンまたはタブレットアプリケーションなどのスタンドアロンアプリケーションとして動作可能である。別の実施形態では、クライアントシステム上で実行される「ウェブブラウザ」アプリケーションが、ユーザが、サーバシステム１２２によって格納された情報を選択し、情報にアクセスし、情報を検索し、またはクエリすることを可能にする。ウェブブラウザの例は、マイクロソフト社（Microsoft Corporation）によって提供されるインターネットエクスプローラ（Internet Explorer）ブラウザプログラム、モジラ（Mozilla）によって提供されるファイアーフォックス（Firefox）（登録商標）ブラウザ、グーグル（Google（登録商標））によって提供されるクローム（Chrome）ブラウザ、アップル（Apple）によって提供されるサファリ（Safari）ブラウザなどを含む。

【００２７】

クライアント－サーバ環境では、いくつかのリソース（たとえばファイル、音楽、ビデオ、またはデータ）はクライアントで格納され、一方、他のリソースは、サーバなどネットワークにおけるどこか他のところに格納されるかまたはそこから配信され、ネットワーク（たとえばインターネット）を介してアクセス可能である。したがって、ユーザのデータは、ネットワークまたは「クラウド」に格納され得る。たとえば、ユーザは、クラウド（たとえばサーバ）上にリモートに格納されている、クライアントデバイス上の文書に対して作業することができる。クライアントデバイス上のデータは、クラウドと同期され得る。

【００２８】

図２は、本発明の例示的なクライアントまたはサーバシステムを示す。一実施形態では、ユーザは、図２に示すようなコンピュータワークステーションシステムを通して、システムとインターフェイス接続する。図２は、モニタ２０３と、スクリーン２０５と、筐体２０７（システムユニット、キャビネット、またはケースとも呼ばれてもよい）と、キーボードまたは他の人間入力デバイス２０９と、マウスまたは他のポインティングデバイス２１１とを含む、コンピュータシステム２０１を示す。マウス２１１は、マウスボタン２１３などの１つ以上のボタンを有していてもよい。

【００２９】

本発明は、特定のフォームファクタ（たとえば、デスクトップコンピュータのフォームファクタ）のどのコンピューティングデバイスにも限定されず、さまざまなフォームファクタのあらゆるタイプのコンピューティングデバイスを含み得る、ということが理解されるべきである。ユーザは、スマートフォン、パーソナルコンピュータ、ラップトップ、電子タブレットデバイス、全地球測位システム（global positioning system：ＧＰＳ）受信機、ポータブルメディアプレイヤー、携帯情報端末（personal digital assistant：ＰＤＡ）、他のネットワークアクセスデバイス、および、データを送受信できる他の処理デバイスを含む、任意のコンピューティングデバイスとインターフェイス接続することができる。

【００３０】

たとえば、特定の一実現化例では、クライアントデバイスは、アップルｉＰｈｏｎｅ（登録商標）（たとえば、アップルｉＰｈｏｎｅ６）、アップルｉＰａｄ（登録商標）（たとえば、アップルｉＰａｄ、またはアップルｉＰａｄ　ｍｉｎｉ）、アップルｉＰｏｄ（登録商標）（たとえば、アップルｉＰｏｄ　Ｔｏｕｃｈ）、サムスン（Samsung）ギャラクシー（Galaxy）製品（たとえば、ギャラクシーＳシリーズ製品、またはギャラクシーノート（Galaxy Note）シリーズ製品）、グーグル（登録商標）ネクサス（Nexus）デバイス（たとえば、グーグルネクサス６、グーグルネクサス７、またはグーグルネクサス９）、およびマイクロソフトデバイス（たとえば、マイクロソフトサーフェス（Surface）タブレット）といった、スマートフォンまたはタブレットデバイスであり得る。典型的には、スマートフォンは、電話部分（および関連付けられた無線機）とコンピュータ部分とを含み、それらはタッチスクリーンディスプレイを介してアクセス可能である。

【００３１】

電話部分のデータ（たとえば、連絡先および電話番号）と、コンピュータ部分のデータ（たとえば、ブラウザを含むアプリケーションプログラム、画像、ゲーム、ビデオ、および

音楽）とを格納するための不揮発性メモリがある。スマートフォンは典型的には、写真およびビデオを撮るためのカメラ（たとえば、前向きのカメラまたは後部カメラ、もしくは両方）を含む。たとえば、スマートフォンまたはタブレットは、１つ以上の他のデバイスにストリーミングされ得るライブビデオを撮るために使用され得る。

【００３２】

筐体２０７は、プロセッサ、メモリ、大容量ストレージデバイス２１７などといった、よく知られたコンピュータコンポーネント（それらのうちのいくつかは図示されず）を収容する。大容量ストレージデバイス２１７は、大容量ディスクドライブ、フロッピー（登録商標）ディスク、磁気ディスク、光ディスク、光磁気ディスク、固定ディスク、ハードディスク、ＣＤ－ＲＯＭ、書込可能ＣＤ、ＤＶＤ、書込可能ＤＶＤ（たとえば、ＤＶＤ－Ｒ、ＤＶＤ＋Ｒ、ＤＶＤ－ＲＷ、ＤＶＤ＋ＲＷ、ＨＤ－ＤＶＤ、またはブルーレイ（登録商標）ディスク）、フラッシュおよび他の不揮発性ソリッドステートストレージ（たとえば、ＵＳＢフラッシュドライブまたはソリッドステートドライブ（solid state drive：ＳＳＤ））、バッテリによってバックアップされた揮発性メモリ、テープストレージ、リーダ、および他の同様の媒体、ならびにこれらの組合せを含んでいてもよい。

【００３３】

この発明の、コンピュータにより実現される、またはコンピュータにより実行可能なバージョン、もしくはコンピュータプログラム製品が、コンピュータ読取可能媒体を使用して具現化され、コンピュータ読取可能媒体上に格納され、またはコンピュータ読取可能媒体に関連付けられてもよい。コンピュータ読取可能媒体は、命令を実行のために１つ以上のプロセッサに提供することに関与する任意の媒体を含んでいてもよい。そのような媒体は、不揮発性媒体、揮発性媒体、および伝送媒体を含むものの、それらに限定されない多くの形を取ってもよい。不揮発性媒体は、たとえば、フラッシュメモリ、もしくは、光学ディスクまたは磁気ディスクを含む。揮発性媒体は、キャッシュメモリまたはＲＡＭなどのスタティックメモリまたはダイナミックメモリを含む。伝送媒体は、同軸ケーブル、銅線、光ファイバー線、およびバスに配置されたワイヤを含む。伝送媒体はまた、無線波および赤外線データ通信中に生成されるような電磁波、無線周波、音響波、または光波の形を取り得る。

【００３４】

たとえば、本発明のソフトウェアの、機械により実行可能なバイナリバージョンが、ＲＡＭまたはキャッシュメモリ内に、もしくは大容量ストレージデバイス２１７上に格納されてもよく、または存在してもよい。本発明のソフトウェアのソースコードも、大容量ストレージデバイス２１７（たとえば、ハードディスク、磁気ディスク、テープ、またはＣＤ－ＲＯＭ）上に格納されてもよく、または存在してもよい。さらに別の例として、この発明のコードは、ワイヤ、無線波を介して、または、インターネットなどのネットワークを通して伝送されてもよい。

【００３５】

図３は、本発明のソフトウェアを実行するために使用されるコンピュータシステム２０１のシステムブロック図を示す。図２と同様に、コンピュータシステム２０１は、モニタ２０３と、キーボード２０９と、大容量ストレージデバイス２１７とを含む。コンピュータシステム５０１はさらに、中央プロセッサ３０２、システムメモリ３０４、入力／出力（Ｉ／Ｏ）コントローラ３０６、ディスプレイアダプタ３０８、シリアルまたはユニバーサルシリアルバス（universal serial bus：ＵＳＢ）ポート３１２、ネットワークインターフェイス３１８、およびスピーカ３２０といったサブシステムを含む。この発明はまた、追加の、またはより少ないサブシステムを有するコンピュータシステムで使用されてもよい。たとえば、あるコンピュータシステムは２つ以上のプロセッサ３０２を含んでいてもよく（すなわち、マルチプロセッサシステム）、または、あるシステムはキャッシュメモリを含んでいてもよい。

【００３６】

３２２などの矢印は、コンピュータシステム２０１のシステムバスアーキテクチャを表わ

す。しかしながら、これらの矢印は、サブシステム同士をリンクするよう機能する任意の相互接続スキームの例示である。たとえば、スピーカ３２０は、ポートを通して他のサブシステムに接続されてもよく、または、中央プロセッサ３０２への内部直接接続を有していてもよい。プロセッサは、複数のプロセッサまたはマルチコアプロセッサを含んでいてもよく、それは、情報の並列処理を許可してもよい。図２に示すコンピュータシステム２０１は、本発明での使用にとって好適なコンピュータシステムの単なる一例である。本発明での使用にとって好適なサブシステムの他の構成は、当業者には容易に明らかになるであろう。

【００３７】

コンピュータソフトウェア製品は、Ｃ、Ｃ＋＋、Ｃ＃、パスカル（Pascal）、フォートラン（Fortran）、パール（Perl）、（マスワークス（MathWorks）、www.mathworks.comからの）マトラボ（Matlab）、ＳＡＳ、ＳＰＳＳ、ＪａｖａＳｃｒｉｐｔ（登録商標）、ＡＪＡＸ、Ｊａｖａ（登録商標）、パイソン（Python）、アーラン（Erlang）、およびルビー・オン・レイルズ（Ruby on Rails）といった、さまざまな好適なプログラミング言語のうちのいずれで書かれてもよい。コンピュータソフトウェア製品は、データ入力モジュールおよびデータ表示モジュールを有する独立したアプリケーションであってもよい。これに代えて、コンピュータソフトウェア製品は、分散オブジェクトとしてインスタンス化され得るクラスであってもよい。コンピュータソフトウェア製品はまた、（オラクル社（Oracle Corporation）からの）Ｊａｖａ　Ｂｅａｎｓ、またはエンタープライズＪａｖａ　Ｂｅａｎｓ（オラクル社からのＥＪＢ）といったコンポーネントソフトウェアであってもよい。

【００３８】

システム用のオペレーティングシステムは、マイクロソフトウィンドウズ（登録商標）ファミリーのシステム（たとえば、ウィンドウズ（登録商標）９５、９８、Ｍｅ、ウィンドウズＮＴ、ウィンドウズ２０００、ウィンドウズＸＰ、ウィンドウズＸＰ　ｘ６４エディション、ウィンドウズビスタ（Vista）、ウィンドウズ７、ウィンドウズ８、ウィンドウズ１０、ウィンドウズＣＥ、ウィンドウズモバイル、ウィンドウズＲＴ）、シンビアン（Symbian）ＯＳ、タイゼン（Tizen）、リナックス（登録商標）、ＨＰ－ＵＸ、ＵＮＩＸ（登録商標）、サン（Sun）ＯＳ、ソラリス（Solaris）、Ｍａｃ　ＯＳ　Ｘ、アップルｉＯＳ、アンドロイド（登録商標）、アルファ（Alpha）ＯＳ、ＡＩＸ、ＩＲＩＸ３２、またはＩＲＩＸ６４のうちの１つであってもよい。他のオペレーティングシステムが使用されてもよい。マイクロソフトウィンドウズは、マイクロソフト社の商標である。

【００３９】

また、コンピュータはネットワークに接続されてもよく、このネットワークを使用して他のコンピュータにインターフェイス接続されてもよい。ネットワークは、とりわけ、イントラネット、またはインターネットであってもよい。ネットワークは、有線ネットワーク（たとえば、銅を使用）、電話網、パケットネットワーク、光学ネットワーク（たとえば、光ファイバーを使用）、または無線ネットワーク、もしくはこれらの任意の組合せであってもよい。たとえば、データおよび他の情報は、Ｗｉ－Ｆｉ（数例を挙げると、ＩＥＥＥ規格８０２．１１、８０２．１１ａ、８０２．１１ｂ、８０２．１１ｅ、８０２．１１ｇ、８０２．１１ｉ、８０２．１１ｎ、８０２．１１ａｃ、および８０２．１１ａｄ）、近距離無線通信（near field communication：ＮＦＣ）、無線自動識別（radio-frequency identification：ＲＦＩＤ）、モバイルまたはセルラー無線（たとえば、２Ｇ、３Ｇ、４Ｇ、３ＧＰＰ　ＬＴＥ、ＷｉＭＡＸ、ＬＴＥ、ＬＴＥアドバンスト、フラッシュＯＦＤＭ、ＨＩＰＥＲＭＡＮ、アイバースト（iBurst）、ＥＤＧＥエボリューション、ＵＭＴＳ、ＵＭＴＳ－ＴＤＤ、１ｘＲＤＤ、およびＥＶ－ＤＯ）といったプロトコルを使用した無線ネットワークを使用して、コンピュータとこの発明のシステムのコンポーネント（またはステップ）との間で渡されてもよい。たとえば、あるコンピュータからの信号が、コンポーネントまたは他のコンピュータへ、少なくとも部分的に無線で転送されてもよい。

【００４０】

ウェブブラウザがコンピュータワークステーションシステム上で実行される一実施形態では、ユーザは、インターネットなどのネットワークを通して、ワールド・ワイド・ウェブ（World Wide Web：WWW）上のシステムにアクセスする。ウェブブラウザは、ＨＴＭＬ、ＸＭＬ、テキスト、ＰＤＦ、およびポストスクリプトを含むさまざまなフォーマットのウェブページまたは他のコンテンツをダウンロードするために使用されており、システムの他の部分へ情報をアップロードするために使用されてもよい。ウェブブラウザは、ウェブ上のリソースを識別するためにユニフォーム・リソース・アイデンティファイヤ（uniform resource identifier：ＵＲＬ）を使用してもよく、ウェブ上のファイルを転送する際にハイパーテキスト転送プロトコル（ＨＴＴＰ）を使用してもよい。

【００４１】

他の実現化例では、ユーザは、ネイティブアプリケーションおよび非ネイティブアプリケーションのいずれかまたは双方を通して、システムにアクセスする。ネイティブアプリケーションは、特定のコンピューティングシステム上にローカルにインストールされ、そのコンピューティングシステムのオペレーティングシステムまたは１つ以上のハードウェアデバイス、もしくはこれらの組合せに固有である。これらのアプリケーション（「アプリ」と呼ばれる場合もある）は、直接インターネットアップグレードパッチングメカニズムを介して、またはアプリケーションストア（たとえば、アップルｉＴｕｎｅｓおよびＡｐｐ　ｓｔｏｒｅ、グーグルプレイ（Google Play）ストア、ウィンドウズフォン（Windows Phone）ストア、およびブラックベリー（登録商標）Ａｐｐ　Ｗｏｒｌｄストア）を通して、（たとえば周期的に）アップデートされ得る。

【００４２】

システムは、プラットフォームから独立した非ネイティブアプリケーションで動作可能である。たとえば、クライアントは、１つ以上のサーバから、当該サーバとのネットワーク接続を使用して、ウェブアプリケーションを通してシステムにアクセスし、当該ウェブアプリケーションをウェブブラウザにロードすることができる。たとえば、ウェブアプリケーションは、ウェブブラウザによって、アプリケーションサーバからインターネットを通してダウンロードされ得る。非ネイティブアプリケーションも、ディスクなどの他のソースから得られ得る。

【００４３】

図４は、典型的にはセンサ４０９とクラウド４１２との間にあるエッジゲートウェイまたは同等物上で動作するエッジコンピューティングプラットフォーム４０６のブロック図を示す。エッジコンピューティングプラットフォームは、産業用機械および他の産業用インターネット・オブ・シングスを管理し最適化するために重要であるエッジインテリジェンスを導き出すことを可能にする。エッジゲートウェイのコンポーネントは、取込み４２１と、強化４２５と、複合イベント処理（complex event processing：ＣＥＰ）エンジン４２９と、適用４３２と、式言語を通した解析４３５と、移送４３８とを含む。クラウドは、エッジプロビジョニングおよびオーケストレーション４４３と、クラウドおよびエッジ解析ならびにアプリポータビリティ４４６とを含み得る。

【００４４】

上述のように、エッジコンピューティングプラットフォームの特定の一実現化例は、フォグホーンからのものである。フォグホーンは、「エッジインテリジェンス」という急速に台頭してきているドメインにおけるリーダーである。制御システムおよび物理センサのより近くで高性能処理、解析、および異種アプリケーションをホストすることにより、フォグホーンの画期的なソリューションは、閉ループデバイス最適化のためのエッジインテリジェンスを可能にする。これは、製造、石油およびガス、電力および水、運輸、採鉱、再生可能エネルギー、スマートシティなどにおける法人顧客のために、ビッグデータおよび現場でのリアルタイム処理をもたらす。フォグホーンの技術は、世界の有力な産業用インターネットイノベータ、ならびに、クラウドコンピューティング、高性能エッジゲートウェイ、およびＩｏＴシステムインテグレーションにおける主要事業者によって採用されている。

【００４５】
フォグホーンは、ストリームモードおよびバッチモード双方でのエッジアプリのための強化されたＩｏＴデバイスおよびセンサデータアクセス；解析関数を実行するための非常に効率的で表現的なＤＳＬ；低フットプリント機械上で動作可能である強力な小型解析エンジン；さらなる機械学習のために収集されたデータをクラウドへ送信するための発行機能；エッジアプリを開発するためのＳＤＫ（多言語）；構成のエッジデプロイメント、アプリ、および解析式を管理するための管理コンソールを提供する。
【００４６】
フォグホーンは、産業用機械からのセンサデータのリアルタイムの現場ストリーム処理を可能にする、効率的で非常にスケーラブルなエッジ解析プラットフォームを提供する。フォグホーンのソフトウェアスタックは、エッジおよびクラウド上で動作するサービスの組合せである。
【００４７】
「エッジ」ソリューションは、未処理のデータをオフライン解析のためにクラウド環境へ発行するオプションを用いて、ローカルストレージリポジトリへのセンサデータの取込みをサポートしてもよい。しかしながら、多くの産業環境および産業デバイスはインターネット接続性がなく、このデータを使用できなくしている。しかし、インターネット接続性があったとしても、生成されるデータの純然たる量は、利用可能な帯域幅を容易に上回るであろう。または、あまりに法外なコストがかかるためにクラウドに送信できないであろう。加えて、データがクラウドにアップロードされ、データセンターで処理され、結果がエッジに送り返される時までに、行動を取るには遅すぎるかもしれない。
【００４８】
フォグホーンのソリューションは、解析エンジンとしても公知である超小型の複合イベント処理（ＣＥＰ）エンジンと、多くの着信センサストリームのデータに関する規則を表現する強力で表現的なドメイン固有言語（domain specific language：ＤＳＬ）とを提供することによって、この問題に対処する。これらの表現からの出力は次に、高くつく機械故障またはダウンタイムを防止するために、ならびに、リアルタイムの産業活動およびプロセスの効率および安全性を高めるために、直ちに使用され得る。
【００４９】
フォグホーンのプラットフォームは、低フットプリント環境および高スループットまたはゲートウェイ環境で動作する能力；着信ストリーミングセンサデータに作用できる非常にスケーラブルで高性能のＣＥＰエンジン；強化されたデータアクセスを用いた異種アプリの開発およびエッジへのデプロイメント；クラウドおよびエッジを越えるアプリケーションモビリティ；進んだ機械学習（machine learning：ＭＬ）およびクラウドとエッジとの間のモデル転送を含む。独創的に、フォグホーンは、主な産業用データ取込みプロトコル（たとえば、ＯＰＣ－ＵＡ、モドバス（Modbus）、ＭＱＴＴ、ＤＤＳなど）、および他のデータ転送プロトコルをサポートする。加えて、ユーザは、カスタムプロトコルアダプタを、フォグホーンのデータ取込み層へ容易にプラグインすることができる。
【００５０】
フォグホーンのエッジサービスは、ＩＩｏＴデバイスが存在するネットワークのエッジで動作する。エッジソフトウェアスタックは、センサおよび産業用デバイスからのデータを高速データバス上へ取込み、次にユーザ定義の解析式をストリーミングデータに対して実行して、洞察を得てデバイスを最適化する役割を担う。これらの解析式は、フォグホーンの非常にスケーラブルで小フットプリントの複合イベント処理（ＣＥＰ）エンジンによって実行される。
【００５１】
フォグホーンのエッジサービスはまた、時間ベースのセンサデータクエリについてのローカル時系列データベースと、ストリームモードおよびバッチモード双方でデータを消費できるアプリケーションを開発するための多言語ＳＤＫとを含む。オプションで、このデータはまた、顧客が選んだクラウド格納先へ発行され得る。

