DEIM Forum 2018 E1-5

シーケンスOLAPにおけるパターン拡張問合せ手法の提案

那須　勇弥† 　中挾　晃介†† 　北川　博之†††

† 筑波大学大学院システム情報工学研究科　〒305–8573 茨城県つくば市天王台 1-1-1
†† 公益財団法人鉄道総合技術研究所　〒185–8540 東京都国分寺市光町 2-8-38
††† 筑波大学計算科学研究センター　〒305–8573 茨城県つくば市天王台 1-1-1
E-mail: †yuuya.n@kde.cs.tsukuba.ac.jp, ††nakabasami.kosuke.39@rtri.or.jp, †††kitagawa@cs.tsukuba.ac.jp

あらまし　シーケンスOLAPはシーケンスデータに対して特定パターンの出現箇所（パターンオカレンス）を抽出し，次元の粒度の切替やパターンの拡張などの機能を用いて多次元・多段階な集約分析を実行できる手法である．シーケンスOLAPにおけるパターンの拡張を行うためには拡張前・拡張後の複数パターンのパターン認識とそれらのパターンオカレンス間の親子関係検出を実行する必要がある．しかし，汎用的なデータ管理システムであるRDBではこれらのパターン拡張に必要な処理を実行できない．そこで本研究ではRDBに蓄積されたシーケンスデータに対するシーケンスOLAPを実現するために複数パターンのパターン認識とパターンオカレンス間の親子関係検出をサポートする拡張記述MULTI_MATCH_RECOGNIZEとその問合せ処理手法の提案を行う．

キーワード　OLAP，時系列データ，SQL，Row Pattern Recognition

1. はじめに

近年，IoTやソーシャルメディアの普及に伴い多種多様なデータが生成されている．特に，RFIDを活用した人や物の移動ログデータや株価の変動データなど，何らかの順序に従って連続的に生成されるデータはシーケンスデータと呼ばれており，シーケンスデータ中の特定パターンの出現箇所（パターンオカレンスと呼ぶ）を抽出し，集約・集計するような分析手法が求められている．そのようなシーケンスデータに対する集約分析手法としてシーケンスOLAPが知られている．シーケンスOLAPはシーケンスデータから抽出したパターンオカレンスに対して次元の切替やパターンの拡張といった集約条件の変更操作を行いながら多次元・多段階の集約分析を実行する手法である．過去にはS-OLAP [8] やE-Cube [6] [7] などのシーケンスOLAP手法が提案されてきたが，いずれも独自の言語やシステムの開発を必要とし，データベースなどに蓄積されたデータとの統合的な処理が困難であるという問題点がある．

そこで，本研究では汎用的なデータ管理システムであるRDB(リレーショナルデータベース) に蓄積されたシーケンスデータに対するシーケンスOLAPを実現することを目的とする．シーケンスOLAPの重要な機能の1つであるパターンの拡張操作をサポートするためには，拡張前と拡張後の複数パターンのパターン認識と拡張前パターンを親，拡張後パターンを子としたパターンオカレンス間の親子関係の検出が必要となる．RDBに蓄積されたデータに対するパターン認識はSQL2016から標準化されたSQL/RPR(Row Pattern Recognition) [1] で定義されているMATCH_RECOGNIZE句を記述することによって実行することができる．しかし，MATCH_RECOGNIZE句で定義できるパターンは1つに限られており，拡張パターンのパターンオカレンスを抽出するためには再度の問合せが必要となる．また，拡張前パターンのパターンオカレンスと拡張パターンのパターンオカレンスの間の親子関係を検出する機能もサポートされていない．

この問題点を解決するために，本研究では複数パターンのパターン認識とパターンオカレンス間の親子関係検出をサポートする拡張記述MULTI_MATCH_RECOGNIZE句とその問合せ処理手法の提案を行う．提案手法によってパターンとその拡張パターンが1つの問合せで同時に定義可能となる．さらに，複数パターンのパターンオカレンス間の親子関係を検出し出力結果に含めることができる．また，本研究ではMULTI_MATCH_RECOGNIZE句の問合せ処理においてパターンオカレンスが抽出されるたびに逐次的に親子関係の検出を行う1-Phase手法とパターン認識終了後にまとめて親子関係の検出を行う2-Phase手法の2つの問合せ処理手法を提案する．本研究では，これらの2つの問合せ処理手法の性能を評価するために人工データを対象とした比較実験を行う．

2. 前提知識

2.1 シーケンスOLAP

シーケンスOLAPは蓄積されたシーケンスデータからパターンオカレンスを抽出し複雑な集約・集計処理を実行することができる集約分析手法である．例として，図1のsequence表に対してシーケンスOLAPを実行することを考える．sequence表には移動ログデータとしてsid(シーケンスのid)，location，timeを持つタプルが格納されているとする．シーケンスOLAPではまず，分析対象となるシーケンスデータに対してパターンを指定しパターン認識を実行することで，パターンの出現箇所を示すパターンオカレンスを抽出する．パターンは通常，(X Y) や (X Y Z) のように何らかの変数 (パターン変数と呼ぶ) の並びで表される．図1では，sequence表に対してパターン (X Y) についてパターン認識を実行した例を示している．ここで，sequence表に対するパターン (X Y) はある場所XからYへ