【００５２】
フォグホーンのプラットフォームはまた、エッジをリモートに構成し管理するためにクラウド環境または構内環境で動作するサービスを含む。フォグホーンのクラウドサービスは、解析式を開発およびデプロイメントし、ドッカー（Docker）（www.docker.com）として公知であるアプリケーションを使用してアプリケーションをエッジへデプロイメントし、顧客のアイデンティティアクセス管理および持続性ソリューションとのサービスの統合を管理するための管理ＵＩを含む。プラットフォームはまた、クラウドで開発された機械学習モデルを、エッジで実行され得るセンサ式へ変換することもできるであろう。
【００５３】
例として、あるアプリケーションは、キャビテーションイベントによるポンプへの高くつく損傷を防止するために、リアルタイムデータ監視および解析、予測保守スケジューリング、ならびに自動化された流れの方向転換を適用する。別の例は、発電を最大化し、機器の寿命を延ばし、正確なエネルギー予測のために履歴解析を適用するためにフォグホーンエッジインテリジェンスソフトウェアを使用した、風力エネルギー管理システムである。
【００５４】
図５は、エッジコンピューティングプラットフォームのより詳細なブロック図を示す。このプラットフォームは、データ取込み５１２、データ処理５１５、およびデータ発行５１８という３つの論理層または区分を有する。データ取込みコンポーネントは、データを生成するセンサまたはデバイス５２３に接続されるエージェント５２０を含む。エージェントは、それぞれのプロトコルサーバからの１つ以上のプロトコルを介して、センサからデータを集め、または取込む。エージェントは、とりわけ、ＭＱＴＴ、ＯＰＣ　ＵＡ、モドバス、およびＤＤＳといったプロトコル用のクライアントまたはブローカーであり得る。センサによって提供または出力されるデータは典型的には、バイナリデータストリームである。センサからエージェントへのこのデータの伝送または配信は、プッシュ法またはプル法によるものであり得る。
【００５５】
プッシュは、所与のトランザクションに対する要求が送信側（たとえばセンサ）によって開始される通信のスタイルを記述する。プル（またはゲット）は、情報の伝送に対する要求が受信側（たとえばエージェント）によって開始される通信のスタイルを記述する。別の通信手法は、受信側またはエージェントが、センサが送信するべきデータを持っているか周期的に問合せるかまたはチェックする、ポーリングである。
【００５６】
ＭＱＴＴ（以前はＭＱテレメトリトランスポート（Telemetry Transport））とは、ＴＣＰ／ＩＰプロトコルに加えて使用するための、ＩＳＯ規格の発行－サブスクライブベースの「軽量」メッセージ通信プロトコルである。代替的なプロトコルは、アドバンストメッセージキューイングプロトコル、ＩＥＴＦ制約アプリケーションプロトコル、ＸＭＰＰ、およびウェブアプリケーションメッセージングプロトコル（Web Application Messaging Protocol：ＷＡＭＰ）を含む。
【００５７】
ＯＰＣ統一アーキテクチャ（OPC Unified Architecture：ＯＰＣ　ＵＡ）とは、ＯＰＣ協議会（OPC Foundation）によって開発された、相互運用性のための産業用Ｍ２Ｍ通信プロトコルである。それは、オープンプラットフォーム通信（Open Platform Communications：ＯＰＣ）の後継版である。
【００５８】
モドバスとは、元々１９７９年にモディコン（Modicon）（現在のシュナイダーエレクトリック（Schneider Electric））によってそのプログラマブルロジックコントローラ（ＰＬＣ）とともに使用するために発行された、シリアル通信プロトコルである。単純かつ頑強であるため、それはそれ以降、すべての意図および目的のために標準通信プロトコルとなった。それは現在、産業用電子デバイスを接続する、一般に利用可能な手段である。
【００５９】

データ処理５１５はデータバス５３２を含み、それは、データ取込み層のエージェント５２０に接続される。データバスは、すべての接続されたコンポーネント間のデータおよび制御メッセージ双方のための中央バックボーンである。コンポーネントは、データバスを通って流れるデータおよび制御メッセージをサブスクライブする。解析エンジン５３５は、１つのそのような重要なコンポーネントである。解析エンジンは、式言語５３８で開発された解析式に基づいて、センサデータの解析を行なう。データバスに接続する他のコンポーネントは、構成サービス５４１と、メトリックサービス５４４と、エッジマネージャ５４７とを含む。データバスはまた、未加工のバイナリデータを消費可能なデータフォーマット（ＪＳＯＮなど）へ復号するとともに、必要で有用な追加のメタデータで装飾することによって、センサからの着信データを強化する「デコーダサービス」を含む。また、強化は、データ復号、メタデータ装飾、データ正規化などを含み得るが、それらに限定されない。

【００６０】

ＪＳＯＮ（ＪａｖａＳｃｒｉｐｔオブジェクト表記法とも呼ばれる）とは、属性値ペアからなるデータオブジェクトを伝送するために人間が読めるテキストを使用する、オープンスタンダードフォーマットである。ＪＳＯＮは、非同期ブラウザまたはサーバ通信（ＡＪＡＪ）または双方のために使用される共通データフォーマットである。ＪＳＯＮの代替例はＸＭＬであり、それはＡＪＡＸによって使用される。

【００６１】

エッジマネージャはクラウド４１２に、特にクラウドマネージャ５５２に接続する。クラウドマネージャは、同様にクラウドにある、顧客アイデンティティおよびアクセス管理（identity and access management：ＩＡＭ）用プロキシ５５５とユーザインターフェイスコンソール５５８とに接続される。また、クラウドを介してアクセス可能なアプリ５６１もある。アイデンティティおよびアクセス管理とは、正しい個人が正しい時間に正しい理由で正しいリソースにアクセスすることを可能にする、セキュリティおよびビジネス規律である。

【００６２】

データ処理５１５内では、ソフトウェア開発キット（software development kit：ＳＤＫ）５６４コンポーネントもデータバスに接続し、それは、エッジゲートウェイ上にデプロイメントされ得る、機能するアプリケーション５６７の作成を可能にする。ソフトウェア開発キットはまた、データを取出すためにローカル時系列データベースに接続する。アプリケーションは、ドッカーなどのコンテナ技術を使用することなどによってコンテナ化され得る。

【００６３】

ドッカーコンテナは、１つのソフトウェアを完全なファイルシステムで包み、ファイルシステムは、コード、ランタイム、システムツール、およびシステムライブラリといった、それが動作するために必要なものすべてを含み、サーバにインストールされ得るものを何でも含む。これは、ソフトウェアが、それが動作している環境にかかわらず、常に同じように動作することを保証する。

【００６４】

データ発行５１８は、クラウド内の格納位置５７３に接続されるデータパブリッシャ５７０を含む。また、ソフトウェア開発キット５６４のアプリケーション５６７は、時系列データベース５７６内のデータにアクセスできる。時系列データベース（time-series database：ＴＳＤＢ）とは、時系列データ、時間によって索引付けされた数の配列（たとえば、日付－時刻、または日付－時間範囲）を取扱うために最適化されたソフトウェアシステムである。時系列データベースは典型的には、新しい情報がデータベースに追加されると最も古い情報が除去される、回転または循環バッファまたはキューである。データパブリッシャ５７０もデータバスに接続し、ローカル時系列データベースに、またはクラウドストレージに格納される必要があるデータをサブスクライブする。

【００６５】

図６は、エッジインフラストラクチャ６０２とクラウドインフラストラクチャとの間の動作フローを示す。いくつかの特定のエッジインフラストラクチャは上に説明された。センサ６０６からデータが集められる。これらのセンサは、産業用デバイス、小売デバイス、ヘルスケアデバイス、または医療デバイス、もしくは、電力または通信用途、もしくは、これらの任意の組合せのためのものであり得る。

【００６６】

エッジインフラストラクチャはソフトウェアプラットホーム６０９を含み、それは、データ処理６１２と、ローカル時系列データベース６１５と、クラウドシンク６１８と、解析複合イベント処理エンジン（ＣＥＰ）６２１と、解析リアルタイムストリーミングドメイン固有言語（ＤＳＬ）６２４（たとえば、フォグホーンによるＶｅｌ言語）と、リアルタイムアグリゲーションおよびアクセス６２７とを有する。プラットフォームは、以下により詳細に説明される仮想センサ６３０を含み得る。仮想センサは、強化されたリアルタイムデータアクセスを提供する。

【００６７】

プラットフォームは、ソフトウェア開発キットまたはＳＤＫを使用して開発され得る、アプリまたはアプリケーション１、２および３などの１つ以上のアプリケーション６３３を介してアクセス可能である。アプリは（たとえば、複数の異なる言語で開発された）異種であってもよく、複合イベント処理エンジン６２１を活用するとともに、機械学習を行なうことができる。アプリは、アプリストア６３７を使用して配布可能であり、アプリストア６３７は、エッジプラットフォーム開発者またはエッジプラットフォームの顧客（パートナーと呼ばれてもよい）によって提供されてもよい。アプリストアを通して、ユーザは、アプリをダウンロードし、他人と共有することができる。アプリは、機械学習、遠隔監視、予測保守、または動作インテリジェンス、またはこれらの任意の組合せを含む、解析および適用６３９を行なうことができる。

【００６８】

アプリについては、エッジとクラウドとの間に動的アプリモビリティがある。たとえば、フォグホーンのソフトウェア開発キットを使用して開発されたアプリケーションは、エッジ上に、またはクラウド内にデプロイメント可能であり、それにより、エッジとクラウドとの間にアプリモビリティを達成する。アプリは、エッジの一部として、またはクラウドの一部として使用され得る。一実現化例では、この特徴はアプリがコンテナ化されることによって可能になり、そのため、アプリは、それらが実行されるプラットフォームから独立して動作可能である。同じことが解析式についても言える。

【００６９】

クラウドまたは私的データセンター６４４でのデータの監視または格納を含む、統合運営および管理６４０を可能にするデータアプリがある。

【００７０】

物理センサは電子変換器であり、それは、その環境のいくつかの特性をアナログまたはデジタル測定値として測定する。アナログ測定値は典型的には、アナログ／デジタル変換器を使用してデジタル量に変換される。センサデータは、必要ベースで測定（ポーリング）されるか、または、一定速度のストリームとして利用可能である。典型的なセンサ仕様は、範囲、精度、解像度、ずれ、安定性、および他の属性である。多くの測定システムおよびアプリケーションは、処理、移送、または格納のためにセンサデータを直接利用し、または通信する。

【００７１】

システムは、解析式言語を使用して作成されたソフトウェアベースのセンサである、仮想センサとも呼ばれる「プログラマブルソフトウェア定義センサ」を有する。一実現化例では、解析式言語は、フォグホーンの解析式言語である。この式言語はＶｅｌとして公知である。Ｖｅｌ言語は、実行の待ち時間が少ない制約された低フットプリント環境においてリアルタイムストリーミング解析をサポートするために効率的に実現される。たとえば、システムの待ち時間は約１０ミリ秒以下であり得る。

【００７２】
一実現化例では、プログラマブルソフトウェア定義センサは、「センサ式言語」（sensor expression language）またはＳＸＬと呼ばれる宣言型アプリケーションプログラムインターフェイス（ＡＰＩ）を用いて作成される。ＳＸＬ言語の特定の一実現化例は、フォグホーンからのＶｅｌである。Ｖｅｌセンサは、この構成を通して作成されたＶｅｌセンサであり、物理センサおよびＶｅｌセンサを含む複数のソースによって生成されたデータを処理することから導き出された測定値を提供する。本願では、ＶｅｌおよびＳＸＬは交換可能に使用される。
【００７３】
Ｖｅｌセンサは、これら３つのソースのうちのいずれか１つまたはそれらの組合せから導き出され得る。
【００７４】
１．単一のセンサデータ。
１．１．単一の物理センサから導き出される仮想センサまたはＶｅｌセンサは、任意の組合せの動的較正、信号処理、数式、データ圧縮またはデータ解析を使用して、着信センサデータを変換し得る。
【００７５】
２．複数の物理センサデータ。
２．１．複数の異種物理センサから（上述の方法を使用して）変換として導き出された仮想センサまたはＶｅｌセンサ。
【００７６】
３．Ｖｅｌセンサ装置の実現にとって利用可能にされた、物理センサデータと仮想センサデータとの組合せ。
【００７７】
Ｖｅｌセンサはドメイン固有であり、特定のアプリケーションを念頭に置いて作成される。Ｖｅｌプログラミングインターフェイスの特定の一実現化例は、アプリケーションが、変換（たとえば数式）およびアグリゲーションを通してデータ解析を定義することを可能にする。Ｖｅｌは、典型的にはプログラミング言語に基づいた１組の数学演算子を含む。Ｖｅｌセンサは、Ｖｅｌ構成またはプログラムを実行することによって、ランタイムにデータに作用する。
【００７８】
Ｖｅｌセンサの作成：Ｖｅｌセンサは、データをリアルタイムで利用可能にするためのソフトウェア装置として設計される。これは、アプリケーションによって必要とされるレートでＶｅｌセンサデータを生成するために、Ｖｅｌで開発されたアプリケーションが、組込まれたコンピュートハードウェアに対してリアルタイムで実行されることを必要とする。これを達成するために、システムは高効率の実行エンジンを含む。
【００７９】
Ｖｅｌセンサの利点は、以下を含む。
１．プログラム可能性：Ｖｅｌは、Ｖｅｌセンサをプログラマブルにして、データを合成し、データ品質、頻度および情報についての特定のアプリケーション要件に整合させる。Ｖｅｌセンサは、物理センサおよび他の（たとえば、前から存在する）Ｖｅｌセンサから供給されたデータにプラグインするための無線ソフトウェアアップグレードとして広く分散され得る。このため、アプリケーション開発者は、物理インフラストラクチャのレイアウトから独立したビジネスロジックの効率的実行の助けとなるデジタルインフラストラクチャを作成することができる。
【００８０】
２．保守性または透明性：Ｖｅｌセンサは、アプリケーションと物理センサとの間にデジタル抽象化層を作成し、それは、物理センサへのアップグレードおよびサービスによる物理インフラストラクチャの変化から開発者を隔離する。
【００８１】

３．効率：Ｖｅｌセンサは、物理センサからの未加工データをそれらに含まれる情報の正確な表現へ変換することによって、情報管理における効率を生み出す。この効率は、アプリケーションにおける下流での計算、ネットワーキング、および格納のようなＩＴリソースの効率的な利用につながる。

【００８２】

４．リアルタイムデータ：Ｖｅｌセンサは、実在または物理センサデータストリームから計算されるリアルタイムセンサデータを提供する。これは、データを最小の時間遅延でアプリケーションにとって利用可能にする。

【００８３】

実現化例：システムは、Ｖｅｌインターフェイスに基づいて、Ｖｅｌセンサのスケーラブルでリアルタイムの実現化例を設計した。Ｖｅｌは、Ｊａｖａ言語によってサポートされる演算子を含み、物理センサおよびそれらのプロトコルと良好に統合される。

【００８４】

システムは、実行される物理センサのデータに対する演算を正確に表現するための新規の方法をもたらす。この宣言型の式は、デジタル抽象化の定義を、物理センサに対する実現から隔てる。

【００８５】

さまざまな型のデータのストリームのセットと、それらのストリームにおけるデータの特定のパターンに反応しそれら特定のパターンを処理することを意図した関数のセットとが与えられると、本発明は、データがストリームで到着するとこれらの関数が適切かつ効率的に呼び出され得るようにこれらの関数を記述および変換する技術である。

【００８６】

この種の問題を解決する必要性は、すべての形式のデータフロープログラミングで共通して発生する。これは、エンタープライズデータセンター内およびエンタープライズデータセンター間のデータフローなどの非常に大規模なアーキテクチャ、ならびに埋込まれたデバイスにおけるイベントのフローなどの非常に小規模なアーキテクチャに適用可能である。

【００８７】

本発明は、データフロープログラミングのすべての領域に適用可能であるが、一致を検出でき、ハンドラ関数を適用できる速度が最も重要であり、実行に費やすストレージおよび計算リソースが限られている場合に最適である。

【００８８】

実施例。所与の整数のストリームから、１つ以上の非零値に１つ以上の零が続くものをマッチングしたい。このパターンがマッチすると、非零値の和を計算し、その結果を別のストリームに書き込みたい。

【００８９】

この問題のパターンマッチング部分を正規表現の表記法で書き、和の計算を別途算術式として書くことができる。それが起こると、エッジコンピューティングにおいてデータフローアプリケーションで使用するために設計されたＶｅｌプログラミング言語を使用することにより、変換全体を統一された表記法で書くことができ、したがって、

【００９０】

【表１】

```
stream(“output”) =
( a:{!= 0} .. {>0}, :0 .. {>0} -> sum(a) )
from stream(“input”)
```

【００９１】

この技術は、上記の関数のパラメータ化を状態機械に変換するであろう。次に、それは、一致をその状態機械に基づいて決定性有限オートマトンとして実現し、結果の一致を合計

式に供給するであろう。このフローを図７に示す。これは、状態０　７０５、状態１　７１０、「リストａから」ブロック７１５、および「和（ａ）をプッシュする」ブロック７２０である。

【００９２】

この問題は、各ハンドラ関数ごとにマッチング関数を生成することで解決できる。マッチング関数は、ストリームからデータのウィンドウを入力として受け入れ、一致の場合は真を返し、不一致の場合は偽を返す。データがウィンドウを通って流れると、一致が見つかるまで、マッチング関数を繰り返し適用する必要がある。一致が見つかると、ハンドラ関数が適用される。

【００９３】

この解決策は、ハンドラ関数がデータベースクエリに対して使用されるのと同様の方法で指定されるため、生ずる。ＳＱＬのようなＷＨＥＲＥ節は、一致のための条件を記述するブール式を与え、マッチング関数はこの式の直接的コンパイルである。

【００９４】

別々のマッチング関数は、新たなデータがストリームバッファに流入するときに個別に評価されなければならない。一致は各関数ごとに独立して判断される。

【００９５】

状態機械を使用して一致を実行する方が、複数の任意のブール式を繰り返し適用するよりも、効率的である。

【００９６】

本発明は、関数のパラメータを宣言するパターン記述言語から状態機械を導出する。導出された状態機械は、データストリーム内において、一致を、従来のブール式マッチング関数よりも、より効率的に検出する。

【００９７】

導出された状態機械はまた、データストリームにおいて検出された一致のためにハンドラ関数のセットを実装してもよい。複数のマッチング関数および対応するハンドラ関数を組み合わせて、どのハンドラ関数に対しても効率的に一致を認識する単一の状態機械に縮小してもよい。

【００９８】

導出された状態機械はまた、状態機械によって認識されるシーケンスを変更することなく、追加のノードを介する自由（イプシロン）遷移を含むように拡張されてもよい。

【００９９】

このような追加のノードを介して遷移することにより、データ上でさまざまなアクションをトリガしてもよい。たとえば、決定性有限オートマトン（ＤＦＡ）またはスタックマシンのシフトバッファにおけるデータの、保持領域への収集をトリガすることができる。これらのデータは、後で関数適用に対する引数の基礎を形成してもよい。

【０１００】

この出願ではＤＦＡという用語を使用しているが、これらのオートマトンまたはユニットはスタックマシンと称されてもよい。厳密に言えば、決定性有限オートマトンは、空間における有限の性能を意味する。しかしながら、この特許のオートマトンは、必ずしも有限であるとは限らず、非有限であり得るが、依然として単純であり得る。したがって、この特許に記載されているＤＦＡは、非有限であってもよい。

【０１０１】

このような追加のノードを介して遷移することにより、以前のノードで取り込まれたデータを関数適用引数として使用して、ハンドラ関数の呼び出しをトリガすることもできる。

【０１０２】

正規表現および値式の局面を組み合わせたスクリプトからの変換により、パターンのマッチングおよび値の計算を効率的に行い得る拡張された状態機械またはＤＦＡが生まれる。

【０１０３】

結果として生じる組み合わされたマッチングまたは計算アルゴリズムは、パターンマッチ

ングおよび値計算の別個の編成よりも効率的である。
【０１０４】
語彙ソースからＤＦＡまたは状態機械を構築する方法は、非決定性有限オートマトン（ＮＦＡ）で始まり、次いでそれを最小のＤＦＡに縮小する。ＤＦＡの目的は、一連の入力データ内でパターンを認識することである。この議論のため、状態機械を通って流れるデータをトークンと呼び、ＤＦＡによって認識される特定のパターンをトークンの言語と呼ぶ。
【０１０５】
図８のＮＦＡの部分を考える。この部分は偶然ＤＦＡでもあるが、この例では重要ではない。それは、トークンアルファの受信で、状態Ａ８０５から状態Ｂ８１０に遷移する。
【０１０６】
図９に示すように、イプシロン遷移９２０を有する追加のノードを追加することによって、このＮＦＡを拡張することができる。イプシロンエッジは、入力の状態にかかわらず、いつでも‐あたかも自由に‐辿られることができる。
【０１０７】
１つ以上のイプシロンエッジが存在すると、状態機械は非決定性になるが、しかし、イプシロンエッジはアルゴリズムによって除去されてもよく、ＮＦＡはこの手段によって同等のＤＦＡに縮小され、表駆動法によって効率的に実施されることができる。したがって、効率的な実装の戦略を依然として維持しながら、これらの別途のイプシロン遷移を導入することができる。
【０１０８】
図９の状態機械は、トークンアルファ９２５の受信で状態Ａ９０５から状態Ｘ９１５に遷移し、それから随意に状態Ｘから状態Ｂ９１０に進むことができる。アルファの刺激は、図８におけるより単純な機械と同様に、依然として状態Ａから状態Ｂへの遷移をもたらす結果となり、この遷移を達成するために追加の入力は必要ない。したがって、図９のＮＦＡは、図８と同じ言語を変換することが分かる。そうするためには、単に、状態Ｘを通る別途の状態遷移が必要である。
【０１０９】
別途の状態は、副作用のパフォーマンスをそれに関連付けるために役立つ。これらの副作用が状態機械の定義も状態機械を流れるデータも変更しない限り、追加のノードは言語の認識に影響を与えないが、副作用は追加作業を行い得る。
【０１１０】
データフロー反応実現例では、追加作業には、データに対する任意の数の有用なアクション、またはそのデータを使用することを含めることができる。例示的な一実現例では、作業は以下を含む。
【０１１１】
１．ノードを流れるデータを検査し、それのコピーを外部コレクタに発する。
２．データがノードを通過する際にデータに変換を適用し、変換されたデータを一時バッファに収集する。または、
３．収集したデータを一時バッファから追加の変換にフラッシュし、その結果を別のＤＦＡまたはスタックマシンにプッシュする。
【０１１２】
例として、ソースフラグメントを考える：
【０１１３】
【表２】

```
( a:{!= 0} .. {>0}, :0 .. {>0} -> sum(a) )
```

【０１１４】
フラグメントは、２つの項からなるパターンを記述する。（１）ａと呼ばれる第１項。非零値の１回以上の繰り返しとマッチする。（２）名前が与えられていない第２項。１つ以

上の零の繰り返しとマッチする。
【０１１５】
これを反応の基礎として使用したいと考える。呼び入れられたソースから値を読み、入力の中でフラグメントのパターンを認識すると、フラグメントの右辺を評価し、呼び出された宛先に結果をプッシュすることで、反応する。
【０１１６】
たとえば、値[101, 202, 303, 0, 0]からなる場合、最初の３つの値をａに、最後の２つの値を匿名の第２項にバインドすることで、パターンをマッチングするであろう。次に、ａにバインドされた値[101, 202, 303]のリストに合計関数を適用することによって、右辺を評価し、６０６を返すであろう。次いで６０６をプッシュアウトするであろう。
【０１１７】
本発明によるこの例のような関数パターンの変換は、コンピュータ実行の変換プログラムを介して実施することができる。プログラムは、２つの異なる形式の変換：関数指向部分“sum(a)”を、計算を実行するであろう実行可能文のブロックに変換すること、およびパターン指向部分“a:{!= 0} .. {〉0}, :0 .. {〉0}”を、パターンを認識し、引数を取り込み、関数を呼び出すであろうＤＦＡまたはスタックマシンに変換すること、を実行しなくてはならないだろう。前者をタスク関数変換と呼び、後者をタスクパターン変換を呼ぼう。
【０１１８】
関数変換は、コンパイラおよびインタプリタの書込みを専門とするコンピュータプログラマによってよく理解されている。パターン変換、関数変換とパターン変換とのフィッティング、ならびにその後のパターン認識および関数ディスパッチの自動化は、本発明の主題である。
【０１１９】
関数変換は、ソーステキストを受け入れ、テキストをトークンに分割し、その後、文法によって導かれて、ソーステキストの構文的内容を記述する抽象構文木（ＡＳＴ）の葉を形成するようにトークンを配置することからなる。次に、抽象構文木は、ソースによって記述される関数を評価するために必要な命令のブロックを最終的に生成する一連のアルゴリズムによって走査される。
【０１２０】
パターン変換は、上述の構文解析によって形成された抽象構文木から始まる。抽象構文木には、パターン宣言のルートを形成する１つ以上のノードが含まれる。たとえば、上記の我々のパターンは、図１０の左下に示すように、２つの子を有する単一のルートノードで構成され、各子はパターンの１つの項を記述するかもしれない。図１０において、反応ルートノード１００５、パターンルートノード１０１０、和(a)ノード１０１５、ノード１０２０、および〈no name〉（名前無し）ノード１０が存在する。
【０１２１】
マッチングする例およびそれをマッチングする繰り返しを認識するように指定するパターン項ノードが、正規表現における項と同じ情報を担持することを認識する。さらに、子ノードのシーケンスは、まとめて順に、正規表現の項の線形連言（linear conjunction）と同じ情報を特定する。正規表現または正規表現の項の線形連言は、ＮＦＡへの変換項とすることができる。我々は、本発明において同じアルゴリズムが使用され得、パターン項は正規表現の項の代わりとなることを発見した。
【０１２２】
基本的なＮＦＡがそのように形成されたら、それに、別途の、副作用誘導状態を、アクションがパターン項によって必要とされる位置において注入し、受容状態の後、反応の関数を呼び出すことができる。
【０１２３】
この例を続けると、項ａは、それにマッチする値のリストを収集し、最終的にそれらを引数として反応の関数に渡し得ることを必要とする。したがって、図９に示す変換を項ａの結果であるＮＦＡ状態に適用し、新たな状態を使用して、マッチングする項を収集する作

業を行なう。次いで、変換を、再び、今度はＮＦＡの受容状態に適用し、収集された値を使用して反応の関数を呼び出し、結果を反応のコンシューマにプッシュし、収集バッファをクリアする。この強化されたＮＦＡがＤＦＡに変換され、状態が縮小された後、図７に示すマシンが残る。

【０１２４】

ステップは、状態機械エンジンを駆動するよう状態アクション表を用いるためのアルゴリズムのように、ＮＦＡをＤＦＡに変換し、ＤＦＡを状態縮小し、ＤＦＡを状態アクション表として表現するために使用される。

【０１２５】

本発明の技術によって生成されたＮＦＡは、変換されて表に表現されることができる。ただし、結果の表には、各状態を通過するときに実行される副作用ラムダからなる別途の列が含まれる。このような状態アクションラムダ表を使用する自動化エンジンは、他の技術とは異なり、遷移するたびに追加のラムダを実行する。

【０１２６】

データフローコンピューティング環境で使用するためのリアクティブ関数を記述し変換する方法であって、（ｉ）リアクティブ関数を識別することと、（ｉｉ）関数への入力を提供するパラメータのパターンを識別することと、（ｉｉｉ）関数に渡された引数に基づいて評価されるべき式を識別することと、（ｉｖ）パラメータのパターンを、パターンにマッチする入力のシーケンスを認識することができる状態機械に変換することと、（ｖ）状態機械を、入力データを収集および変換してそれを関数に対する引数として使用する準備をする作業を行なう追加の状態で拡張することと、（ｖｉ）状態機械を、単純なソフトウェアまたはハードウェアによる自動化が可能な状態アクション効果表に縮小することとを備える。

【０１２７】

関数のセット、および引数として値のシーケンスを与えられると、本発明は、引数がマッチする関数に実行をディスパッチするか、または引数がいずれの関数ともマッチしないと判断する、方法である。この方法は、値式、型式、および正規表現を組み合わせることによって、それが、型システムで表現可能な値のシーケンスを曖昧さなしにマッチングできるという点において新規である。

【０１２８】

この種の問題を解決する必要があるのは、トランスレータ、インタープリタ、コンパイラの開発においてであり、多相的ディスパッチの概念と密接に関係している。シーケンスの任意の接頭辞を構成する要素が単一のオブジェクト（タプル）を構成すると考えるならば、正しい関数にディスパッチするタスクは、タプルのクラスのメソッドの多相的ディスパッチと等価であると考えることができる。

【０１２９】

本発明は、この種の多相的ディスパッチが必要とされるあらゆる状況に適用可能である。これには、プログラムの外部から発生したデータのストリームに応答する必要があるイベント駆動型またはリアクティブ型プログラムのすべての態様が含まれる。本発明は、エッジもしくはフォグコンピューティング環境またはネットワーク接続環境で頻繁に発生するような、複数のデータストリームのリアルタイム処理に関する適用例において特に有用である。

【０１３０】

正規表現は、通常、特定のパターンに適合する文字列を検出するために使用される。最も密接に関連する多くの正規表現言語、およびそれらに基づいた効率的なマッチングエンジンを実装する多くのツールがある。これらは、一般に、文字のシーケンスのマッチングに限定される。

【０１３１】

文字列以外のドメインで動作する他のパターンベースの表記法がある。ＸＭＬ文書においてパターンを記述するＸＰＡＴＨがその一例である。これらの表記法は、通常、正規表現

よりも完全性が低く、パワフルではないことが多く、特定のドメインに合わせて調整されている。

【０１３２】

いくつかのプログラミング言語は、型ベースのパターンマッチングシステムを用いてランタイム多相的ディスパッチを実行する。関数の複数のオーバーロードが定義され、それらの各々は異なるパターンの型および値をとり、引数の型および値を関数パラメータのパターンと照合することによって、ディスパッチが実行時に解決される。Haskellはそのようなプログラミング言語の１つである。

【０１３３】

言語仕様言語は、文脈自由文法を一連の生成規則として記述する。これらの規則は言語の構文を構成する。コンパイラコンパイラは、これらの規則を、言語のインスタンスを認識できる表駆動型決定性有限状態機械に変換する。Bisonは、そのような言語仕様言語およびそれの関連するコンパイラコンパイラの例である。

【０１３４】

正規表現などの文法駆動型パターンマッチングシステムは、決定性有限オートマトン（ＤＦＡ）や状態機械などの単純な機械として表現できるため、効率的実行の利点があるが、それらには、完全な型システムの幅広いモデリング機能はない。Haskellで使用されているような型駆動型パターンマッチングシステムは、はるかにより豊富なモデリング機能を備えているが、合理的に効率的な実装を優先して表現できるものをしばしば犠牲にし、それでも依然として、ＤＦＡに基づく高速マッチングシステムほど効率的ではない。

【０１３５】

本発明は、型に基づくマッチングシステムを扱い、それは、その型の中で表現可能なすべての状態と照合することができながら、依然として状態機械として効率的に実施され得る。型と状態との一般化されたパターンは、パターンのインスタンスを効率的に認識する表駆動型状態機械に変換される。

【０１３６】

これらのパターンに基づいて関数パラメータを定義することにより、関数はまさにどのような恣意的なデータパターンともマッチングし、マッチングにおいて、マッチングするデータ要素間からその引数をバインドすることができる。メンバー関数の状態機械をマージし、その結果を最小の状態数に縮小することによって、関数の合併（union）のためのマッチングパターンを記述する状態機械が形成される。オーバーロード間の曖昧さ回避、または全体的な不一致の検出は、シーケンス内で可能な限り早く起こり、関数適用の解決をスピードアップする。また、シーケンス内でできるだけ遅くまで一致を遅らせて、可能な限り多くの入力を受け入れる関数の「貪欲」バージョンを生成することもできる。

【０１３７】

あるメソッドは、値式、型式、および正規表現を組み合わせて、それが、型システムで表現可能な任意の値のシーケンスと曖昧さなくマッチングし得るようにする。このメソッドは関数適用を解決し、最小限の判断数で正しいオーバーロードにディスパッチする。このメソッドにより、オーバーロードされた関数適用は、文脈自由文法と同じ作業を実行し、文法的な下位構成要素を帰納的に認識して変換関数をそれに適用することによって特定の言語を認識することができる。

【０１３８】

このメソッドは、複数の異なる型を含む型システムに関連して適用可能であり、たとえば、（１）整数、実数、および文字列などの基本的な単相型の集合（set）。（２）多相型の集合およびそれらのコンストラクタ、特に、我々が間もなく論ずるある特定のプロパティを持つ多相集合型。（３）sum型。（４）レコードの形式のproduct型。（５）パターンの形式のproduct型。タプルの、それのフィールドの繰り返しを含むことへの一般化である。（６）ラムダ型。パターン型を任意の型にマッピングする。（７）ラムダのリストからなる多ラムダ型。

【０１３９】

集合は、１つ以上の要素の範囲からなる多相型である。集合型は、それが含む要素の型上においてパラメータ化され、たとえば、整数の集合は文字列の集合とは異なる型である。集合型は、その内容の制限によってさらに特徴づけられる。特に、集合型は、有限もしくは無限であるよう、またはその左辺もしくは右辺において閉じられているかもしくは開いているよう、またはこれらの任意の組み合わせであるように、制約されてもよい。以下の整数の集合の例を考えてみよう。

【０１４０】

【表３】

表Ａ

表記	長さ	閉度	意味
[1]	1	左右で閉じている	１つの整数1からなる集合。
[1, 2, 3]	3	左右で閉じている	３つの整数1, 2, 3からなる集合。
[5000 .. 6000]	1001	左右で閉じている	5000から6000までの整数。
[10 ..]	無限	左で閉じ、右で開いている	10以上のすべての整数。
[.. 10]	無限	左で開き、右で閉じている	10以下のすべての整数。
[> 5]	無限	左で閉じ、右で開いている	5より大きいすべての整数。 [6 ..]と同じ。
[>= 5]	無限	左で閉じ、右で開いている	5以上のすべての整数。 [5 ..] と同じ。
[< 5]	無限	左で開き、右で閉じている	5未満のすべての整数。 [.. 4]と同じ。
[<= 5]	無限	左で開き、右で閉じている	5以下のすべての整数。 [.. 5]と同じ。
[!= 5]	無限	左右で開いている	5を除くすべての整数。
[>= 1] and [<= 3]	3	左右で閉じている	３つの整数1, 2, 3からなる集合。 [1, 2, 3]または[1 .. 3]と同じ。
[<= -10] or [>= 10]	無限	左右で開いている	絶対値が10以上のすべての整数。
not [1 .. 3]	無限	左右で開いている	1, 2, 3を除くすべての整数。

【０１４１】

[〉 = 1]と[〉 0]との間に区別はなく、なぜならば、要素は整数型であり、整数は相異なって可算であるからである。要素が、仮に、実数や文字列のように、非可算型である場合、明示的な包含または特定の端点の包含が必要となる。たとえば、集合[〉 = “ cat ”]は、文字列“ cat ”、および辞書編集上“ cat ”の後に分類するすべての文字列で構成されている。

【０１４２】

集合のインスタンスを型として使用してもよい。そのような型のインスタンスは、集合の要素でなければならない。たとえば、型として使用される集合[〉 0]は、正の整数だけを値として許可するであろう。実際、すべての型をこのように考えることができる。たとえば、単相の整数型は、すべての整数の集合からなる集合型と考えることができる。

【０１４３】

我々のsum型は、他の型の単純な合併（union）である。たとえば、型int（整数）または型string（文字列）は、その２つの構成要素型の和である。sum型の構成要素型のいずれかの何れのインスタンスも、sum型のインスタンスである。これにより、たとえば、各々が整数または文字列のいずれかである値のリストである型リスト（intまたはstring）を記

述することができる。unionのunionは平坦（flatten）になり、型式（intもしくはstring）または（intもしくはreal（実数））はintまたはrealまたはstringに等しくなる。unionにおける型の順序は重要ではないが、正規性のため、すべてのunion型をここにおいてそれらの構成要素がアルファベット順であるように提示する。

【０１４４】

我々のレコード型は名前付きフィールドを使用し、各フィールドを型に関連付ける。たとえば、{birthday: date; first_name: string; last_name: string}（{誕生日: 日付; 名: 文字列; 姓: 文字列}）レコード型は常に有限数のフィールドを有し、各フィールドは型内において一意の名前を有する。フィールドの順序は重要ではない。{x: int; y: int}は{y: int; x: int}と同じであるが、unionに対してしたように、我々はレコード型を、それらの構成要素がアルファベット順にある状態で提示する。

【０１４５】

レコードの型はそれ自体がレコードであることに注意されたい。値{x: 3; y: 4}は型{x: int; y: int}を有する。

【０１４６】

我々のパターン型は、型のシーケンスとして定義されている点で、タプルに似ている；しかしながら、タプルは暗示的にその要素の各々が正確に１回現れると仮定しているが、パターンはそれの各要素が繰り返しを有することを許す。この繰り返しは整数の集合として与えられる。たとえば、パターン〈a: int # [1 .. 3]; b: string # [1 .. 3]〉は１つから３つの整数の後に１つから３つの文字列が続くものとマッチする。

【０１４７】

ラムダのパラメータとして使用すると、パターンのフィールドはラムダの評価内でバインドされる引数を生じさせる。たとえば、前の段落で与えられたパターンをマッチングした後、範囲に２つのローカル識別子ａおよびｂを有するであろう。Ａの値は１つから３つの整数のリストになり、ｂの値は１つから３つの文字列のリストになるであろう。

【０１４８】

パターンについて１つ以上のフィールドが名前を持たないことも有効である。名前のないフィールドはマッチングされるが、それに対するどのような値も引数としてバインドされない。たとえば、〈a: int # [1 .. 3]; string # [1 .. 3] 〉をマッチングした場合、前のように１つから３つの整数の後に１つから３つの文字列が続くものをマッチングさせ、整数をａというリストとしてバインドするが、文字列はバインドしないであろう。

【０１４９】

パターンは無限長であってもよい。たとえば、パターン〈a: int # [1 ..]〉は１つ以上の整数と上限なしでマッチする。これは有効である。ただし、無限入力ストリームを処理するために使用する場合、無限のパターンを、値の収集をいつ停止すべきか示す時間間隔などの他のトリガとペアにする必要がある。

【０１５０】

一般に、パターンは、それがマッチするデータを消費するが、しかしながら、そのデータの部分集合のみを消費することも、またはまったく消費しないことが可能である。パターンには、ピークポイントと呼ばれるアットマークが含まれてもよく、それを越えて、パターンはデータとマッチし、引数をバインドするが、入力ストリームからは消費しない。たとえば、パターン〈a: int; b: int; peek; c: int〉 は３つの整数とマッチし、３つのローカル識別子をバインドするが、入力から２つの整数しか消費しない。

【０１５１】

フィールドのないレコードまたはフィールドのないパターンを有することは有効である。これらの２つのケースは、両方ともproduct型を意味するため、互いに有意に区別できない。語彙的には、この概念をボイドというキーワードで指定する。ボイドは一意の値である。それ自身の型でもある。合併で使用される場合、ボイドはintまたはvoidなど、任意選択的な型の表記をもたらし、存在する場合にはintである値を意味するが、まったく存在しないかもしれない。