の移動パターンを表す．sequence 表からパターン (X Y) のパターンオカレンスを抽出した結果は図 1 のパターンオカレンス表に示されている．パターンオカレンス表では 1 タプルが 1 つのパターンオカレンスを示し，X_lo はパターン認識で X に該当したタプルの location，Y_lo はパターン認識で Y に該当したタプルの location を示している．X_time，Y_time も同様にそれぞれのパターン変数に該当したタプルの time を示している．次に，抽出したパターンオカレンスに対して次元の粒度を切り替えながら様々な次元の組合せに対応した集約計算を実行することで集約分析を行うことができる．

sequence

sid	location	time
1	群馬	6:00
1	埼玉	7:00
1	東京	8:00
1	千葉	9:00
…	…	…

パターン認識
パターン(X Y)

パターン(X Y)のパターンオカレンス表

sid	X_lo	Y_lo	X_time	Y_time
1	群馬	埼玉	6:00	7:00
1	群馬	東京	6:00	8:00
1	群馬	千葉	6:00	9:00
1	埼玉	東京	7:00	8:00
…	…	…	…	…

図 1: シーケンスデータに対するパターン認識実行結果

次に，抽出したパターンオカレンスに対して次元の粒度を切り替えながら様々な次元の組合せに対応した集約計算を実行することで集約分析を行うことができる．シーケンス OLAP で集約計算を行った結果は図 2 のようなデータキューブで表現することができる．図 1 のパターンオカレンス表に対して X_lo, Y_lo を集約対象の次元として該当するパターンオカレンスの数を集計した例を図 2 の左の図に示す．シーケンス OLAP では集約対象の次元を変化させるとそれに対応した集約結果を得ることができる．

さらに，シーケンス OLAP ではパターンを拡張することによって，より詳細な集約分析を行うことができる．図 2 では，パターン (X Y) をパターン (X W Y) に拡張する例を示している．ここで，パターン (X W Y) はある場所 X から Y への移動時に W を経由したパターンを表している．つまり，パターン (X Y) から (X W Y) に拡張することで，X から Y へ移動した集約結果に加えて，それらが経由した場所毎の集約結果を得ることができる．また，拡張後のパターンのパターンオカレンスは必ず拡張前のパターンのパターンオカレンスとしても出現するという性質を持つ．例えば，X_lo=群馬，W_lo=埼玉，Y_lo=東京となるパターン (X W Y) のパターンオカレンスは X_lo=群馬，Y_lo=東京となるパターン (X Y) のパターンオカレンスとしても出現する．この性質から，パターンには拡張前のパターン・パターンオカレンスを親，拡張後のパターン・パターンオカレンスを子とする親子関係が存在すると捉えることができる．シーケンス OLAP では拡張前・拡張後のパターンオカレンスの親子関係を利用することでパターン拡張時の集約計算を効率的に実行することができる．このパターン拡張時の効率的な集約計算をサポートするためには拡張前，拡張後の両方のパターンオカレンスの抽出とそれらの間の親子関係の検出を事前に行っておく必要がある．

このようにシーケンス OLAP では集約次元だけではなくパターンを切り替えることによって複雑な集約分析を効率的に行うことができる．しかし，既存のシーケンス OLAP 手法は独自の言語やシステムの開発を必要とし，データベースなどに蓄積されたデータとの統合的な処理が困難であるという問題点が存在する．

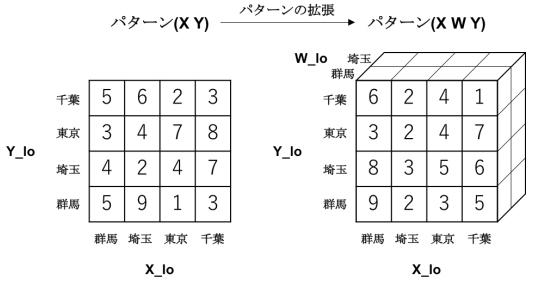

図 2: パターンオカレンス集約結果

2.2 MATCH_RECOGNIZE

汎用的なデータ管理システムである RDB では，蓄積されたリレーショナルデータに対して行間のパターン認識を実行することのできる SQL/RPR(Row Pattern Recognition) [1] が SQL2016 より標準化された．SQL/RPR では MATCH_RECOGNIZE 句にパターンを定義しパターン認識を実行することができる．