【０１５２】
我々の目的のために、型マッチングは構造的であり、名目上のものではない。型には名前はなく、記述だけがある。同じ記述を有する２つの型は同じ型である。記述が別の型の記述の部分集合である型は、その型の一般化である。たとえば、型{x: int; y: int}および{x: int; y: int; z: int}を考える。２つのフィールド-xおよびy-を有する型は、３つのフィールド-x, yおよびz-を有する型の部分集合であるため、前者は後者の一般化と見なすことができる。これはパターンにも当てはまる。別のものの接頭辞であるパターンもその一般化である。
【０１５３】
我々のラムダ型は、入力パターンを出力型にマッピングする。たとえば、〈int # [1 .. 3]〉 → intは、１つから３つの整数をとり、ある整数を返す、ある関数の型である。我々の多ラムダ型は、ラムダ型のリストで構成されている。ラムダの順序はここでは重要である。多ラムダ適用を解決する際、それの構成要素ラムダのうちマッチする最初のラムダにディスパッチすることになる。
【０１５４】
このように定義されると、多ラムダをディスパッチするのに必要なパターンマッチングは、決定性有限オートマトン（ＤＦＡ）に縮小され得る。どのようにするかを示すために、比較のための基礎として状態機械の構築方法を使用し、必要に応じてそれを拡張する。ある記述は、まず、非決定性有限オートマトン（ＮＦＡ）を構築すること、および次いでそれをＤＦＡに縮小することを含むが；しかしながら、実際には、これは一般に単一のステップで行うことができる。
【０１５５】
前述したように、この出願ではＤＦＡという用語を使用しているが、これらのオートマトンまたはユニットはスタックマシンと称されてもよい。厳密に言えば、決定性有限オートマトンは、空間における有限の性能を暗示する。しかしながら、この特許のオートマトンは、必ずしも有限であるとは限らず、非有限であり得るが、依然として単純であり得る。したがって、この特許に記載されているＤＦＡは、非有限であってもよい。
【０１５６】
第１に、多ラムダの構成要素-個々のラムダパターン-は、離接の要素として考えなければならない。正規表現を変換する場合、構文a ¦ b (a OR B)は離接であり、一致ａ１１０５または一致ｂ１１１０である。我々の例では、a AND bは各々ラムダパターンである。図１１に従って離接のための部分グラフを作成する。
【０１５７】
我々は個々のパターンのフィールドをまず合接により表わす。正規表現の変換では、構文ａｂ１２１０は合接であり、一致ａ１２０５の後にｂ１２１５が続く。我々の例では、a AND bはパターンの各フィールドである。図１２に従って、合接のための部分グラフを作成する。
【０１５８】
フィールドの反復係数は、従来はa+またはa*またはa{n:m}と書かれた正規表現の閉包と同じである。ここでも、図１３のような構造でこれらの閉包を表現できる。この場合、反復集合の値に基づいて、部分グラフにおけるなんらかの変化が必要になる。たとえば、ノードｉ１３０５からノードｊ１３１０への順方向イプシロン１３１５は、集合が零を含む場合にのみ含まれる。これらの変化は大部分明白であり、ここで説明したのと同じ基本的な考え方に沿って継続する。
【０１５９】
中間のＮＦＡが完結した後、それをＤＦＡに縮小し、次いでそのＤＦＡを最小のＤＦＡに達するまで状態縮小する。次に、ＤＦＡを、状態機械の自動化に使用される通常の種類のソフトウェアまたはハードウェアによる自動化に適した状態－アクション表として表現する。この表の受容状態は、多ラムダへのエントリポイントをマーキングし、中間状態は引数をバインドするために使用されるデータの集まりを提供する。

【０１６０】
ＤＦＡがそのように自動化され、入力ストリームが提供されると、それは、ストリームからの入力の接頭辞にマッチし、正しいオーバーロードにディスパッチしてそれらを処理し、計算結果を生成する。このプロセスが繰り返されることが許されている場合、結果は、入力ストリームからの一致ごとに１つの、生じた結果のシーケンスである。これは、データストリーム内で検出されたさまざまな型の引数の対応するパターンによってトリガされる多相型関数によって、入力データストリームの効率的なリアルタイム処理を提供する。
【０１６１】
値と型識別子との混合を含む複数の種類の関数引数を含むデータストリームに応答して多相型関数の実行をディスパッチする方法であって、（ｉ）実行されるべき多相型関数を識別することを備え、当該多相型関数は、異なる種類の引数のパターンに各々が関連付けられる複数のオーバーロードを有し、前記方法はさらに、（ｉｉ）各オーバーロードについて、オーバーロードの引数パターンをマッチングすることによって、入力ストリームからバインドされる引数値の集合にわたって評価される出力式を識別することと、（ｉｉｉ）各オーバーロードの引数パターンを、入力ストリームにおけるパターンについて一致を効率的に認識するＤＦＡに変換することと、（ｉｖ）個々のオーバーロードのＤＦＡを、全体として多相型関数のための単一のＤＦＡに結合することとを備え、結果として得られる結合されたＤＦＡは、個々のＤＦＡによってマッチングされるであろう任意のパターンをマッチングし、マッチングする入力を処理すべきオーバーロードを選択することができ、前記方法はさらに、（ｖ）結合されたＤＦＡにデータストリームを適用し、ＤＦＡは、次いで、必要に応じてストリームからデータを検査もしくは消費またはその両方を行なって、一致または無一致を判定し、一致の場合には、入力引数値を適切にバインドし、評価されるべき適切な出力式を選択することと、（ｖｉ）出力式の評価をディスパッチし、結果を返すこととを備える。
【０１６２】
リアクティブ関数によって生成される異なる型のデータのストリームのセットが与えられると、本発明は、それらのストリームを、それらの出力が、統合された型の単一のストリームに効率的に構成され得るように、表現する技術である。
【０１６３】
この種の問題を解決する必要性は、すべての形式のデータフロープログラミングで共通して発生する。これは、エンタープライズデータセンター内およびエンタープライズデータセンター間のデータフローなどの非常に大規模なアーキテクチャ、ならびに埋込まれたデバイスにおけるイベントのフローなどの非常に小規模なアーキテクチャに適用可能である。
【０１６４】
本発明は、データフロープログラミングのすべての領域に適用可能であるが、一致を検出でき、ハンドラ関数を適用できる速度が最も重要であり、実行に費やすストレージおよび計算リソースが限られている場合に最適である。
【０１６５】
実施例。ｎ個の別個の入力ストリームのセットからなる流入Ai:0〈k〈nを考える。各ストリームは、型Tiの要素のキューで構成される。各ストリームは、型Ti→Uiのリアクティブ関数fiによって消費および変換されており、型Uiの要素のキューから各々がなる流出のｎ個のストリームBiが存在する。すべてのストリームBiを、Σ Tk→Σ Ukのマージ関数ｍを用いて、単一のストリームCにマージしたい。
【０１６６】
３つのストリームの間で発生するこのようなマージの例をＶｅｌ言語で書いて以下に示す。
【０１６７】

【表４】

B0 = f0 from A0 B1 = f1 from A1 B2 = f2 from A2 C = B0 or B1 or B2

【０１６８】
ストリームCは、B0、B1およびB2からの値から構成され、それらが生成されるときにインターリーブされる。Bストリームの内容を実現することにポイントがないことに注意されたく、なぜならばそれらはCストリームを構成するためだけに使用されるためである。それらは匿名の一時的な部分式として同じく簡単に表現できるであろう。
【０１６９】
【表５】

C = (f0 from A0) or (f1 from A1) or (f2 from A2)

【０１７０】
本発明は、各変換関数fiを決定性有限オートマトン（ＤＦＡ）に変換し、マージ関数mをこれらのＤＦＡの合併として単一の最小ＤＦＡに変換することを記載する。その結果、中間流Biの内容を実現する必要なく流入Aiを流出Cにマージする最大効率的な手段が得られる。
【０１７１】
この技術は、繰り返し適用され、後続の中間流の層を単一のリアクティブ関数に融合させることができる。これは、Ｖｅｌの場合のように、宣言型データフロー言語のインフィックスまたは演算子で表されるマージの概念と整合している。
【０１７２】
この問題はブルートフォースによって解決できる。すなわち、マージ関数が中間流の唯一のコンシューマであっても、中間流を実現し、それらを消費することによって達成される。
【０１７３】
また、マージ関数では、それの流入および流出が、すべて、同じ型のものであるか、そうでなければ、型のないシステムの場合においては未分類のものであることを必要とすることもよくある。これは、それらの型システムにunion型（sum型とも呼ばれる）がないためである。データフローシステムにおける真のマージの存在は、union型の使用を要求する。
【０１７４】
一部のデータフローシステムでは、真のマージを欠き、代わりに複数入力単一出力リアクティブ変換を実施する。これらはそれら自体有用な構成体であるが、真のマージ関数ほど単純でも一般的でもなく、完全に最適化することはできない。
【０１７５】
マッチング関数をＤＦＡとして表現することは、ブール型の任意の式として表現するよりも効率的である。各自の駆動流入を伴う複数のマッチング関数のＤＦＡは、単一の流出を伴うマージ関数を表す単一の効率的なＤＦＡを形成するために統合される。ＤＦＡのマージは、できるだけ早くまたはできるだけ遅く結果がマッチングするように行われ、結果として２つの異なる潜在的に望ましい挙動が生じる。複数の反応を１つのＤＦＡに構成することにより、最小の機械、つまり、最小数の判断を用いてすべての一致を実行するアルゴリズムが得られる。最小の機械は、小規模なプラットフォーム用の複数の反応の最も好適な実現例である。最小の機械は、マッチング式の複数の別々の評価よりもアルゴリズム上の利点を有するため、他のすべてが等しい場合、より効率的に実行される。

【０１７６】
変換ＤＦＡのセットを単一のＤＦＡにマージするには、正規表現で代替（alternation）を考えるように変換ＤＦＡを考慮する必要がある。正規表現を変換する場合、構文a┆bは代替であり：一致a OR 一致bである。我々の例では、a AND bは各々変換関数からのＤＦＡである。図１１に従ってそれらの代替のための部分グラフを作成する。
【０１７７】
中間の非決定性有限オートマトン（ＮＦＡ）が完結した後、それをＤＦＡに縮小し、次いでそのＤＦＡを最小のＤＦＡに達するまで状態縮小する。次に、ＤＦＡを、状態機械の自動化に使用される通常の種類のソフトウェアまたはハードウェアによる自動化に適した状態－アクション表として表現する。この表の受容状態は、マージされたデータ要素が出力ストリームに放出される点をマーキングする。
【０１７８】
ＤＦＡがそのように自動化され、入力ストリームのセットが提供される場合、ＤＦＡは、各入力を、その入力に関連付けられた元の変換関数に従って変換し、すべての結果を１つの出力上でともにインターリーブして与える。
【０１７９】
入力データの複数の独立したストリームを出力データを単一のストリームにマージする方法は、（ｉ）複数の潜在的な入力データストリームを識別することと、（ｉｉ）入力ストリームごとに１つ、各入力ストリームにおけるデータ上で実行され、その結果が一緒になるようマージされる、複数の変換関数を識別することと、（ｉｉｉ）入力データ要素を、複数のストリームから同時に受け取り、データ要素を単一の出力ストリームにインタリーブするマージ関数を識別することと、（ｉｖ）各変換関数を、変換を効率的に実行するＤＦＡに変換することと、（ｖ）変換ＤＦＡを、変換を効率的に実行し、その結果を単一のストリームにインターリーブする単一の結合されたＤＦＡにマージすることと、（ｖｉ）結合されたＤＦＡにデータストリームを適用し、ＤＦＡは次いで変換およびマージの作業を実行することと、（ｖｉｉ）マージされた出力を使用のために宛先にディスパッチすることとを備える。
【０１８０】
本発明は、Ｖｅｌプログラミング言語でソフトウェアを開発するためのツールおよび関連する方法である。Ｖｅｌは、データフロープログラムの表現に役立つプログラミング言語である。正しいデータフロープログラミングには多くの課題がある。いくつかはすべての形式のコンピュータプログラミングに共通する課題であり、他のいくつかはデータフローパラダイムに特有の課題である。このツールは、以下を含む、Ｖｅｌプログラミングの多くの分野に対応している：（１）構文的および意味的な正しさをチェックする。（２）論理的な正しさをチェックする。（３）デバッグ支援。（４）ソースコードを安全かつポータブルな形式（つまり、パッケージ化されたコード）に変換する。（５）さまざまなコンピューティングプラットフォーム、特に小規模なプラットフォームに適した、ネイティブで最適なバイナリ形式へのソースコードまたはパッケージ化されたコードの変換。（６）パッケージ化されたコードの記述およびその署名の確認。（７）パッケージ化されたコードのバッチモード解釈。（８）Ｖｅｌソースのインタラクティブな解釈。（９）パッケージ化されたコードまたはネイティブコードを実行するデータフロー環境のシミュレーション。（１０）ライブデータフロー環境におけるバイナリコードの遠隔実行、監視、および制御。
【０１８１】
これらは、Ｖｅｌ言語でソフトウェアを開発している誰もが達成する必要があるタスクである。本発明は、Ｖｅｌプログラミングに熟練した者が、正確で有用なソフトウェアを作成することを可能にするために、これらの分野すべてにおいて十分なサポートを提供する。
【０１８２】
構文上および意味上の正しさをチェックすることは、多くの形態の自動ソフトウェア変換に共通するタスクである。論理的な正しさをチェックするためのツールは、通常、変換ツ

ール自体には組み込まれていない。これらの種類のツールは別々に存在するのが一般的であり、それらがテストしているコードの不完全な洞察をしばしば伴う。

【０１８３】

デバッグはソフトウェア開発の共通タスクであるが、ほとんどのデバッグツールは命令型プログラミングに重点を置いている。関数型およびリアクティブプログラミングのデバッグは、命令型デバッグとはまったく異なる課題を提示するため、それほど一般的に対応されない。特に、これらの言語では、「フライト中の」計算を検査するのは困難であり得、なぜならば、それらの値はデバッガ（およびデバッガプログラマ）が覗き得るアドレスを有さないことが多いからである。

【０１８４】

複数のネイティブプラットフォームアーキテクチャを対象とする能力は、Ｃなどのシステム言語のコンパイラにはまれではないが、スクリプトレベルの言語では一般的に見られるパワーではない。スクリプト言語は、コンパイルされないか、またはそれらのホストのために部分的にコンパイルされるかまたはジャストインタイムでコンパイルされる（ジットされる(jitted)）傾向があるが、クロスコンパイル（１つのアーキテクチャで実行されるが、別のアーキテクチャに対してコードを生成するコンパイラ）はまれである。小規模なプラットフォーム上で実行するためにスクリプトレベルの言語を具体的にコンパイルすることは非常に珍しいことである。

【０１８５】

対話型シェルは、スクリプト言語の共通の機能である。たとえば、Pythonはシェルを実装している。実際の、またはシミュレートされたデータフロー環境に接続されるシェルは、あまり一般的ではない。

【０１８６】

コンパイルされたコードの遠隔実行は、一部のオペレーティングシステムの機能であり、オープンソースおよび商用の両方のいくつかのサードパーティツールからも利用可能である。これらは、具体的に小規模なプラットフォームを対象としない傾向があるが、小規模なプラットフォーム用の遠隔実行ツールの例が存在する。これらはデータフロープログラミングに特有のものではなく、遠隔で実行されるべきプログラムを開発するのに使用されるツールに組み込まれていない。

【０１８７】

Ｖｅｌコードを開発するための単一の統合されたツールは、Ｖｅｌ言語で作業するソフトウェア開発者にとって有用で便利である。このツールは主にＶｅｌ言語を変換するコンパイラであるが、Ｖｅｌプログラミングに関連する他のいくつかの関数のセットも提供する。ツールで論理的な正しさのテストを構文的および意味的な正しさのテストとともに実行すると、開発者はより効率的になり、コードの正確性を高めることができる。論理テストでは、コンパイラによるコードの洞察の恩恵があり、診断メッセージはより完全なものになり得る。対話型シェルにより、開発者はコードをテストして即座の応答を得ることができる。これは開発やデバッグに便利である。シェルはまた、プログラマがデータフロー環境を視認できるようにする。

【０１８８】

小規模なプラットフォームでの使用に適したスタンドアロンのバイナリ実行可能コードを生成することは、さまざまな小型デバイスで複雑な計算を実行することにしばしば依存する、物のインターネット使用事例を可能にする。シミュレートされたデータフロー環境を提供することで、開発者はコードにおけるバグを解消し、論理的な正しさに対するテストと連携して、パッケージが正しく動作していることを実証できる。コンパイルされたパッケージを遠隔で実行する場合、特に遠隔プラットフォームが小さい場合、プログラマは、自分のプログラム上で迅速に、プログラムをそれのターゲットハードウェア上で１つのコマンドでコンパイルしテストすることを、たとえターゲットプラットフォームが、自分がその上で開発しているものでなくても、反復できる。

【０１８９】

言語をそれの語彙的表現からこの中間の記号的表現に変換（フェーズ１コンパイル）し、次いでこの中間表現を計算ハードウェアによって実行され得る形式に変換（フェーズ２コンパイル）するプロセス。

【０１９０】

Ｖｅｌフェーズ１変換ツールは、コンパイラに共通する一般的な戦略に従い、具体的には次のとおりである。（１）入力文字列を解析してそれをトークンのシーケンスに分解する。（２）トークンのシーケンスを解析して構文木を形成する。（３）構文木内の記号宣言を識別する。（４）構文木内の記号参照の特定および解決。（５）共通の部分式の削除や定数の折り畳みなどの初期の最適化。（６）型チェック。（７）最適化および記号成熟の追加フェーズ。（８）記号の最終決定および中間表現の放出。

【０１９１】

Ｖｅｌフェーズ１トランスレータの際立った特徴の１つは、それが、決定性有限オートマトンまたはＤＦＡを用いて、関数適用に必要なパターンマッチングを実行し、反応をトリガすることである。フェーズ１変換ツールは以下を含む。（１）入力言語を構文木に変換する構文解析器。（２）解析中の言語がＤＳＬまたはマクロ解析器の態様で解析器によって修正され得るように、変換中のプログラムが自己参照を行うことを可能にする語彙バインドコンポーネント。（３）バインドされた構文木を、データフロー、パターン、反応、関数式、タイマー、および入出力パラメータ化を表す記号に変換する意味解析アルゴリズム。（４）式木を、多かれ少なかれマイクロプロセッサＡＬＵ命令への直接変換に適したスタックに変換する式トランスレータ。（５）反応のパターンおよび式を潜在的に非最小のＤＦＡの中間収集物に変換するためのＤＦＡジェネレータ。（６）ＤＦＡの中間収集物から統合された最小のＤＦＡを作成するためのＤＦＡ結合および縮小アルゴリズム。

【０１９２】

フェーズ１変換ツールの出力は以下を含む：（１）変換に関与する各ストリームの論理的アイデンティティ。各々が複数のストリームの中で一意的な参照であり得るようにする。（２）各ストリームにおけるデータにおけるフローの記述。各々、内方向（反応に向かう、すなわち外部ソースへのサブスクリプション）、外方向（反応から離れる、つまり外部の宛先へのパブリケーション）、内方向および外方向の両方（パブリケーション／サブスクリプションのペア）、または内部（他の反応における中間ステップとしてのみ使用されるため、パブリケーションまたはサブスクリプションとして表面化されない）である。（３）各ストリームにおいて流れるデータの型の記述。ストリームに注入またはストリームから抽出されるデータが型の正しさを静的にチェックされるように、都度有限項で記述される。（４）ＤＦＡの状態および遷移を記述する表のセット。（５）反応の間に実行されるべき計算を記述する式スタックのセット。（６）ストリーム入力をＤＦＡ入力にマッピングする表。（７）時限イベント（timed event）をＤＦＡ入力にマッピングする表。（８）ＤＦＡ出力をアクションのペアにマッピングする表。各ペアは、式スタックへの参照およびストリーム出力から構成され、ＤＦＡの出力が所与の式で変換され、次いで所与のストリームにプッシュされることを示す。

【０１９３】

Ｖｅｌインタプリタおよびデータフローシミュレータはフェーズ１変換の出力を直接使用する。インタプリタはコードの実行においてハードウェアプラットフォームをエミュレートし、データフローシミュレータはストリーミングデータ環境をエミュレートし、Ｖｅｌストリームに入力を提供し、Ｖｅｌストリームから出力を収集する。これらの２つのタスクを命令解釈およびデータフローエミュレーションと呼ぼう。

【０１９４】

命令解釈は、コンパイラおよびインタプリタの作成を専門とするコンピュータプログラマがよく理解するタスクのカテゴリである。タスクには、実行時変数の状態を格納できる実行コンテキストを構築し、プログラムの命令を一度に１つずつ実行し、実行コンテキストからデータにアクセスし、必要に応じてそれを更新することが含まれる。

【０１９５】

Ｖｅｌの場合、実行コンテキストには、変換の過程でデータのストリームを保持するキューのセット、およびＤＦＡで記述された変換を実行する表駆動型の状態機械エンジンも含まれている必要がある。キューは、データの流通チャネルを記述する、Ｖｅｌソースにおける宣言のため、発生する。これらのうちのいくつかは、Ｖｅｌプログラムの外部入力または出力であり、他のいくつかは入力と出力との間の中間状態を記述する純粋な内部チャネルである。

【０１９６】

データフローエミュレーションは、ファイルやソケットなどのデータのために外部ソースおよびシンクへのアクセスを提供することと、これらの外部システムと解釈されているＶｅｌプログラムとの間でデータを交換するために必要なプログラミングとからなる。これには、外部ソースからデータを読み出してそれらデータをプログラムの入力を表すキューにプッシュする注入関数と、プログラム出力を表すキューからデータをポップしてそれらデータを外部シンクに書き込む抽出関数とが含まれる。

【０１９７】

本発明によるＶｅｌの解釈が正規とは異なるところでは、ＤＦＡが関与するようになる態様である。状態機械エンジンは、キューからデータを読み取り、それらデータを使用してそれらのＤＦＡの状態を前進させる。ＤＦＡ表には、ＤＦＡがそれらの状態を通過するときに実行される副作用の列が含まれる。これらの副作用は、命令解釈を呼び出して計算を実行し、その結果は他のキューにプッシュされ、これが他のＤＦＡをトリガする。

【０１９８】

このようにして、本発明による解釈中のＶｅｌプログラムは、最初に、高速かつ小規模な状態機械のセットによって表され、必要なときに一般的な命令解釈にドロップバックするだけである。これにより、命令解釈だけですべて処理される場合よりも、プログラムの実行効率が向上する。

【０１９９】

Ｖｅｌフェーズ２変換ツールは、大部分、Ｖｅｌ言語に固有ではなく、実行対象のプラットフォームに固有である。フェーズ２トランスレータのＶｅｌ言語関連コンポーネントは次のとおりである。（１）フェーズ１によって生成された中間表現の初期取入れ。（２）反応性システムを生成するフェーズ２コード生成の全体的編成。（３）データフォーマットの外部エンコードおよびデコード、またはリアルタイムクロックの内部調整を実行するもののような、ランタイムサポートコンポーネントのライブラリの提供。

【０２００】

マルチソース・マルチ宛先データフロー環境におけるデータストリームのリアルタイム処理のためにプログラムを作成するためのツールは、以下を含む。（１）複数の潜在的なデータストリームを識別すること。（２）ストリームにおけるデータのパターンに対応するリアクティブ関数およびパラメータのセットを識別すること。（３）宣言されたパターンにマッチするデータを変換するための処理関数およびパラメータのセットを識別すること。（４）データが収集または破棄される時間の間隔や、データが収集または破棄される前後の特定の時点など、データフローのパターンが比較される時限イベントのセットを識別すること。（５）識別されたストリーム、反応、関数、および時限イベントを記述するデータフロープログラムを作成すること。（６）このプログラムを、Ｖｅｌプログラム文を対応するＤＦＡに変換するためのＤＦＡジェネレータを組み込んだフェーズ１変換ツールと、プラットフォーム上での実行のために変換されたＶｅｌ文に対応するプラットフォーム固有のハードウェア命令を生成するためのフェーズ２変換ツールとを備える２フェーズ変換ツールに、入力として与えること。（７）変換ツールの各フェーズの出力を受け取ること。

【０２０１】

フェーズ１変換ツールの出力は、以下を含むインタプリタコンポーネントによって使用されてもよい：（１）コードの実行においてハードウェアプラットフォームをエミュレートする命令インタプリタ。（２）ストリーミングデータ環境をエミュレートし、Ｖｅｌスト

リームに入力を提供し、Ｖｅｌストリームから出力を収集するデータフローシミュレータ。
【０２０２】
フェーズ１変換ツールの出力は、以下を含むフェーズ２変換ツールへの入力として使用することができる：（１）中間表現からの命令をターゲットハードウェアプラットフォームによる実行に適した形式に変換するハードウェア命令ジェネレータ。（２）データフロー環境におけるリアクティブプログラムとしての使用に適した形式に出力の生成を向けるプログラム編成モジュール。（３）実行に必要なランタイムサポートコンポーネントのライブラリ。フェーズ２変換ツールの出力は、対象とされるハードウェアプラットフォーム上での使用に適した実行可能なプログラムである。
【０２０３】
さまざまな型のデータのストリームのセットと、それらのストリームにおけるデータの特定のパターンに反応しそれら特定のパターンを処理することを意図した関数のセットが与えられると、本発明は、データがストリームで到着するとこれらの関数が適切かつ効率的に呼び出され得るようにこれらの関数を記述および変換する技術である。
【０２０４】
この種の問題を解決する必要性は、すべての形式のデータフロープログラミングで共通して発生する。これは、エンタープライズデータセンター内およびエンタープライズデータセンター間のデータフローなどの非常に大規模なアーキテクチャ、ならびに埋込まれたデバイスにおけるイベントのフローなどの非常に小規模なアーキテクチャに適用可能である。
【０２０５】
本発明は、データフロープログラミングのすべての領域に適用可能であるが、一致を検出でき、ハンドラ関数を適用できる速度が最も重要であり、実行に費やすストレージおよび計算リソースが限られている場合に最適である。
【０２０６】
実施例。たとえば、所与の整数のストリームから、１つ以上の非零値に１つ以上の零が続くものをマッチングしたい、とする。このパターンがマッチすると、非零値の和を計算し、その結果を別のストリームに書き込みたい。
【０２０７】
この問題のパターンマッチング部分を正規表現のような表記法で記述し、その和の計算を別途算術表現として書くことができる。それが起こると、エッジコンピューティングにおいてデータフローアプリケーションで使用するために設計されたＶｅｌプログラミング言語を使用することにより、変換全体を統一された表記法で書くことができ、したがって、
【０２０８】
【表６】

```
1. stream foo is int
2. p = pn a:{!= 0}.., :0.. -> sum(a)
3. bar = p(foo)
```

【０２０９】
１行目では、fooが整数のストリームであると宣言する。２行目では、pを非零値に１つ以上の零が続くものにマッチするパターンとして定義し、非零値の和を計算する。３行目では、pをfooに適用して新たなストリームバーを定義し、それに適用の結果がプッシュされる。
【０２１０】
この技術は、上記のパターンを状態機械に変換するであろう。次に、それは、その一致をその状態機械に基づいて決定性プッシュダウンオートマトンとして実現し、結果の一致を合計式に供給するであろう。
【０２１１】

上記の例で示した種類の問題は、２つの関数、つまりマッチングのための関数および一致の発見後に結果を計算するための関数を生成することで解決できる。マッチング関数は、ストリームからデータのウィンドウを入力として受け入れ、一致の場合は真を返し、不一致の場合は偽を返す。一致が見つかると、データウィンドウは結果計算関数に渡され、出力が生成される。

【０２１２】

データがウィンドウを通って流れると、一致が見つかるまで、マッチング関数を繰り返し適用する必要がある。従って、効率は、マッチング関数を実行できる速度に制限される。特定の呼び出しにおいてマッチング関数によって計算されたどのような値も、一般的に、後の呼び出しでは再利用されもしなければ、一致が検出されたときに結果計算関数による使用について利用可能でもない。おそらく有用である中間計算の結果の保持の欠如は、非効率性の潜在的原因である。

【０２１３】

ストリーム処理がストリーミングＳＱＬのような言語で指定されている場合は、この２関数構成が単純になる。ストリーミングSELECTステートメントのwhere文節は、一致条件を記述するブール式を提供し、マッチング関数はこの式の直接コンパイルである。SELECTのproduct文節で名前を付けられた値のタプルは、次いで、出力生成関数を定義するための基礎を提供する。

【０２１４】

状態機械を使用して一致を実行する方が、複数の任意のブール式を繰り返し適用するよりも、効率的である。

【０２１５】

本発明は、関数のパラメータを宣言するパターン記述言語から状態機械を導出する。導出された状態機械は、データストリーム内において、一致を、従来のブール式マッチング関数よりも、より効率的に検出する。

【０２１６】

導出された状態機械はまた、データストリームにおいて検出された一致に基づいて出力を生成するよう、ハンドラ関数のセットを実装してもよい。複数のマッチング関数および対応するハンドラ関数を組み合わせて、単一の状態機械に縮小し、それによって、多数のパターンについて同時に一致を効率的に認識し、多くの種類の出力を同時に生成してもよい。

【０２１７】

正規表現および値式の局面を組み合わせるスクリプトは、重要な中間計算の結果を保持することによってパターンを効率的にマッチングして出力を計算できる決定性状態機械に自動的に変換できる。結果として生じる組み合わされたマッチングまたは計算アルゴリズムは、パターンマッチングおよび出力生成の別個の編成よりも効率的である。

【０２１８】

したがって、全体のアプローチは次のようになる。（１）正規表現のような表記法を使用して一致を特定し、関数式のような表記法を使用してこれらの一致に基づいて出力計算を特定するソーススクリプトから開始する。（２）スクリプトの正規表現のような部分を文法規則のセットに変換し、スクリプトの関数式のような部分をそれら文法規則の属性のセットに変換する。（３）文法および属性から状態機械を生成する。（４）状態機械を、マッチングされるべき入力が流入し、その結果が流出するプッシュダウンオートマトンとして実装する。

【０２１９】

ある技術は、時限順方向右端（timed forward right-most）（１）パターンマッチングまたはＴＦＲ（１）と呼ばれる。以下の議論では、ＴＦＲ（１）パターンマッチング状態機械を規定し、それを適用してステップ４を達成する方法を詳述する。以下のセクションは、ステップ３の目標に従って文法をＴＦＲ（１）状態機械に変換する（すなわち、文法および属性から状態機械を生成する）方法を説明する。

【０２２０】

ソーススクリプトから文法規則および属性のセットに変換する具体的な方法は、必然的にソーススクリプト言語自体に依存し、したがって、本文書の範囲外である。しかし、文法規則および属性の使用は、コンピュータ言語実装の中では一般的な手法であり、したがって、ステップ２は、実装者にとって大きなハードルを呈しはしないと仮定することは妥当である。

【０２２１】

ＴＦＲ（ｌ）パターンマッチング

このセクションでは、文脈自由文法の大きなクラスに基づいてストリーミング入力のパターンを効率的に認識するために使用できる技術であるＴＦＲ（１）パターンマッチングを提示する。この省略形は、「時限順方向右端（Timed Forward Right-most）（１）」技術を表し、この技術の持つ時間を認識する性質、一度に１つの入力トークンだけを見ながら、（時間に関して）順方向の順序で入力を走査し、最も右の構文的派生を生成するという事実を指す。

【０２２２】

概要。パターンマッチャー（以下、単に「マッチャー」）の目的は、出力のストリームを生成することによって入力のストリームに反応することである。予想される入力のパターンが事前に記述され、そのパターンにマッチするようにマッチャーが予め構築される。入力パターンにマッチした後、マッチャーは、マッチした入力に基づいて１つ以上の出力を生成する。

【０２２３】

ＴＦＲ（１）マッチャーには、いくつかの特筆すべき特徴がある。（１）マッチャーは、マッチングされるべきパターンおよび適用されるべき変換を記述する文法のような規則のセットからアルゴリズムを介して構築できる。（２）マッチャーは、マッチング中にタイムアウトという概念を組み込むことができ、時間の経過により一致を進ませる。（３）マッチャーは、任意の明確な文脈自由文法からアルゴリズムを介して構築することができる。（４）マッチャーは、コンパクトで効率的であり、限られた記憶域もしくはローエンドプロセッサーまたはその両方を伴うコンピュータでの実装に適している。入力をマッチングするには最小限のステップ数が必要である。（５）マッチャーはバックトラッキングなしで動作し、入力内で厳密に前方に移動し、それを、流れるストリームの処理に適するものにする。（６）マッチャーは、不一致（誤った）入力の検出が可能になるとすぐにそのようにし、それらを通過して早送りし、入力ストリームと再同期し、マッチングを再開することができる。（７）マッチャーは、入力シーケンスをマッチングした後、マッチングするデータを任意の関数への引数として使用することができ、その結果は出力として生成される。

【０２２４】

時間および無限を伴う構文。ストリームで流れるデータは、構文的に解析することができる。このアプローチは、コンピュータソースコードの構文解析に使用される種類の構文解析に似ている。実際、ＴＦＲ（１）は正規のＬＲ（１）構文解析アルゴリズムに似ており、正規のＬＲ（１）および構文解析に概して精通していることは、ＴＦＲ（１）に興味のある読者にとって役に立つ。

【０２２５】

ストリームは、ファイルとは２つの点で異なる。（１）ストリームにおける各データ要素は、それに対して（暗示的または明示的に）それがストリームに入った特定の時刻が関連付けられ、一方ファイルにおけるデータ要素は、どのような特定の時間もそれとは関連付けられない。（２）ストリームは明示的な終端を有する必要はなく、無限に流れ続けることができるが、ファイルは固定されたサイズを有するので、その終端は常に期待され得る。

【０２２６】

ＬＲ（１）は、他の大抵の構文解析アルゴリズムと同様に、ファイルに対して動作することを意図される。それは時間を考慮せず、正しい動作のためにファイル終端マーカーに依存する。しかしながら、ＴＦＲ（１）はストリーミング入力との使用を意図される。それ

は時間を意識しており、ファイル終端マーカーを必要としない。
【０２２７】
入力
入力キューは、一度に１つずつ処理されるべき離散的な入来イベントの、先入れ先出しの、非終端シーケンスである。これは、ネットワークソケット、シリアルポート、インメモリ構造、またはエンドレスキューもしくはストリームの概念の任意の他の有形もしくは抽象的実現例であってもよい。マッチャーの目的は、このストリームにおいてパターンを認識することである。
【０２２８】
各々は、トークンがキューに入った時刻をマーキングするスタンプ、および場合によってはその他の何らかの情報を担持する。イベントには以下の３つの種類：トークン、経過、およびブレークがある。
【０２２９】
トークンは、解析されるべきデータを担持するイベントである。これは最も一般的な種類のイベントであり、「イベント」および「トークン」という用語は往々にして相互交換可能に使用される。トークンによって担持されるデータ要素は、任意の強力なデータ型のインスタンスであることができ、特定の入力キューにおけるすべてのトークンは、同じ型のデータを担持する。トークンは、通常、（マッチャーがワイヤフォーマットをデコードしているかまたは語を認識しているときは）シングルバイトまたはシングル文字を、または（マッチャーがセンサから多次元データを解析しているときは）多値のデータレコードを、担持する。
【０２３０】
場合によっては、トークン型の可能な状態の数は非常に多い。これは、可能な状態が２５６しかない８ビットのバイトには当てはまらないが、３２ビットの整数または多くのフィールドで構成されたレコードなど、より大きな型の場合には当てはまる。これらの場合、しばしばトークン分類関数を提供する必要がある。トークンが与えられると、分類関数はトークンが属するクラスを返す。たとえば、あるパターンには３２ビット整数の２つのクラス（零および非零）が含まれる。ドライバは、実行するアクションを決定するときに、トークン自体の代わりにトークンのクラスを使用する。
【０２３１】
経過は、それのスタンプの他はデータを担持しないイベントである。それは、本質的には、時間の経過以外は何も示さない、「非イベント」である。この種のイベントを入力キューに含めることは有用であり、なぜならば、それは、たとえ報告すべきデータがなくてもイベントを積極的に受信している入力キューと、何も受信していない入力キューとを区別するからである。
【０２３２】
ブレークは、入力キューの瞬間的な中断を示すイベントである。マッチャーは、作業中の部分一致をどのようなものであろうとも済ませ、すぐにその出力を報告することによって、ブレークに応答する。ブレークは、入力の終わりのマーカーではなく、追加のイベントはブレークに続くことができ（および一般的にはそうす）る。それは、単に、入力ストリームに任意で注入できるイベントであり、マッチャーのアクションを加速する。
【０２３３】
規則。我々は文法を一連の規則として書く。各規則では、左にリダクションが現れ、右に記号のシーケンスが現れる。各記号はトークンまたはリダクションのいずれかである。たとえば、
【０２３４】

【表７】

(r1) E → E ‘+’ E
(r2) E → num

【０２３５】
これらの２つの規則は、トークンnumおよび“＋”ならびにリダクションEからなる言語を記述する。r1によって、入力にnumが現れるたびに、それはEとみなされてもよい。r2によって、入力にE + Eのシーケンスを見るたびに、それら３つの記号をまとめて新たなEを構成してもよい。したがって、シーケンスnum、num + num、およびnum + num + numは、すべて、当該言語の有効なセンテンスである。
【０２３６】
規則が空である可能性もあり、それは、規則の右辺に記号がないことを意味する。これは、特定の構文片をオプションと見なす場合に便利である。たとえば、
【０２３７】
【表８】

(r3) L → L ‘x’
(r4) L → λ

【０２３８】
我々は空を表すために小文字のラムダを右辺に書く。この場合、リダクションLは、それ自体に“ｘ”が続くもの（r3による）からなり、そうでなければ、それは無である（r4による）。したがって、シーケンスx、ｘｘ、およびｘｘｘは、すべて、当該言語の有効なセンテンスである。空のシーケンスも、当該言語の有効なセンテンスである。Lがそれにマッチするであろうからである。空の規則でタイムアウトを指定することもできる。たとえば、
【０２３９】
【表９】

(r5) C → C ‘x’
(r6) C → τ5

【０２４０】
この言語のCリダクションは、上記の言語のLリダクションと非常によく似ているが、Lはｘを無限に収集するが、Cは限られた時間量だけxを収集する。ここでの時間量は５単位として与えられる。この単位は、任意の都合のよい時間の尺度とすることができる。実際には、タイムアウトはミリ秒単位で与えられる可能性がある。ここでその違いを理解するために、タイムスタンプ付きの入力のシーケンスを考えてみる。
【０２４１】
【表１０】

1000	‘x’
1001	‘x’
1002	‘x’
1003–1009	no input
1010	‘y’

【０２４２】

ここでは、３つのxが速やかに連続して到着し、次に切れ間があり、次に「y」が到着する。r3およびr4を使用してこの入力と照合すると、「y」の出現はxの文字列の終わりを示すであろうから、マッチャーはLを時間1010で報告するであろう。しかしながら、r5およびr6を使用すると、時間の経過だけでリダクションを呼び出すのに十分であろうから、マッチャーはCを時刻1005で報告するであろう。

【０２４３】

直感的に、ラムダルール（タイムアウトのない空の規則）は、何をすべきかを決定する前に次のトークンを見ることができるまで待つことになり、それは、ラムダルールが無限に待つことを意味する。タウルール（タイムアウト付きの空の規則）も次のトークンを待つが、そのトークンが充分にすぐに来ない場合、タウルールはそのトークンが存在しないなかで行動することを決定する。