list1 に図 1 の sequence 表に対して MATCH_RECOGNIZE 句によるパターン認識を実行する SQL クエリの例を示す．MATCH_RECOGNIZE 句はパターン認識の対象となる表を指定する FROM 句の後ろに記述し，さらに内部でいくつかの句を使用する．まずは，PARTITION BY 句でシーケンスを形作るためのキーとなる属性を指定する．例では同じ sid を持つデータを 1 つのシーケンスとみなすために sid を指定している．次に，ORDER BY 句でシーケンスの順序を定める属性を指定する．例では time の順序に従うように指定している．説明の都合上，図 1 の sequence 表は既に time の順序でソートされたものを示している．次に，PATTERN 句によって検出したいパターンを正規表現で定義する．例では 2 つのパターン変数 X と Y からなるパターンを定義している．次に，DEFINE 句において PATTERN 句で定義した変数に対して条件を指定する．例では変数 Y に対して直前の行である X の location と異なる location を持つものをマッチさせるという条件を指定している．次に，MEASURES 句で抽出したパターンオカレンスに対して新しく生成する行の属性を定義する．例では X にマッチした行の location を X_lo，time を X_time，Y にマッチした行の location を Y_lo，time を Y_time として新たに定義している．なお，PARTITION BY 句で指定した属性はデフォルトで定義されるようになっている．この SQL を実行することで図 1 のパターンオカレンス表を得ることができる．しかし，MATCH_RECOGNIZE 句で定義できるパターンは 1 つに限定されており，拡張パターンのパターンオカレンスを抽出する場合は再度の問合せが必要となる．また，拡張前・拡張後のパターンオカレンス間の親子関係を検出する機能を持っていない．

つまり，MATCH_RECOGNIZE 句ではシーケンス OLAP のパターン拡張操作をサポートできないことがわかる．

list 1: MATCH_RECOGNIZE 句を用いた SQL

```
SELECT sid, X_lo, Y_lo, X_time, Y_time
FROM sequence
MATCH_RECOGNIZE(
        PARTITION BY sid
        ORDER BY time
        MEASURES X.location AS X_lo,
                 Y.location AS Y_lo,
                 X.time AS X_time,
                 Y.time AS Y_time
        PATTERN (X Y)
        DEFINE Y AS (Y.location != X.location)
);
```

3. 関 連 研 究

過去にはシーケンスデータに対してパターン認識を実行する手法が数多く研究されている．その多くは SASE [11] [4]，Cayuga [3]，Zstream [9] などの CEP(Complex Event Processing) として提案されている [12] [2] [5]．しかし，これらの手法は抽出したパターンオカレンスに対して OLAP 分析を実行する機能を持たない．

シーケンスデータに対して OLAP 分析を実行する手法として S-OLAP [8] が提案されている．S-OLAP はあらかじめ用意されたシーケンスデータに対してパターン認識を実行し，シーケンス OLAP を実行する手法である．S-OLAP は問合せたパターンオカレンスの情報を転置インデックスに保持する．さらに，拡張パターンの問合せの発生時に転置インデックスを利用し，効率的にパターン認識を実行することでパターン拡張操作をサポートしている．さらに，リアルタイムデータに対してシーケンス OLAP 分析を実行可能な手法として E-Cube [6] [7] が知られている．E-Cube は定義した親子パターンの集約結果を部分的に共有することでパターン拡張操作をサポートしている．しかし，S-OLAP，E-Cube はどちらも独自の言語とシステムの構築を必要とし，データベースなどに蓄積されたデータとの統合的な処理が困難であるという問題点が存在する．そこで，本研究では汎用的なデータ管理システムである RDB に蓄積されたシーケンスデータに対するシーケンス OLAP を実現することを目的とする．

4. MULTI_MATCH_RECOGNIZE

本研究では RDB(リレーショナルデータベース) に蓄積されたシーケンスデータに対するシーケンス OLAP を実現するために拡張記述 MULTI_MATCH_RECOGNIZE 句を提案する．MULTI_MATCH_RECOGNIZE 句ではシーケンス OLAP におけるパターン拡張操作に必要な複数パターンのパターン認識とパターンオカレンス間の親子関係検出を実行することができる．まずは，4.1 節でパターンと親子関係について定義する．次に，4.2 節で MULTI_MATCH_RECOGNIZE のシンタックスについて説明する．

4.1 パターンに関する定義

まずは，パターンを構成する要素について定義する．

定義 1 (パターンの要素) 以下の項目はパターンの要素である．

- パターン変数 *(*任意の文字列*)*
- パターンの要素のシーケンス
- パターンの要素を用いた正規表現

正規表現の修飾子としては以下のものを用いることができる．

- * (0 回以上の繰返し)
- + (1 回以上の繰返し)
- ? (0 回か 1 回の出現)
- {n} (n 回の繰返し)
- {n,m} (n 回から m 回までの繰返し)
- {n,} (n 回以上の繰返し)
- {,m} (m 回までの繰返し)
- | (alternative pattern)

次に，パターンについて定義する．

定義 2 (パターン) *1* つ以上のパターンの要素を並べたものはパターンである．ただし，同一パターン内に同じパターン変数を *2* つ以上指定することはできない．

次に，パターンの親子関係について定義する．

定義 3 (パターンの親子関係) あるパターン P の前後またはパターンの要素間に新たにパターンの要素を挿入するという操作を有限回繰り返すことによって, パターン C が構築可能な場合，P を親パターン，C を子パターンとする親子関係が成り立つ．

図 3 にパターン (X Y) に対する子パターンや孫パターンの例を示す．パターンの親子関係は親パターンをルートとする木構造で表現することができる．子パターンを新たに定義する場合，他の子パターンで既に使用されているパターン変数を追加するパターンの要素として用いることはできない．

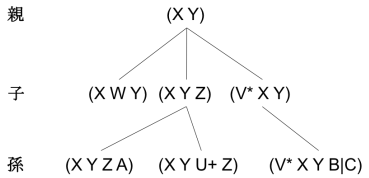

図 3: パターンの親子関係

次にパターンオカレンスの親子関係を定義する．

定義 4 (パターンオカレンスの親子関係) 親パターンを P，子パターンを C とした時，P と C で共通して存在するパターンの要素の集合を $P' = \{p_1, p_2, \ldots, p_n\}$，$C' = \{c_1, c_2, \ldots, c_n\}$ とする．また，親パターン P のパターンオカレンスを O_P，子パターン C のパターンオカレンスを O_C とした時，O_P のうち

パターンの要素 p_i にマッチしたタプルの集合を $O_{P,i}$，O_C のうちパターンの要素 c_i にマッチしたタプルの集合を $O_{C,i}$ とする．この時，1 から n までの全てで $O_{P,i} = O_{C,i}$ が成り立つ場合，O_P を親パターンオカレンス，O_C を子パターンオカレンスとする親子関係が成り立つ．

4.2 MULTI_MATCH_RECOGNIZE クエリ

RDB に蓄積されたシーケンスデータに対して複数パターンのパターン認識とパターンオカレンス間の親子関係検出をサポートできる拡張記述として MULTI_MATCH_RECOGNIZE 句の提案を行う．MULTI_MATCH_RECOGNIZE 句では MATCH_RECOGNIZE 句の PATTERN 句に代わり新たに提案される PATTERN SET 句を用いることで，複数パターンを定義することができる．list2 にクエリの例を示す．

list 2: MULTI_MATCH_RECOGNIZE 句を用いた SQL

```
SELECT *
FROM sequence
MULTI_MATCH_RECOGNIZE(
        PARTITION BY sid
        ORDER BY time
        MEASURES X.rid AS (X_rid; P1, P2, P3),
                W.rid AS (W_rid; P2),
                Y.rid AS (Y_rid; P1, P2, P3),
                Z.rid AS (Z_rid; P3),
                X.location AS (X_lo; P1, P2, P3),
                W.location AS (W_lo; P2),
                Y.location AS (Y_lo; P1, P2, P3)
                Z.location AS (Z_lo; P3)
        PATTERN SET (P1; X Y (P2; X W Y) (P3; X Y Z
              ))
        DEFINE Y AS (Y.location != X.location),
                W AS (W.location != X.location AND
                        W.location != Y.location),
                Z AS (Z.location != Y.location)
);
```

list2 は図 1 の sequence 表に対して，親パターン (X Y) と子パターン (X W Y)，(X Y Z) のパターン認識を実行するクエリである．例では PATTERN SET 句によってパターン (X Y) を P1，パターン (X W Y) を P2，パターン (X Y Z) を P2 として定義している．PATTERN SET 句では複数パターンを定義する際にパターンの親子関係を考慮して記述を行う．PATTERN SET 句は list3 のようなシンタックスで記述する．子パターンからさらにその子パターン (親パターンから見た孫パターン) をネストして記述することができる．PATTERN SET 句によって，パターンの親子関係は親パターンをルートとする木構造で表現することができる．

list 3: PATTERN SET 句のシンタックス

```
PATTERN SET (
        [親パターン名]; [親パターン] (
                [子パターン名]; [子パターン] ( [孫] … )
                [子パターン名]; [子パターン] ( [孫] … ) ...
        )
)
```

MULTI_MATCH_RECOGNIZE における MEASURES 句の指定では，属性毎に属性値を出力するパターンを指定する．例では X，Y に関連する属性は P1，P2，P3 共に出力するように指定されている．W に関連する属性は P1 と P3 が W を持たないため P2 のみ出力するように指定されている．

MULTI_MATCH_RECOGNIZE では後にパターンオカレンスの親子関係検出を実行するために，それぞれの行の連番を示す rid と呼ばれる属性を追加する．親子パターンの全ての共通部分において，マッチした行の rid が一致していれば親子関係とみなす．を図 4 に rid 追加後の sequence 表を示す．さらに図 4 の表に対してパターン認識と親子関係検出を実行すると図 5 のような結果が得られる．パターンオカレンス表には PARTITION BY 句で指定した属性の他に主キーとなる oid，そのパターンオカレンスがマッチしたパターンを示す pid，抽出したパターンオカレンスの連番である oid(occurrence id) がデフォルトで出力される．図 5 では，P1 にマッチしたパターンオカレンスの連番が P1_oid，P2 にマッチしたパターンオカレンスの連番が P2_oid，P3 にマッチしたパターンオカレンスの連番が P3_oid として出力されている．さらに，子パターンオカレンスは親パターンオカレンスの (パターン名)_oid を持つ．例えば，3 行目に格納されている P2 のパターンオカレンスは親パターン P1 との共通部分 X・Y の rid が 2 行目の P1 のパターンオカレンスと一致しているため，そのパターンオカレンスと親子関係を持つ．従って，3 行目のパターンオカレンスは親パターンオカレンスである 2 行目の P1_oid=2 を持つ．