【０２４４】

属性。属性は、文法規則のセットにわたって定義された関数である。マッチャーがある規則を適用してリダクションを実行すると、それはその規則に対する属性関数も適用する。結果は、さらなる属性を計算するため、もしくはマッチャーの出力になるため、またはその両方に使用できる値である。このプロセスを属性合成と呼ぶ。上記のサンプル言語の１つをもう一度見てみよう。

【０２４５】

【表１１】

```
(r1) E → a:E '+' b:E
(r2) E → a:num
```

【０２４６】

ここで認識されている言語は先の例から変わっていないが、ただし、明示的な名前が記号に追加されている。我々の属性の定義において、これらの名前を使用できる。

【０２４７】

【表１２】

```
int val   {
          r1: a,
          r2: val(a) + val(b)
          }
```

【０２４８】

ここでは、我々は、型intの値を生成する属性valを定義している。r1が適用されると、valの値は単純に数aであるよう定義され、これは入力ストリームから取られたリテラル番号である。r2が適用されると、valの値は帰納的に定義され、aおよびbのvalを評価し、結果を合計する。記号ａおよびｂはEのインスタンスである。

【０２４９】

直感的には、属性合成は、個々のトークン（またはトークン自体だけ）から導出され得る値で始まる。マッチャーが規則を適用してリダクションを生成するにつれて、合成はより大きな範囲の操作に向かって進行する。ハイレベルのリダクションの属性は、それがまたがるトークンのいずれかまたはすべてに直接的または間接的に依存する。

【０２５０】

拡張された文法および再起動。拡張された文法は、厳密に１つのオメガルールを含む文法である。オメガルールは通常の文法規則と似ているが、ただし、右に厳密に１つの記号があり、左に特殊なオメガ記号があることを除く。たとえば、

ここで、オメガルールであるr0を追加して、先の例からの文法を拡張した。

【０２５１】
【表１３】

(r0) Ω → E
(r1) E → E ‘+’ E
(r2) E → num

【０２５２】
ある文法のオメガルールは、マッチャーに対して、どこから始まり、どこで終わるかを教える。最初に起動されると、マッチャーはオメガルールの右側にある記号をマッチングしようとする。この場合、r0は、マッチャーは最初にリダクションEをマッチングしようとすることになる、と言う。
【０２５３】
オメガルールの右側の記号をマッチングした後（またはそれをマッチングすることに失敗したことから回復した後）、マッチャーは自動的に再起動し、その記号を再びマッチングしようと試みる。したがって、マッチャーはオメガルールを無期限にマッチングし続ける。
【０２５４】
入力キューにおけるブレークイベントは、マッチャーに、現在待ち状態かもしれないラムダルールまたはタウルールを受け入れさせ、すぐにオメガルールに縮小するよう進ませる。その後、マッチャーは再起動してブレーク後のイベントを受け入れる。
【０２５５】
オメガルールは、厳密に１つの属性の定義において現れなければならない。この属性は、次のセクションで説明するように、マッチャーの出力を計算するために使用される。
【０２５６】
出力。出力キューは、離散的な出力イベントの、先入れ先出しの非終端シーケンスである。出力キューは概念的には入力キューと似ているが、マッチャーに向かう代わりに、マッチャーから離れる方向に流れる。入力キューと同様に、出力キューは、ネットワークソケット、シリアルポート、インメモリ構造、またはエンドレスキューもしくはストリームの概念の任意の他の有形もしくは抽象的実現例であってもよい。マッチャーの目的は、このキューにイベントを送信することである。マッチャーは出力キューからの読出を決して試みない。
【０２５７】
出力キューの各イベントには、マッチャーがイベントを生成した瞬間をマーキングするスタンプが含まれている。各イベントはデータ要素も担持し、それは、任意の強力なデータ型のインスタンスであることができ、特定の出力キューにおけるすべてのイベントは、同じ型のデータを担持する。これは、入力キューにおけるイベントと同じ型でも、異なる型でもよい。
【０２５８】
マッチャーがオメガルールによって縮小すると、マッチャーはオメガルールの（唯一の）属性を評価し、結果をイベントとして出力キューにプッシュする。このように生成された出力イベントは、最新の入力イベントのスタンプと等しいスタンプを有することになる。
【０２５９】
アーキテクチャ。図１４は、ＴＦＲ（１）パターンマッチャーのコンポーネントを示す。各ＴＦＲ（１）マッチャーは、入力キュー１４０５、出力キュー１４１２、ドライバプログラム１４１９、状態スタック１４２６、および状態表１４３３からなる。
【０２６０】
マッチャーは、入力キューの先頭に読出位置を保持する。それは、キューの前から一度に１つずつイベントをポップする。マッチャーは入力キュー内を先読みすることも、読出位置をキューにおける前のポイントに巻き戻そうとすることも、入力キューにイベントをプッシュしようとすることもしない。この１度に１つの、順方向のみの、読出のみの、入力

の処理は、ＴＦＲ（１）を、リアルタイムストリームの効率的な処理に適したものにする。
【０２６１】
ドライバプログラムはすべてのマッチャーに対して同じである。それは、別の表で定義されている状態によって駆動されるプッシュダウンオートマトンである。ドライバプログラムに対するあるアルゴリズムを以下に示す。ドライバプログラムの各実装は、それのホストハードウェア、オペレーティングシステムプラットフォーム、およびその環境の他の詳細に特有のものである。しかしながら、ドライバはマッチングされるべきパターンに固有ではない。ドライバプログラムは常に同じであり、マッチングされるべきパターンに固有の状態表である。
【０２６２】
ドライバは、状態スタックを用いて、状態表によって定義された状態を介してそれの進行状況を追跡する。スタック上の最初のエントリは、常に、表における最初の状態への参照で構成される。スタック上の各後続のエントリは、表におけるある状態への参照、その状態を解決しなければならないデッドライン、および入力キューから取ったトークンまたは文法規則の左辺から取ったリダクションからなる。
【０２６３】
状態スタック上のリダクションは、それらに対して、それらの規則から計算され得る属性が関連付けられる。新たなリダクションを適用する準備が整うと、スタックの一番上の状態は、その新たなリダクションに関連付けられた属性定義関数を適用するために使用される呼び出しフレームを提供する。このように、状態スタックは関数呼出スタックでもある。これはまた、中間計算の結果が後で使用するために自動的に保持されるが、（スタックがポップされたときに）それらがもはや潜在的な関心がなくなったときには破棄されることも意味する。
【０２６４】
状態表は、マッチングされるべきパターンおよび生成されるべき出力を記述する文法規則ならびに属性からアルゴリズムを介して生成される。次に、表の生成方法について説明する。一旦生成されると、状態表は変更されない。ドライバプログラムは表によって導かれるが、それは表を変更しない。実際、ドライバプログラムのいくつかのインスタンスの場合、同じ状態表を共有することが可能である。これは、複数の入力ストリームで同じパターンをマッチングし、複数の出力ストリームを生成したい場合に便利な構成であろう。
【０２６５】
状態表は有限数の行で構成され、各行は構文解析状態を定義する。構文解析状態は、文法駆動型パターンマッチングの意思決定プロセスにおいて明確なポイントを定義する抽象的な概念である。各構文解析状態は、アクション、遷移、タイムアウト値の３つの部分から構成される。
【０２６６】
構文解析状態のアクションは、各トークン値を、ドライバがとるべきアクションにマッピングする。ドライバは、表のこの部分をそれのメインループで使用する。表のこのセクションは、文法のトークンまたはトークンがグループ化されるクラスによってキーイングされる（keyed）。ブレークイベントも、表のこのセクションにおけるキーである。
【０２６７】
構文解析状態の遷移は、各種類の記号を構文解析状態にマッピングする。ドライバは、縮小アクションを実行するときに表の一部を使用する。表のこのセクションは、文法のリダクションによってキーイングされる（keyed）。
【０２６８】
構文解析状態のタイムアウトは、ドライバがその状態に留まれる最大時間量を示す。これは、明示的に無限であり得（一般的には明示的に無限である）、ドライバがその状態で任意の時間量を費やし得ることを意味する。
【０２６９】
ドライバアルゴリズム。ドライバは、単一のエントリを含むスタックで開始する。このエ

ントリは開始状態（常にｓ０と示される）を指し、開始時間にｓ０のタイムアウトを加えたものに等しいデッドラインを有する。この最初のスタックエントリには関連付けられる記号はないが、後のエントリは各々トークンまたはリダクションのいずれかを担持する。

【０２７０】

ドライバは次いでそれのメインループに入り、それは無期限に反復する。各反復の開始時に、ドライバはスタックの一番上のエントリおよび入力キューの前部分のイベントを見る。

【０２７１】

ドライバは、現在のイベントのスタンプと現在のスタックエントリのデッドラインを比較する。スタンプがデッドライン以上の場合、状態は期限が切れ、ドライバはすぐにその状態を離れるように試みなければならない。期限切れ状態の間、ドライバは実際のイベントの種類にかかわらず、現在の入力をブレークとして処理することになる。

【０２７２】

現在のイベントが経過であり（現在の状態が期限切れでない）場合、ドライバは単にそれを破棄して（下記参照）、メインループの先頭に戻る。経過は、状態を期限切れとすること以外の目的を果たさない。

【０２７３】

現在のイベントがトークンまたはブレークの場合（または現在の状態が期限切れで、現在のイベントをブレークとして処理している場合）、ドライバは現在の状態のアクション部分においてとるべきアクションを参照する。アクションは、入力キューを次のトークンに進めるか、スタックをプッシュもしくはポップするか、出力キューにプッシュするか、またはこれらを組み合わせる。アクションを実行した後、ドライバはメインループの先頭に戻る。

【０２７４】

ドライバが実行できるアクションには５種類：シフト、破棄、縮小、再起動、またはパニックがある。各反復において、ドライバはこれらのうちの厳密に１つを実行することになる。

【０２７５】

シフトアクションは状態への参照からなる。このアクションは、ドライバに、入力キューから現在のイベントをポップさせ、入力キューを次のイベントに進めさせる。イベントはトークンでなければならず、ドライバは経過またはブレークをシフトしない。ドライバは、次いで、アクションによって示された状態、新たなデッドライン、およびポップされたトークンからなる新たなエントリをスタックにプッシュする。新たなデッドラインは、現在のデッドライン、またはイベントのスタンプ＋状態のタイムアウトのうち、より小さいものと等しい。

【０２７６】

破棄アクションは、ドライバに、入力キューから現在のイベントをポップさせ、入力キューを次のイベントに進めさせる。ドライバは同じ状態のままである。これは経過および本当のブレークに起こることである。

【０２７７】

縮小アクションは、文法規則への参照で構成される。縮小アクションは、合成、スタックからのポップ、スタックへのプッシュという３つのフェーズで行われる。

【０２７８】

合成する。ドライバは、適用されている規則の属性を評価する。

スタックからポップする。ドライバはスタックから１つ以上のエントリをポップする。ポップするエントリの数は、適用される規則の右辺の長さによって与えられる。ポップ後のスタックの一番上における状態をPと呼び、適用された規則の左辺側の記号をLと呼ぶ。

【０２７９】

スタックにプッシュする。ドライバは状態Pの遷移において記号Lを参照し、新たな状態Nを与える。次に、ドライバは、状態N、新たなデッドライン、および記号Lからなる新たなエントリをスタックにプッシュする。

【０２８０】
再起動アクションは、合成、スタックリセット、ブレーク破棄の３つのフェーズで実行される。
【０２８１】
合成。ドライバは、オメガルールの属性を評価し、結果を出力キューにプッシュする。
スタックリセット。ドライバはスタックからすべてのエントリをクリアし、次いで、新たな開始エントリをプッシュする。新たなエントリは、ｓ０への参照と、入力イベントのスタンプ＋ｓ０のタイムアウトに等しいデッドラインとから構成される。エントリには記号は含まれない。
【０２８２】
ブレーク破棄。現在の入力が本当のブレークであり（そして現在の状態が期限切れになったために単にそれとして処理されているのではない）場合、ドライバはそれをそこで破棄する。
【０２８３】
パニックアクションは、ドライバに、再同期化できるまで入力を破棄させ、ついで、スタックを再起動させる。
【０２８４】
構文解析
直感的には、マッチャーは、パターンマッチングのタスクを、パターンのピースを各１つがマッチングする一連のサブタスクに分割する。これらのタスクの各々は、次いで分解され、このプロセスは、マッチングされるべきピースが単純に、自明にマッチされ得る入力トークンそれら自体になるまで繰り返される。
【０２８５】
マッチング可能なピースにパターンを分解するタスクはマッチャージェネレータによって行われ、結果は状態表の状態においてエンコードされる。ドライバはこれらの状態に従って、マッチングサブタスクを実行する。
【０２８６】
ドライバがシフトアクションをとるとき、それは、入力トークンがドライバの現在のマッチングサブタスクの一部であるからである。ドライバは、一致の現在のピースを、トークンを含むように拡張している。
【０２８７】
ドライバが縮小アクションを実行するとき、それは、入力トークンがドライバの現在のマッチングサブタスクの一部ではないからである。ドライバは一致の現在のピースの終わりをマーキングし、入力トークンが最初の部分になる次のピースに移動している。
【０２８８】
ドライバがパニックアクションをとるとき、それは、入力トークンがマッチングサブタスクの一部ではなく、そのポイントまでの全マッチング状態が誤っているからである。ドライバは、それの累積された状態を問題のトークンとならんで破棄し、新たに開始する。
【０２８９】
ドライバの挙動は完全に決定論的なので、ダイアグラムで簡単にモデル化できる。この文書にはいくつかのそのような図があり、それらは一貫した表記を採用している。
【０２９０】
円。表におけるある状態。これは、状態のタイムアウトも示し、それは無限であり得る。
【０２９１】
実線矢印。シフトアクション。エッジ上のラベルは、マッチングされるトークンを示す。矢印は次の状態に至る。
【０２９２】
後ろ向き矢印。縮小アクション、ポップフェーズ。エッジ上のラベルは、マッチングされるトークンを示す。矢印は適用されている文法規則から至る。
【０２９３】

菱形。文法規則。括弧内のテキストは、スタックからポップするエントリの数およびマッチングされるべき記号を示す。

【０２９４】

破線矢印。縮小アクション、プッシュフェーズ。エッジ上のラベルは、マッチングされるべき記号を示す。

【０２９５】

ダイアグラムを使用して、ドライバのアルゴリズムを実行することができる。

１．別の紙片を使用して、スタックを追跡する。最初のデッドラインとともに、状態ｓ０をスタックにプッシュすることから始める。

【０２９６】

２．スタックの一番上の状態および現在の入力トークンを見る。その状態から抜け出し、現在のトークンでラベル付けされているエッジに従う。

【０２９７】

３．エッジが別の状態につながる実線の矢印の場合は、それが指す状態を新たなデッドラインおよび現在のトークンとともにスタックにプッシュし、入力を次のトークンに進めて、ステップ２に戻る。

【０２９８】

４．エッジが、規則に接続する後ろ向き矢印の場合は、それが接続する規則を見る。この規則は、ポップするエントリ数およびリダクションを一覧表示する。示された数のエントリをスタックからポップし、新たな一番上の状態を見る。その状態から通じ、リダクションでラベル付けされた破線のエッジを見つける。破線のエッジが指し示す状態を新たなデッドラインおよびリダクションとともに、スタック上にプッシュする。リダクションがオメガではない場合は、ステップ２に戻り、それ以外の場合は、ステップ１に戻る。

【０２９９】

実施例。このセクションでは、アクション中のＴＦＲ（１）のいくつかの例を示す。実行可能な場合は、それを、ストリーミングＳＱＬのように式の評価に基づくストリームベースのパターンマッチングの他の一般的な形式の挙動と比較する。

【０３００】

実施例１：ペット

この例では、ペットのリストを探すマッチャーを考える。各リストは任意の数のペット（またはペットがいない）を含むことができ、各ペットはcat（猫）、dog（犬）またはcanary（カナリア）のいずれかである。リストは、入力ストリームから５０の時間単位の間隔で収集される。

【０３０１】

表Ｂは、文法を示している。オメガルール(r0)はペットのリストをマッチングする。各リストは、ペットのリストにペットが続くもの(r5)、または無(r4)から構成されている。リストを収集するためのこの文法的構造は、入力の右端導出を求めるパーサの共通の特徴である。規則r1, r2,およびr3は、ペットが何であり得るかを定義する。

【０３０２】

表Ｃは、この文法から導出した状態を示している。「s4」などのアクションは「シフトして状態４に進む」を意味し、「r2」などのアクションは「規則２で縮小して遷移を辿る」ことを意味する。カールする矢印は再起動を意味する。空白のアクションはパニックを意味する。

【０３０３】

図１５Ａ～図１５Ｂは、表Ｃと同じ情報をより視覚的に理解できる方法で示す。この図では、たとえば、「dog」と他の２種類のペットとの区別が、一致において非常に早い時期に起こるのに対し、「cat」と「canary」との区別は、さらに２つのステップを要することが容易にわかる。

【０３０４】

【表１４】

表Ｂ：“pets”（ペット）の文法

規則	構文
r0	Ω → pets
r1	pet → «c» «a» «n» «a» «r» «y»
r2	pet → «c» «a» «t»
r3	pet → «d» «o» «g»
r4	pets → τ50
r5	pets → pets pet

【０３０５】

【表１５】

表Ｃ：“pets”のパーサ表

状態	アクション										遷移		タイムアウト
	a	c	d	g	n	o	r	t	y	—	pet	pets	
s0		r4	r4							r4		s1	τ50
s1		s3	s4							Ծ	s2		τ∞
s2		r5	r5							r5			τ∞
s3	s5												τ∞
s4						s6							τ∞
s5					s7			s8					τ∞
s6				s9									τ∞
s7	s10												τ∞
s8		r2	r2							r2			τ∞
s9		r3	r3							r3			τ∞
s10							s11						τ∞
s11									s12				τ∞
s12		r1	r1							r1			τ∞

【０３０６】

図１５Ａ～図１５Ｂは、“pets”の状態機械図を示す。図は規則および状態を示している。

図１５Ａには、状態ｓ０　１５１０、ｓ１　１５１５、ｓ２　１５２０、ｓ３　１５２５、

およびｓ４　１５３０がある。規則ｒ０　１５３３、ｒ５　１５３５、およびｒ４　１５３６がある。矢印１５４０および１５４２は、図１５Ｂの対応する矢印に接続する。図１５Ｂには、状態ｓ６　１５５０、ｓ５　１５５５、ｓ７　１５６０、ｓ８　１５６５、ｓ９　１５６８、ｓ１０　１５７０、ｓ１１　１５７５、およびｓ１２　１５８０がある。規則ｒ２　１５８３、ｒ１　１５８５、ｒ３　１５８８がある。

【０３０７】

表Ｄは、ドライバが入力「catcanarydog」を時間で１単位分分離された各イベントとマッチングするトレースを示している。入力は次いで時間単位５０まで経過で終了する。

【０３０８】

注意すべき点がいくつかある：（１）ドライバは入力トークン（スタック上の）を、それらがマッチするパターンの部分を認識するのに十分な時間だけバッファリングする。それらがマッチすると、入力は縮小される。（２）各入力は、ドライバによって一度だけ検査される。（３）ドライバは、入力があることとならんで、入力がないことに、反応する。最終的なリダクションは単に経過トリガされる。（３）ドライバは、起動した状態で終了し、マッチングを続ける準備ができている。

【０３０９】

【表１６】

表Ｄ：“pets”のサンプルトレース

ステップ	スタック	入力	アクション
1	[s0-1050]	1000:«c» 1001:«a» 1002:«t» 1003:«c» 1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	r4により縮小，s1に進む
2	[s0-1050] [s1-1050 pets]	1000:«c» 1001:«a» 1002:«t» 1003:«c» 1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s3に進む
3	[s0-1050] [s1-1050 pets] [s3-1050 «c»]	1001:«a» 1002:«t» 1003:«c» 1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s5に進む
4	[s0-1050] [s1-1050 pets] [s3-1050 «c»] [s5-1050 «a»]	1002:«t» 1003:«c» 1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s8に進む
5	[s0-1050] [s1-1050 pets] [s3-1050 «c»] [s5-1050 «a»] [s8-1050 «t»]	1003:«c» 1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	r2により縮小，s2に進む
6	[s0-1050] [s1-1050 pets] [s2-1050 pet]	1003:«c» 1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	r5により縮小，s1に進む
7	[s0-1050] [s1-1050 pets]	1003:«c» 1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s3に進む
8	[s0-1050] [s1-1050 pets] [s3-1050 «c»]	1004:«a» 1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s5に進む