rid	sid	location	time
1	1	群馬	6:00
2	1	埼玉	7:00
3	1	東京	8:00
4	1	千葉	9:00
	…	…	…

図 4: rid 追加後の sequence 表

パターンオカレンス表

oid	sid	X_rid	W_rid	Y_rid	Z_rid	X_lo	…	pid	P1_oid	P2_oid	P3_oid
1	1	1		2		群馬	…	P1	1		
2	1	1		3		群馬	…	P1	2		
3	1	1	2	3		群馬	…	P2	2	1	
4	1	1		2	3	群馬	…	P3	1		1
5	1	2		3		埼玉	…	P1	3		
6	1	1		4		群馬	…	P1	4		
7	1	1	2	4		群馬	…	P2	4	2	
…	…	…	…	…		…	…	…	…	…	

図 5: MULTI_MATCH_RECOGNIZE クエリ実行結果

5. 問合せ処理手法

本章では，図 4 のような表から図 5 のような表を生成するための問合せ処理手法の提案を行う．図 5 ような表を生成するためには，複数パターンのパターン認識と親子関係検出を実行する必要がある．提案手法では，NFA(非決定性有限オートマトン) を用いて複数パターンのパターン認識を実行する．さらに，パターン毎に管理したパターンオカレンス間の結合処理によっ

て親子関係検出を実行する．また，これらの処理を実行する場合に，パターンオカレンスが抽出される度に逐次的に親子関係検出を実行する手法 (1-Phase 手法) とパターン認識終了後にまとめて親子関係検出を実行する手法 (2-Phase 手法) の 2 通りの手法が考えられる．

まず，5.1 節で問合せ処理に用いるデータ構造を説明する．次に，5.2 節で 2 つの手法 (1-Phase 手法と 2-Phase 手法) について説明する．

5.1 データ構造

提案手法ではパターン認識を実行する NFA と親子関係検出を実行する middle buffer を組合わせたデータ構造を用いる．まずは，5.1.1 節で NFA について説明する．次に，5.1.2 節で middle buffer について説明する．

5.1.1 NFA(非決定性有限オートマトン)

NFA(非決定性有限オートマトン) [10] は有限オートマトンの 1 種であり，遷移先が一意に決定しない状態を持つことができるものである．NFA は以下のように定義される．

$$A = (Q, E, q_0, F)$$

- Q は状態の集合である．
- E は状態の遷移を示すエッジの集合である．
- q_0 は初期状態である．
- F は受理状態の集合である．

また，エッジは以下のように定義される．

$$e = (q_s, q_d, action, name, condition)$$

ここで q_s は遷移元の状態，q_d は遷移先の状態を示す．action は遷移時に実行される処理を示す．定義されている action については後述する．name は遷移条件の判定対象となるパターン変数名を示す．condition は name で指定されたパターン変数における条件を示す．

提案手法では親子パターンの共通部分を活用し，効率的に複数パターンのパターン認識を実行するために統一的な 1 つの NFA を作成する．具体的には，定義されたパターンの数だけ受理状態を持ち，親子パターンの間で先頭からの共通部分までを共有した NFA を作成する．例えば，親パターン P1(X Y) と 2 つの子パターン P2(X W Y)，P3(X Y Z) のパターン認識を実行する NFA は図 6 のようになる．図 6 では，P1(X Y) と P2(X W Y) は先頭からの共通部分が X であるため状態 X までが共有され，そこから先は分岐している．また，P1(X Y) と P3(X Y Z) は先頭からの共通部分が X，Y であるため状態 Y までが共有され，さらに P3 の状態 Z へ遷移するようになっている．

パターン認識は初期状態から開始される．エッジ間の遷移は行の入力によってトリガされる．指定された条件によっては複数の遷移条件に該当し，分岐が発生する．複数の遷移先が同時に見つかった場合はそれぞれに遷移した現在状態を複製する．各現在状態はそれぞれマッチした行の情報を格納する match buffer に関連付けられる．match buffer は初期状態では空であり，遷移時の action に応じてマッチした行へのポインタがパターン変数ごとに管理されている配列へ格納される．受理状態は各パターンに対応しており，遷移時に後述する take action が実行された場合にパターンが持つパターン変数のポインタを match buffer から後述する middle buffer へ出力する．