9	[s0-1050] [s1-1050 pets] [s3-1050 «c»] [s5-1050 «a»]	1005:«n» 1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s7に進む
10	[s0-1050] [s1-1050 pets] [s3-1050 «c»] [s5-1050 «a»] [s7-1050 «n»]	1006:«a» 1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s10に進む
11	[s0-1050] [s1-1050 pets] [s3-1050 «c»] [s5-1050 «a»] [s7-1050 «n»] [s10-1050 «a»]	1007:«r» 1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s11に進む
12	[s0-1050] [s1-1050 pets] [s3-1050 «c»] [s5-1050 «a»] [s7-1050 «n»] [s10-1050 «a»] [s11-1050 «r»]	1008:«y» 1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s12に進む
13	[s0-1050] [s1-1050 pets] [s3-1050 «c»] [s5-1050 «a»] [s7-1050 «n»] [s10-1050 «a»] [s11-1050 «r»] [s12-1050 «y»]	1009:«d» 1010:«o» 1011:«g» 1050:τ	r1により縮小，s2に進む
14	[s0-1050] [s1-1050 pets] [s2-1050 pet]	1009:«d» 1010:«o» 1011:«g» 1050:τ	r5により縮小，s1に進む
15	[s0-1050] [s1-1050 pets]	1009:«d» 1010:«o» 1011:«g» 1050:τ	シフト，s4に進む
16	[s0-1050] [s1-1050 pets] [s4-1050 «d»]	1010:«o» 1011:«g» 1050:τ	シフト，s6に進む
17	[s0-1050] [s1-1050 pets] [s4-1050 «d»] [s6-1050 «o»]	1011:«g» 1050:τ	シフト，s9に進む
18	[s0-1050] [s1-1050 pets] [s4-1050 «d»] [s6-1050 «o»] [s9-1050 «g»]	1050:τ	r3により縮小，s2に進む
19	[s0-1050] [s1-1050 pets] [s2-1050 pet]	1050:τ	r5により縮小，s1に進む
20	[s0-1050] [s1-1050 pets]	1050:τ	報告および再起動
21	[s0-1100]	1050:τ	破棄

【０３１０】
ストリームプロセッサは、一致を探すために式の評価を使用することができる。たとえば、ストリーミングＳＱＬのSELECT文では、where節が検出される。この節は、評価する式を特定し、結果が真の場合に選択はマッチする。たとえば、現在の入力が識別子語で表された場合、ペットを探すwhere節は、語＝“cat”または語＝“dog”または語＝“canary”かもしれない。
【０３１１】
選択が一致を達成するまで、語の内容をバッファリングする必要がある。新たな入力が到着するたびに、語を１文字分拡張し、その後、式全体を再評価する必要がある。

【０３１２】
以前の反復から学習したことは、次のものに持ち込まれない。たとえば、語が現在“ｄｏ”を含むとした場合、式はそれを“ｃａｔ”および“ｃａｎａｒｙ”と比較する必要が依然としてあるであろうが、それらはマッチする可能性はない。判定に達する前に、語の各文字が任意の回数調べられるかもしれない。
【０３１３】
これは、入力を２回以上調べることはないＴＦＲ（１）よりはるかに効率が悪い。各構文解析状態には、それに先行する状態に関する情報が暗示的に担持される。たとえば、犬対非犬の判断はｓ１で行われる。ｓ３およびｓ４のような後続の状態は、“ｄｏｇ”が依然として一致の可能性があるかどうかをすでに知っている。
【０３１４】
ＴＦＲ（１）状態表を構築する。このセクションでは、文法規則のセットから状態表を構築するために使用されるアルゴリズムについて記載する。３つの全体的なステップがある：
１．すべての記号の最初の集合および後続の集合を計算する。
【０３１５】
２．開始アイテムから開始状態を生成し、開始状態から直接的または間接的に到達可能なすべての状態を生成する。
【０３１６】
３．生成された各状態について表エントリを生成する。
しかしながら、これらのステップを説明し得る前に、まず、関連する概念のいくつかを定義する必要がある。
【０３１７】
概念。最初および後続。記号の最初の集合は、その記号から導出されたイベントのシーケンスを開始できるトークンのセットである。記号が空の規則を伴うリダクションである場合、記号の最初の集合には特別なマーカーεも含まれる。
【０３１８】
First(X)を計算するために：
１．Xがトークンである場合、First(X)は単に{X}である。
【０３１９】
２．Xが空の規則を伴うリダクションである場合は、First(X)にεを加える。
３．Xをその左辺とする各空でない規則に対して、First(X)に最初の記号の各々の最初の集合を右辺に加える。この右辺の記号がそれ自体が空の規則を伴うリダクションである場合、First(X)に次の記号の最初の集合を右辺に加える。空の規則のない右辺の記号に遭遇するまで、または右辺の記号がなくなるまで、この方法で続行する。
【０３２０】
最初の集合は、２つ以上の記号からなるシーケンスαに対しても定義される。First(α)を計算するために：
１．αにおける最初の記号の最初の集合から始める。これにεが含まれていない場合は、この集合をFirst(α)として返す。
【０３２１】
２．それにεが含まれている場合は、集合からεを除去し、αにおける次の記号の最初の集合を加える。このようにして、εを含まない最初の集合を有する記号に遭遇するまで、またはαの終わりに達するまで、継続する。
【０３２２】
記号の後続集合は、認識可能なシーケンスにおいてその記号に続き得るイベントのセットである。Follow(X)を計算するには：
１．XがΩの場合、そのFollow(X)にはFirst(Ω)におけるすべてのイベントとならんでブレークイベントが含まれる。
【０３２３】
２．Xが右辺に現れる各規則を調べる。Aをこの規則の左辺とし、βをこの規則においてX

の右に現れる記号のシーケンスとする。βが空でない場合、Follow(X)にはFirst(β)が含まれ；そうでなければ、Follow(X)にはFollow(A)が含まれる。

【０３２４】

アイテムおよび状態。文法のアイテムは、一致中に到達した特定のポイントを表す。アイテムは、文法の規則の１つへの参照、規則内のドット位置、および先読みイベントから構成される。たとえば、[pet → ≪c≫ ・ ≪a≫ ≪t≫, ≪c≫]はペットの文法のアイテムである。

【０３２５】

アイテムにおけるドット位置は、次に一致が生ずる厳密な位置を示す。ドットは、規則の右辺にある任意の記号の前に来ることもあれば、最後の記号の後に来ることもある。後者の場合、アイテムは完全であると言われる。アイテムが完全でない場合は、ドットの直後の記号はアイテムのコーナーと呼ばれる。完全なアイテムにはコーナー記号はない。ドットの前の記号のシーケンスはアルファシーケンスと呼ばれ、コーナーの後の記号のシーケンスはベータシーケンスと呼ばれる。アルファシーケンスおよびベータシーケンスのいずれかまたは両方を空にすることができる。

【０３２６】

空の規則を参照するアイテムは、単一の可能なドット位置しか有さない。そのようなアイテムは完全と考えられる。それは、コーナー記号はなく、それのアルファシーケンスおよびベータシーケンスは空である。

【０３２７】

アイテムの先読みイベントは、トークンまたはブレークのいずれかである。完全なアイテムでは、先読みは、パーサジェネレータがアイテムをそのまま縮小するか別のイベントにシフトするかを決定するのを助ける。

【０３２８】

文法の開始アイテムは、マッチングが始まり得るアイテムの１つである。各開始アイテムは、オメガルールを参照し、そのドットを左端に有する。文法の開始アイテムのセットを生成するには、Follow(Ω)を計算する。Follow(Ω)の各記号Fに対して、開始アイテムを先読みFとともに加える。

【０３２９】

状態はアイテムのセットである。たとえば、{[pets → pets pet ・, ≪c≫], [pets → pets pet ・, ≪d≫], [pets → pets pet ・, ―]}は、ペットの文法における状態ｓ２である。

【０３３０】

各アイテムはタイムアウトを有する。アイテムが完全である場合、そのタイムアウトはその規則のタイムアウトと同じになる。空でない規則またはラムダルールは無限のタイムアウトを有する。）アイテムが完全でない場合、そのタイムアウトは無限である。

【０３３１】

各状態もタイムアウトを有し、それはそのアイテムの中で最も小さいタイムアウトに等しい。

【０３３２】

クロージャ。状態はアイテムのセットであるが、アイテムのすべてのセットが状態であるのではない。状態になるには、セットは完全なクロージャでなければならない。クロージャ関数は、アイテムの初期セットから、状態を完成させるのに必要なアイテムを加えることによって、完全な状態を生成する。

【０３３３】

Jがアイテムのセットである場合、Closure(J)を次のように計算する。

１．最初にJにおけるすべてのアイテムを含むアイテムのリストtodo、および最初は空であるアイテムのセットdoneで開始する。

【０３３４】

２．todoが空の場合、doneを文法の状態として返す。

３．todoから１つのアイテムをポップする。アイテムがすでにdoneにある場合、または

アイテムが完成している場合（つまり、その右端にドットがある場合）、アイテムを破棄してステップ２に戻る。
【０３３５】
４．Bをアイテムのコーナー記号とし、βをアイテムのベータシーケンスとし、Xをアイテムの先読みとし、nextをイベントFirst(βX)のセットとする。
【０３３６】
５．ブレークイベントをnextに加える。
６．Bが左辺に現れる各規則について、およびnextにおける各イベントYについて、アイテム[rule, 0, Y]をtodoに加える。
【０３３７】
７．ステップ２に戻る。
Goto。状態は、あるシーケンスγに対して実行可能なアイテムのセットを含み、各シーケンスは、それをこのように表す状態を有する。したがって、γと同一であるが、１イベント分だけより長いシーケンスγXも、ある状態によって表されなければならない。γの状態がJの場合、γXの状態はXを伴うJのgotoと呼ばれる。
【０３３８】
Goto(J, X)を計算するには：
１．最初にＪにおけるすべてのアイテムを含むアイテムのリストtodo、および最初は空であるアイテムのセットnextで開始する。
【０３３９】
２．todoが空の場合、Closure(next)をgoto状態として返す。
３．todoから１つのアイテムをポップする。アイテムが完成している場合、またはアイテムのコーナー記号がXでない場合は、アイテムを破棄してステップ２に戻る。
【０３４０】
４．アイテムとドットとの規則による規則をアイテムのドット位置とし、Yをアイテムの先読みとする。アイテム[rule, dot + 1, Y]をnextに加える。
【０３４１】
５．ステップ２に戻る。
構築アルゴリズム。ここで、この章の冒頭で述べた構築アルゴリズムの３つのステップについて説明する。最初の集合および後続の集合を計算する最初のステップは、単純に、すべての記号を通って反復し、前のセクションの定義を適用することである。ここで、第２のステップおよび第３のステップについて説明する。
【０３４２】
状態生成。文法のすべての状態を生成するには：
１．最初に開始アイテムのクロージャを含む状態のリストtodo、および最初は空である状態のセットdoneで、開始する。
【０３４３】
２．todoが空の場合は、doneを文法の状態のセットとして返す。
３．todoから１つの状態Jをポップする。Jがすでにdoneにある場合は、それを破棄してステップ２に戻る。
【０３４４】
４．（Ωおよびブレークを含む）文法における各記号Xについて、KをGoto(J, X)とする。Kが空でない場合は、Kをtodoに加える。
【０３４５】
５．ステップ２に戻る。
表Ｅは、このアルゴリズムを表Ｂの文法に適用した結果を示す。
【０３４６】

【表１７】

表Ｅ：“pets”の状態のアイテム

状態	アイテム
s0	[Ω → · pets, «c»] [Ω → · pets, «d»] [Ω → · pets, —] [pets → τ50, «c»] [pets → τ50, «d»] [pets → τ50, —] [pets → · pets pet, «c»] [pets → · pets pet, «d»] [pets → · pets pet, —]
s1	[Ω → pets ·, «c»] [Ω → pets ·, «d»] [Ω → pets ·, —] [pet → · «c» «a» «n» «a» «r» «y», «c»] [pet → · «c» «a» «n» «a» «r» «y», «d»] [pet → · «c» «a» «n» «a» «r» «y», —] [pet → · «c» «a» «t», «c»] [pet → · «c» «a» «t», «d»] [pet → · «c» «a» «t», —] [pet → · «d» «o» «g», «c»] [pet → · «d» «o» «g», «d»] [pet → · «d» «o» «g», —] [pets → pets · pet, «c»] [pets → pets · pet, «d»] [pets → pets · pet, —]
s2	[pets → pets pet ·, «c»] [pets → pets pet ·, «d»] [pets → pets pet ·, —]
s3	[pet → «c» · «a» «n» «a» «r» «y», «c»] [pet → «c» · «a» «n» «a» «r» «y», «d»] [pet → «c» · «a» «n» «a» «r» «y», —] [pet → «c» · «a» «t», «c»] [pet → «c» · «a» «t», «d»] [pet → «c» · «a» «t», —]

s4	[pet → «d» · «o» «g», «c»] [pet → «d» · «o» «g», «d»] [pet → «d» · «o» «g», —]
s5	[pet → «c» «a» · «n» «a» «r» «y», «c»] [pet → «c» «a» · «n» «a» «r» «y», «d»] [pet → «c» «a» · «n» «a» «r» «y», —] [pet → «c» «a» · «t», «c»] [pet → «c» «a» · «t», «d»] [pet → «c» «a» · «t», —]
s6	[pet → «d» «o» · «g», «c»] [pet → «d» «o» · «g», «d»] [pet → «d» «o» · «g», —]
s7	[pet → «c» «a» «n» · «a» «r» «y», «c»] [pet → «c» «a» «n» · «a» «r» «y», «d»] [pet → «c» «a» «n» · «a» «r» «y», —]
s8	[pet → «c» «a» «t» ·, «c»] [pet → «c» «a» «t» ·, «d»] [pet → «c» «a» «t» ·, —]
s9	[pet → «d» «o» «g» ·, «c»] [pet → «d» «o» «g» ·, «d»] [pet → «d» «o» «g» ·, —]
s10	[pet → «c» «a» «n» «a» · «r» «y», «c»] [pet → «c» «a» «n» «a» · «r» «y», «d»] [pet → «c» «a» «n» «a» · «r» «y», —]
s11	[pet → «c» «a» «n» «a» «r» · «y», «c»] [pet → «c» «a» «n» «a» «r» · «y», «d»] [pet → «c» «a» «n» «a» «r» · «y», —]
s12	[pet → «c» «a» «n» «a» «r» «y» ·, «c»] [pet → «c» «a» «n» «a» «r» «y» ·, «d»] [pet → «c» «a» «n» «a» «r» «y» ·, —]

【０３４７】
表生成。状態が生成された後、各状態はパーサ表の行に変換される。各行は、アクションおよび遷移の２つの部分で構成されている。
【０３４８】
状態Jのアクションを計算するには：
Ｊの各アイテムについて、規則をアイテムの規則とし、Bをアイテムのコーナー記号（もしあれば）とし、Xをアイテムの先読みとする。
【０３４９】
ａ．アイテムが完成しており、その左辺がΩの場合、action(J, X)はrestart（再起動）である。

【０３５０】
ｂ．アイテムが完成しており、その左辺がΩでない場合、action(J, X)はreduce(rule)である。
【０３５１】
ｃ．アイテムが完成しておらず、Bがトークンであり、Goto(J, B)が空でない場合、action(J, X)はshift(Goto(J, B))である。
【０３５２】
ｄ．他のすべての場合において、action(J, X)はパニックである。
状態Jの遷移を計算するには：
文法における各リダクションAについて、KをGoto(J, A)とする。
【０３５３】
ａ．Kが空でない場合、transition(J, A)はKである。
ｂ．Kが空である場合、transition(J, A)は未定義である。
【０３５４】
これらのアルゴリズムを表Ｅに適用すると、表Ｂが生成される。
その他の最適化。先に述べたように、ＴＦＲ（１）パーサ表を構築するためのこのアルゴリズムは、標準ＬＲ（１）パーサ表を構築するためのアルゴリズムと同様であり、それの拡張である。ＬＲ（１）パーサに適用されるさまざまな一般化、最適化、および状態リダクション戦略は、ＴＦＲ（１）パーサ表にも準用される。そのような戦略には、ＬＲ（１）からＬＡＬＲ（１）への最適化およびＬＲ（１）のＧＬＲ（１）への一般化が含まれるが、これらに限定されない。
【０３５５】
データフローコンピューティング環境で使用するためのリアクティブ関数を記述し変換する方法は、（１）リアクティブ関数を識別することと、（２）関数への入力を提供するパラメータのパターンを識別することと、（３）関数に渡された引数に基づいて評価されるべき式を識別することと、（４）パラメータのパターンを、パターンにマッチする入力のシーケンスを認識することができる状態機械に変換することと、（５）入力データを出力データに変換するよう関数を呼び出す機能で状態機械を拡張することと、（６）単純なソフトウェアまたはハードウェアによる自動化が可能な決定性プッシュダウンオートマトンとして状態機械を実現することとを備える。
【０３５６】
一実装形態では、方法は、物理量を監視し、監視された物理量をデジタル形式のデータストリームに変換するハードウェアセンサからデータストリームを受信することと、前記データストリームを入力キューに格納することとを備え、データストリームにおける各トークンは、トークンが受信されたときのタイムスタンプと共に格納され、前記方法はさらに、単一の方向に入力キューを通ってトークンを読出し、以前に読出されたトークンを再読出ししないことによって、１つ以上の所定のパターンにマッチするデータストリームにおけるパターンを識別することと、データストリーム内において所定のパターンを識別すると、所定のパターンが識別されたという肯定的な表示を前記出力キューにおいて出力することとを備える。
【０３５７】
この方法は、所定のパターンがデータストリームにおいて識別されていない間に、出力キューにおいて、所定のパターンが識別されたという否定的な表示を出力することをさらに備えることができ、所定のパターンは状態表および状態スタックに格納される。
【０３５８】
別の実装形態では、システムは以下を備える：入力キューは、コンピュータメモリにおいて第１のメモリ位置を含み、入力キューは、処理されるべきトークンの先入れ先出しシーケンスを第１のメモリ位置に格納し、キューにおける各トークンに関連付けられるタイムスタンプは、関連付けられるトークンが入力キューに入った時間を示し、トークンはネットワークを介して入力キューによって受信される。ドライバコンポーネントは入力キュー

に接続され、ドライバコンポーネントは、バックトラッキングなしで入力キューにおけるトークンを処理し、１つ以上の所定の入力パターンにマッチする、トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベント出力を生成する。出力キューは、ドライバコンポーネントに接続され、コンピュータメモリにおいて第２のメモリ位置を含み、出力キューは、第２のメモリ位置に、ドライバコンポーネントによって生成された出力イベントの先入れ先出しシーケンスを格納する。状態表コンポーネントはドライバコンポーネントに接続され、状態表コンポーネントは所定の入力パターンを状態表フォーマットで格納する。状態スタックコンポーネントは、ドライバコンポーネントに接続され、コンピュータメモリにおいて第３のメモリ位置を含み、状態スタックコンポーネントは第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納する。フレームには、変換状態番号、記号、およびデッドラインが含まれる。

【０３５９】

入力キューで受信されるトークンは、物性をデジタル量に変換するハードウェアセンサによって生成することができる。ドライバコンポーネントの動作は、状態スタックの一番上のフレーム上で参照される状態を検査することと、入力キューにおける次の入力トークンを検査し、状態スタックに基づいてそれのカテゴリを判断することと、状態表によって示されるカテゴリのトークンのために状態表によって示されるアクションを実行することとを含み得る。

【０３６０】

入力キューは、入力キューによって受信された時間に基づく順序付けられたリストにトークンを格納する。キューにおける第１の方向は、最も早く受信されたトークンから最後に受信されたトークンまでの時間順序である。ドライバコンポーネントは、入力キューを第１の方向において検査することによってトークンを処理し、以前に検査されたトークンを検査しない。

【０３６１】

入力キューは、入力キューによって受信された時間順序でトークンをリストに格納する。キューにおける第１の方向は、最も早く受信されたトークンから最後に受信されたトークンまでの時間順序である。キューにおける第２の方向は、最後に受信されたトークンから最も早く受信されたトークンまでの時間順序である。第２の方向は、第１の方向とは反対である。ドライバコンポーネントは、入力キューを第１の方向または第２の方向のいずれかのみで検査し、第１および第２の方向の両方では検査しないことによって、トークンを処理する。

【０３６２】

ドライバは入力キューの各トークンを１回だけ読出し、読出されたトークンをあとで再び読出すためにバッファに保持しない。マッチングされる所定の入力パターンは、無限長のある数のトークンを有することができる。状態表コンポーネントは状態機械を指定し、ある数の行を含み、各行は状態機械の変換状態を提供する。

【０３６３】

各行は、入力トークンカテゴリ値または入力ブレークカテゴリ値を、その値の入力トークンが受信されるととられるアクションにマッピングするアクションのセットと、派生記号型を、その型の派生記号を合成するとドライバが変更すべき状態にマッピングする遷移のセットと、状態機械が特定の状態にとどまることが許される時間の間隔を示すタイムアウトとを含む。

【０３６４】

変換状態番号は、状態表の行への参照である。記号は、入力トークンもしくは他の記号から導出されたトークンもしくは中間記号、または組み合わせである。デッドラインは、ドライバがフレームをスタックからポップして外す未来の瞬間である。

【０３６５】

一実装形態では、ドライバコンポーネントおよび状態表コンポーネントは、プログラマブ

ルゲートアレイにおいて実施される。別の実装形態では、ドライバコンポーネントおよび状態表コンポーネントは、コンピュータを使用して実施される。状態表コンポーネントは状態機械を指定し、ドライバコンポーネントは、ストリームデータを有するトークンが入力キューにおいて受信されない時間の経過に基づいて、状態機械における第１の状態から状態機械の第２の状態に変化する。

【０３６６】

状態表コンポーネントは、状態機械を指定する。ドライバコンポーネントは、ストリームデータを有するトークンのシーケンスが第１の期間中に入力キューにおいて受信されることに基づいて、状態機械における第１の状態から状態機械の第２の状態に変化する。ドライバコンポーネントは、ストリームデータを有するトークンのシーケンスが第２の期間中に入力キューにおいて受信されないことに基づいて、状態機械における第３の状態から状態機械の第４の状態に変化する。

【０３６７】

別の実装形態では、方法は以下を備える：コンピュータメモリにおいて第１のメモリ位置を含む入力キューを提供することを備え、入力キューは、処理されるべきトークンの先入れ先出しシーケンスを第１のメモリ位置に格納し、キューにおける各トークンに関連付けられるタイムスタンプは、関連付けられるトークンが入力キューに入った時間を示し、トークンはネットワークを介して入力キューによって受信される。入力キューに接続されるドライバコンポーネントを提供することを備え、ドライバコンポーネントは、バックトラッキングなしで入力キューにおけるトークンを処理し、１つ以上の所定の入力パターンにマッチする、トークンのシーケンスにおけるパターンを識別し、マッチした所定の入力パターンを識別すると、出力イベント出力を生成する。ドライバコンポーネントに接続され、コンピュータメモリにおいて第２のメモリ位置を含む出力キューを提供することを備え、出力キューは、第２のメモリ位置に、ドライバコンポーネントによって生成された出力イベントの先入れ先出しシーケンスを格納する。ドライバコンポーネントに接続される状態表コンポーネントを提供することを備え、状態表コンポーネントは所定の入力パターンを状態表フォーマットで格納する。ドライバコンポーネントに接続され、コンピュータメモリにおいて第３のメモリ位置を含む状態スタックコンポーネントを提供することを備え、状態スタックコンポーネントは第３のメモリ位置にフレームの後入れ先出しのシーケンシャルストレージを格納し、フレームには、変換状態番号、記号、およびデッドラインが含まれる。

【０３６８】

状態表コンポーネントのために状態表を生成することは、終端記号の集合を識別することを含み、各記号は入力データのカテゴリを表し、さらに、非終端記号の集合を識別することを含み、各非終端記号は、終端記号または非終端記号の少なくとも１つのパターンを表し、さらに、文法規則のセットを識別することと、トップレベルの規則を識別することと、属性のセットを識別することと、各識別された終端記号および非終端記号に対して最初の集合を計算することと、各識別された終端記号および非終端記号について後続集合を計算することと、対応する後続集合の要素の各々を有する、トップレベルの規則のクロージャから開始状態を生成することとを含む。

【０３６９】

状態表を生成することは、開始状態から状態変化遷移が存在する各状態のクロージャを帰納的に生成することによって開始状態から到達可能なすべての状態を生成することと、組み合わせおよびリダクションによって状態のセットを最適化することと、各状態についてのアクション、遷移、およびタイムアウト値を生成することとを含む。

【０３７０】

文法規則のセットは、以下を含み得る：各空でない規則はA → B0 . . . Bnの形を取り、ここでAは非終端記号であり、B0からBnは各々終端記号または非終端記号のいずれかであり；各空の規則はA → τの形を取り、ここで、τはある有益な時間単位で特徴づけられる有限時間量であるか、または無制限な時間量を示す無限大である。

【０３７１】
属性のセットは、以下のものを含むことができる：各属性は、α:T = {A0 → λ0, . . . An → λn}の形式をとり、αは属性の名前であり、Tは属性の型であり、Aは非終端記号であり、λはその記号の属性の値を計算するために使用される関数である。トップレベルの規則は、Ω→Bの形を取ることができ、Bは終端記号または非終端記号のいずれかである。
【０３７２】
本発明のこの記載は、例示および説明のために提示されたものである。それは、網羅的であることを意図するものでもなく、本発明を記載された正確な形態に限定することも意図されておらず、上記教示に照らして多くの修正および変形が可能である。実施形態は、本発明の原理およびその実際の適用を最もよく説明するために選択され、記載された。この記載により、当業者は、本発明を、さまざまな実施形態において、および特定の用途に適したさまざまな修正を加えて、最も有効に活用し実施することが可能になる。本発明の範囲は、特許請求の範囲によって規定される。

【図面】

【図１】

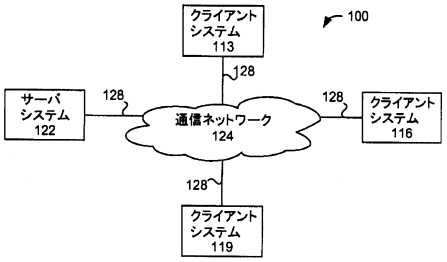

Figure 1

【図２】

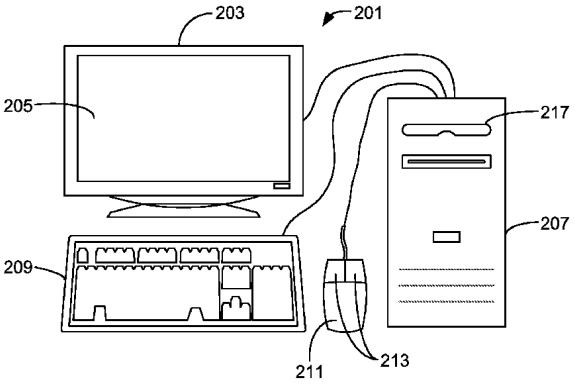

Figure 2

【図３】

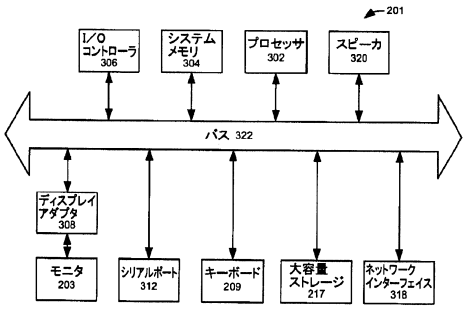

Figure 3

【図４】

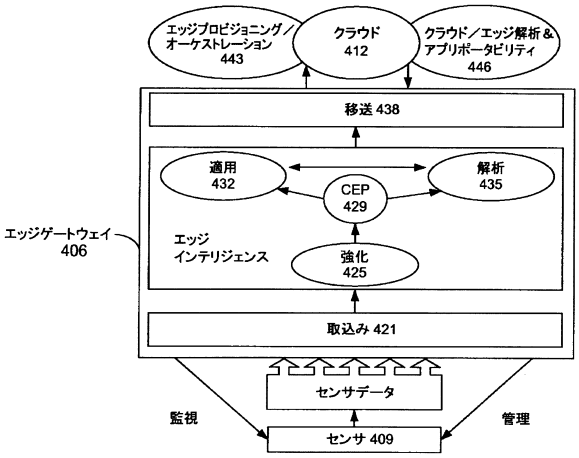

Figure 4

【図５】

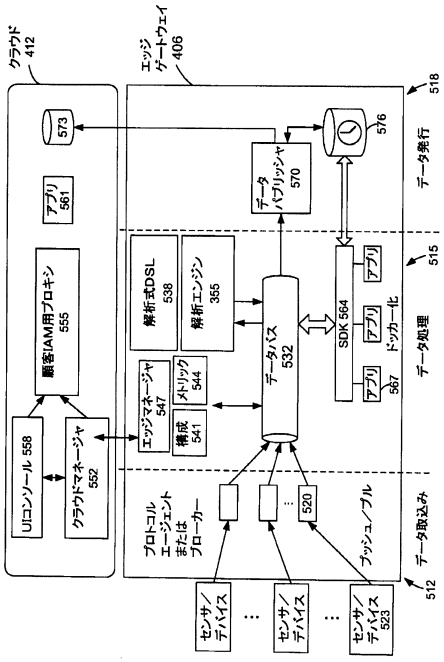

Figure 5

【図６】

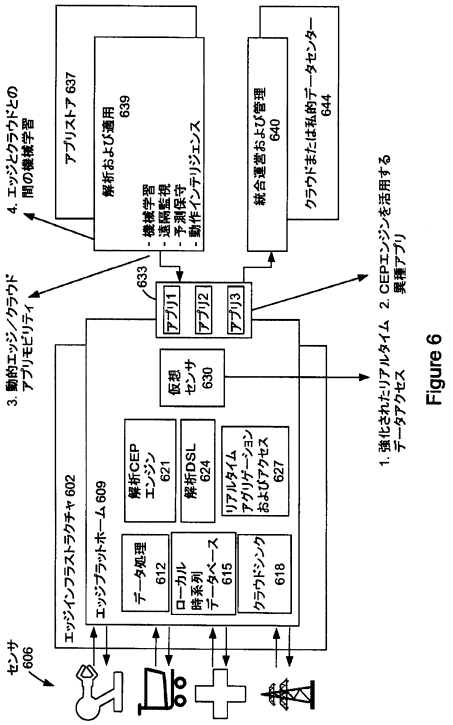

Figure 6

【図７】

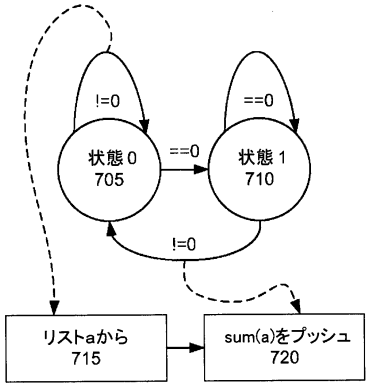

Figure 7

【図８】

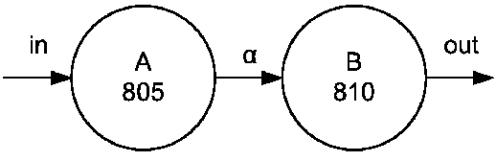

Figure 8

【図９】

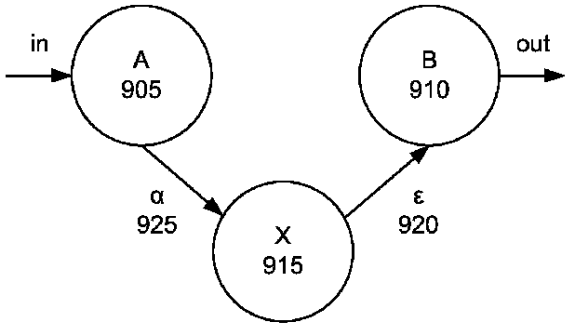

Figure 9

【図１０】

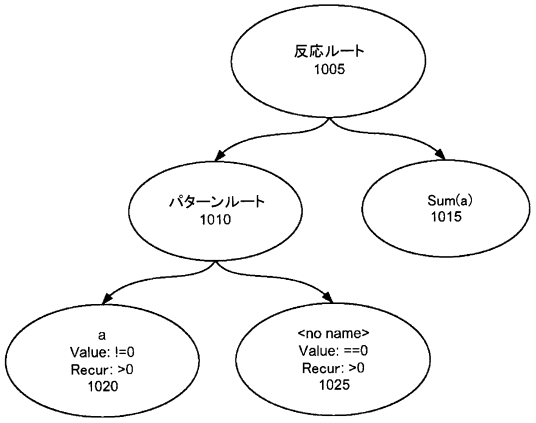

Figure 10

【図１１】

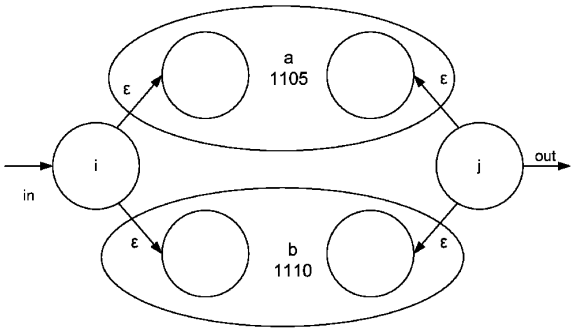

Figure 11

【図１２】

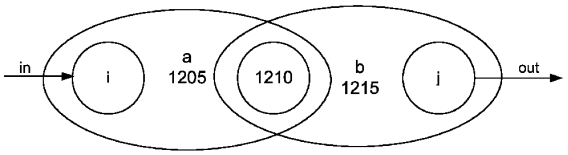

Figure 12

【図１３】

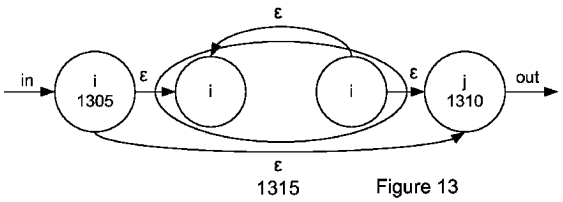

Figure 13

【図１４】

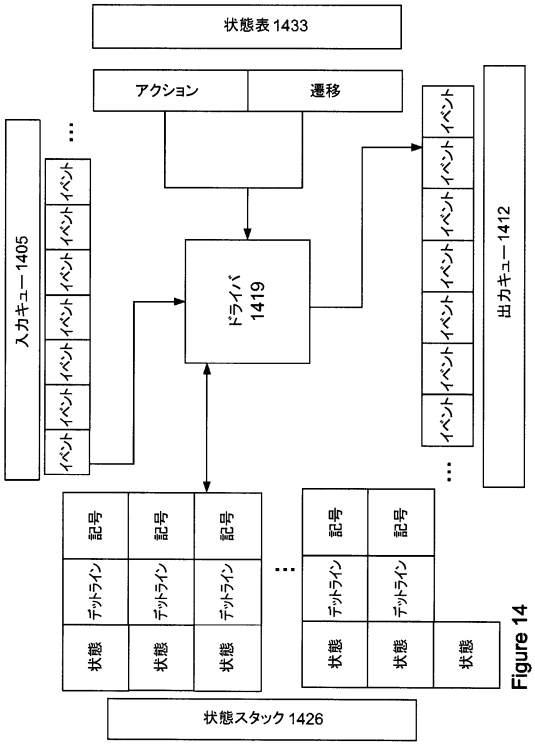

Figure 14

【図１５Ａ】

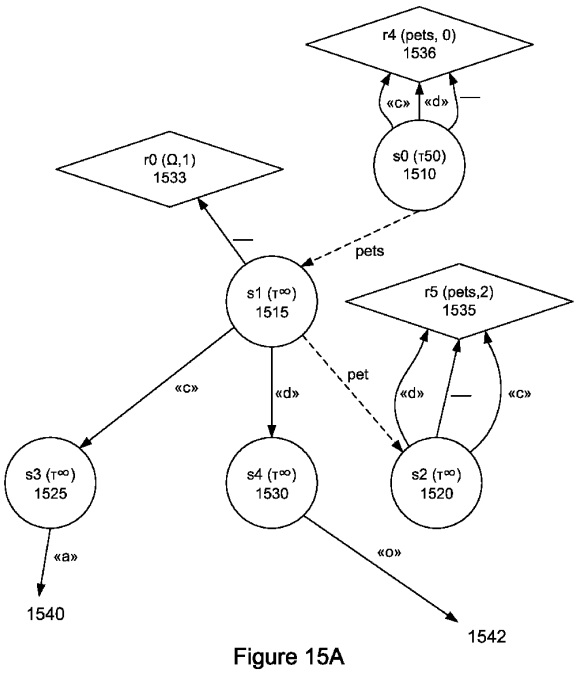

Figure 15A

【図１５Ｂ】

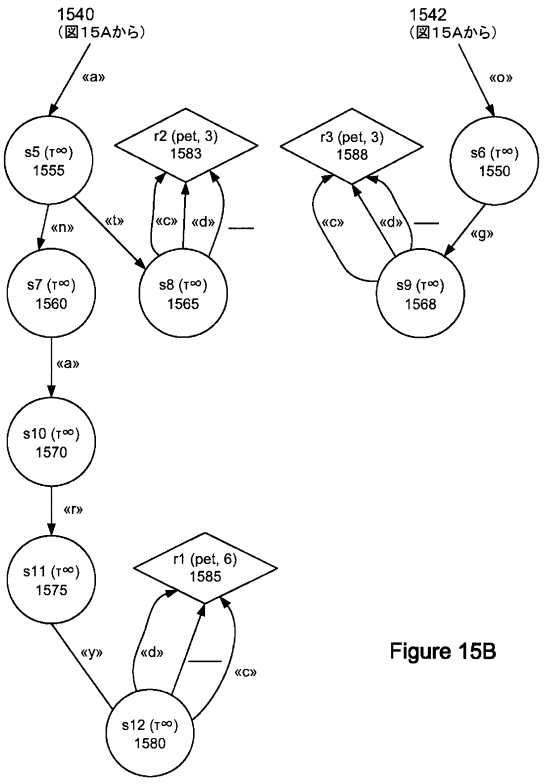

Figure 15B

フロントページの続き

(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　62/312,223
(32)優先日　平成28年3月23日(2016.3.23)
(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　62/312,255
(32)優先日　平成28年3月23日(2016.3.23)
(33)優先権主張国・地域又は機関
米国(US)
(72)発明者　ルーカス，ジェイソン
アメリカ合衆国、９８０５９ ワシントン州、レントン、キトサップ・プレイス・ノース・イースト、１８００
(72)発明者　シャーマ，アビシェーク
アメリカ合衆国、９４０４０ カリフォルニア州、マウンテン・ビュー、コンチネンタル・サークル、７０７、アパートメント・２３０
審査官　野元 久道
(56)参考文献　特表２００８－５２４９６５（ＪＰ，Ａ）
国際公開第２０１５／０７１９７８（WO，Ａ１）
米国特許出願公開第２００３／００３５５８２（ＵＳ，Ａ１）
米国特許出願公開第２００８／０２７０３４２（ＵＳ，Ａ１）
米国特許出願公開第２０１５／０１６１２１４（ＵＳ，Ａ１）
韓国公開特許第１０－２０１１－０１３８２３７（ＫＲ，Ａ）
(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｈ０４Ｌ　１２／００