エッジに関連付けられた action はエッジの遷移時に実行される．action は SASE [11] [4] で定義された action を参考に以下のものを定義した．

take
: condition の条件を満たした行を match buffer に格納する．遷移先が受理状態であれば match buffer に格納されたポインタを出力する．

ignore
: condition の条件を満たした行をスキップする．

図 6 に図 4 の表に対して P1(X Y)，P2(X W Y)，P3(X Y Z) のパターン認識を実行する NFA の例を示す．図 6 の q_0 は初期状態，Y_1，Y_2，Z は受理状態を示す．各状態から伸びている線はエッジを表している．例えば，q_1 から X に遷移するエッジの各パラメータは $q_s = q_1$，$q_d = Y_1$，$action = take$，$name = y$，$condition = (Ey, y.location! = x.location)$ となる．また，図 6 の左下の表は match buffer を表している．図 6 の match buffer は図 4 の表の 1 ～ 3 行目までが入力され，$q_0 \to X \to W \to Y_2$ のように遷移した場合の例を示している．まず，初期状態 q_0 からパターン認識が開始され 1 行目入力時に遷移条件を満たすため状態が X へ遷移する．さらに，1 行目へのポインタが変数 X の ptr に格納される．図 6 で変数 X の ptr に格納されている x_1 は 1 行目が変数 X にマッチしたことを示している．次に，2 行目時に状態が W へと遷移すると，2 行目のポインタが変数 W の ptr に格納される．次に，3 行目入力時に状態が Y_2 へと遷移すると，3 行目のポインタが変数 Y の ptr に格納される．さらに take action により受理状態 Y_2 に遷移したため，match buffer に保持されている X，W，Y の ptr がパターンオカレンスとして middle buffer へ出力される．

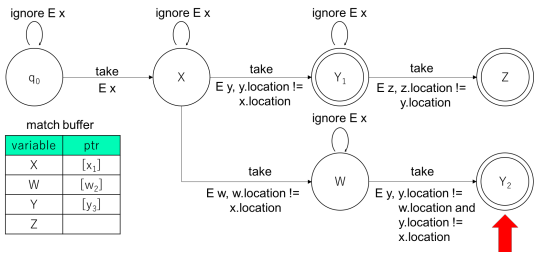

図 6: P1(X Y),P2(X W Y),P3(X Y Z) に対する NFA

5.1.2 middle buffer

図 7 は P1(X Y),P2(X W Y),P3(X Y Z) に対応する middle buffer を示した図である．各パターンに対応するバッファには抽出されたパターンオカレンスとパターンオカレンスの count

が格納される．新たなパターンオカレンス格納時に count の値がインクリメントし，さらにその値を (パターン名)_oid としてパターンオカレンスに付与する． また，middle buffer ではパターンの親子関係を木構造で表した時に枝で繋がっている親子のバッファに対して結合処理を行い，パターンオカレンスの親子関係を検出する．図 7 の例では，P1 と P2，P1 と P3 が枝で繋がっているため親子関係検出時には 2 回の結合処理が実行される．結合処理はより上位のノードを含むものから順番に実行する．これにより，直接枝で繋がっていない親パターンオカレンスの親子関係を検出することが可能となる．例えば，P1(X Y)，P2(X W Y)，P4(X W V Y) の 3 つのパターンが定義されていた場合，P4 は P2 の他に枝で繋がっていない P1 と親子関係を持つ．この場合は，上位ノードである P1 と P2 の結合処理が先に実行される．この時，結合処理によって親子関係が検出されると P2 のバッファに検出された P1_oid が格納される．従って，P2 と P4 の結合処理で P4 のバッファに P1 と結合処理を行うことなく親子関係を持つ P1_oid を格納することができる．

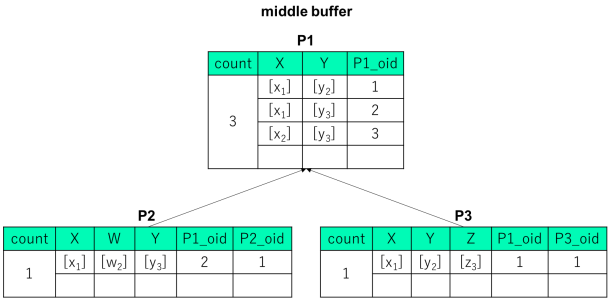

図 7: P1(X Y)，P2(X W Y)，P3(X Y Z) の middle buffer

5.2 パターン認識と親子関係検出

問合せ処理では複数パターンのパターン認識と親子関係の検出を実行する．その際に，パターンオカレンスが抽出される度に逐次的に親子関係検出を実行する手法 (1-Phase 手法) とパターン認識終了後にまとめて親子関係検出を実行する手法 (2-Phase 手法) の 2 通りの手法が考えられる．まずは，5.2.1 節で 1-Phase 手法について説明する．次に，5.2.2 節で 2-Phase 手法について説明する．

5.2.1 1-Phase 手法

1-Phase 手法はパターンオカレンスが抽出される度に逐次的に親子関係検出を実行する手法である．1-Phase 手法では行の入力毎にその時点で middle buffer に格納されているパターンオカレンスのみを対象として結合処理を実行し親子関係を検出する．また，新たな行が入力される前に middle buffer のパターンオカレンスを図 5 のようなパターンオカレンス表に出力し，middle buffer から削除する．このように，行の入力の度に middle buffer の中身を count を除いてリセットすることで逐次的に親子関係の検出を実行することができる．

しかし，1-Phase 手法では全ての親子関係を結合処理のみで検出することはできない．1-Phase 手法において結合処理で検出できるのは P1(X Y) と P2(X W Y) のようにパターンの最後の要素が共通している親子関係 (このような親子関係を backward match pattern と呼ぶ) に限られる．対して，P1(X Y) と P3(X Y Z) のようにパターンの最後の要素が共通していない親子関係 (このような親子関係を forward match pattern と呼ぶ) は結合処理によって検出できない．何故ならば，P3 のパターンオカレンスと親子関係を持つ P1 のパターンオカレンスは P3 のパターンオカレンスが抽出された時には既に middle buffer から削除されているからである．

そこで，1-Phase 手法では forward match pattern の親子関係を検出するために match buffer と middle buffer を拡張する．図 8 と図 9 に拡張した match buffer と middle buffer を示す．1-Phase 手法では match buffer に子パターンを持つパターンの (パターン名)_oid を保持する領域を加える．さらに，middle buffer において子パターンを持つパターンのバッファにパターンオカレンスを出力した match buffer のポインタを格納する buf_ptr という領域を加える．子パターンを持つパターンのバッファでは (パターン名)_oid を取得した後にその値を buf_ptr に記録されている match buffer へ出力する．この 2 つの拡張によって，match buffer に保持した (パターン名)_oid から forward match pattern の親子関係を検出することが可能となる．

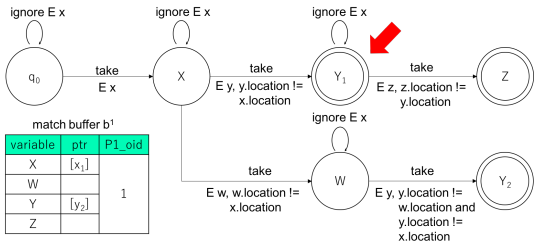

図 8: 1-Phase 手法で用いる match buffer

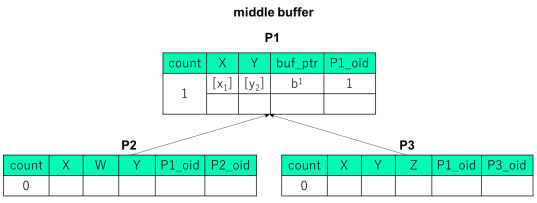

図 9: 1-Phase 手法で用いる middle buffer

以下に図 8，図 9，図 10，図 11 を用いて例を示す．図 8 は図 4 の 2 行目までが入力され $q_0 \rightarrow X \rightarrow Y_1$ と遷移した状態を示している．また，図 9 は 2 行目までが入力された時の middle buffer を示している．Y_1 に遷移することによって match buffer b^2 から P1 のパターンオカレンスが出力される．その際に b^2 へのポインタも共に出力され buf_ptr に記録される．$P1\_oid = 1$ を取得後，buf_ptr に記録されたポインタをたどって b^2 に $P1\_oid = 1$ が出力される．3 行目が入力される前に middle buffer に格納されているパターンオカレンスはパターンオカレンス表へと出力される．次に 3 行目が入力されて $Y_1 \rightarrow Z$ のように遷移すると match buffer は図 10 のようになる．Z は

受理状態であるため P3 のバッファにパターンオカレンスが出力されるが，その際に match buffer から $P1\_oid = 1$ も共に出力され，middle buffer は図 11 のようになる．このようにして結合処理で検出できない forward match pattern の親子関係を検出することができる．

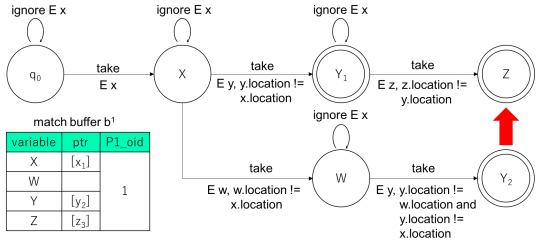

図 10: 1-Phase 手法で用いる match buffer

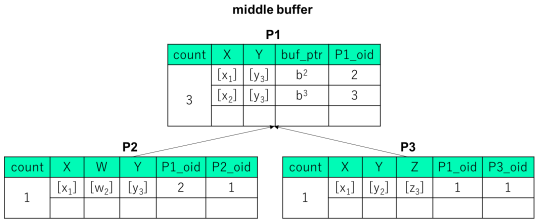

図 11: 1-Phase 手法で用いる middle buffer

5.2.2 2-Phase 手法

2-Phase 手法はパターン認識の終了後にまとめて親子関係の検出を行い，結果を出力する手法である．2-Phase 手法は 1-Phase 手法と異なり，行の入力毎に middle buffer に格納されたパターンオカレンスを削除しないため，図 7 ように全てのパターンオカレンスが蓄積されていく．全ての行が入力され，パターン認識が終了すると 5.1.2 節で説明したように結合処理によって親子関係が検出され，図 5 のような結果が出力される．

6. 実　　験

本実験では 5 章で提案した 2 つのパターン認識手法の比較実験を行う．あらかじめ用意された実験用の人工データが格納されたテーブルに対して 1-Phase 手法，2-Phase 手法を用いて問合せ処理を行い，結果が出力されるまでの処理時間を計測する．人工データは全て同じシーケンスのデータとする．つまり，複数のシーケンスデータではなく 1 つのシーケンスデータに対するパターン認識の処理時間を計測する．実験 1 は定義するパターンの数を固定し，人工データの行数を変化させ処理時間を計測する．また，実験 2 は人工データの行数を固定し，定義するパターンの数を変化させ処理時間を計測する．各実験ではパターンオカレンスの最大長は 10 に制限している．処理時間は各パラメータで 10 回ずつ計測を行い，その平均を用いる．実験環境は表 1 に示す通りである．

表 1: 実験環境

OS	CentOS 7
CPU	Intel(R) Xeon(R) CPU 3.00GHz
Memory	32GB
使用言語	Python 3.4.4

6.1 実　験　1

実験 1 では定義するパターンの数を固定し，人工データの行数を変化させ処理時間を計測する．list4 に示すように (X Y)，(X W Y)，(X Y Z) の 3 つのパターンのパターン認識を 1-Phase 手法，2-Phase 手法を用いて実行する．また，人工データの行数は 10 から 100000 までの間で変化させる．図 12 にその結果を示す．図 12 の横軸は人工データの行数，右の縦軸は処理時間示している．また，左の縦軸は結果に出力されたパターンオカレンス数を示している．実験結果から，パターン認識終了後にまとめて親子関係の検出を行う 2-Phase 手法の方が処理時間が短いことがわかる．

list 4: 実験 1 で用いたクエリ

```
SELECT *
FROM fact
MULTI_MATCH_RECOGNIZE(
        PARTITION BY pid
        ORDER BY time
        PATTERN SET (P1; X Y (P2; X W Y) (P3; X Y Z
            ))
        DEFINE Y AS (Y.location != X.location and
                        Y.rid − X.rid <= 10),
               Z AS (Z.location != Y.location and
                        Z.rid − X.rid <= 10),
               W AS (W.location != X.location and
                        W.location != Y.location and
                        W.rid − X.rid <= 10)
);
```

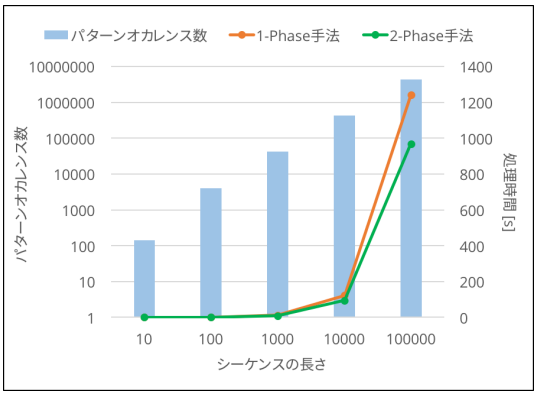

図 12: 実験 1 による処理時間の比較

6.2 実　験　2

実験 2 では人工データの行数を固定し，定義するパターンの数を変化させ処理時間を計測する．人工データの行数は 10000 行とする．

図 13 に実験 2 の結果を示す．横軸は定義したパターンの数，

右の縦軸は処理時間示している．また，左の縦軸は結果に出力されたパターンオカレンス数を示している．図 13 から，実験 1 と同様に 2-Phase 手法の方が処理時間が短いことがわかる．

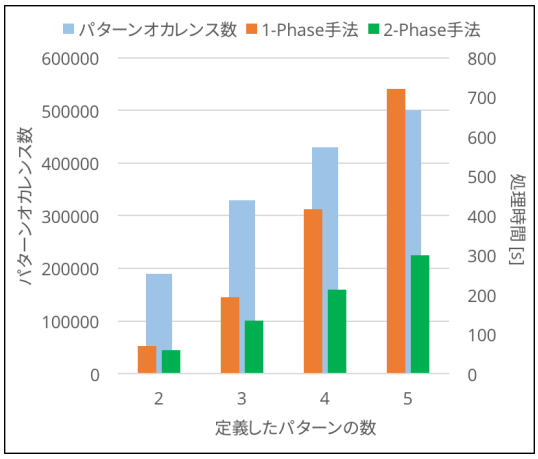

図 13: 実験 2 による処理時間の比較

7. おわりに

本研究では RDB 上でのシーケンス OLAP におけるパターン拡張処理に必要な複数パターンのパターン認識とパターンオカレンス間の親子関係検出をサポートする拡張記述 MULTI_MATCH_RECOGNIZE とその問合せ処理手法を提案した．問合せ処理手法はパターンオカレンスが抽出される度に逐次的に親子関係検出を実行する 1-Phase 手法とパターン認識終了後にまとめて親子関係検出を実行する 2-Phase 手法の 2 つを提案した．人工データを用いた比較実験の条件下では，2-Phase 手法の方が高速に処理を実行できることがわかった．今後の課題として，シーケンスデータの性質やメモリサイズなどを変化させ，より詳細な比較実験を行いそれぞれの手法のメリット・デメリットを考察する予定である．

謝　辞

本研究の一部は, 理研「実社会ビッグデータ利活用のためのデータ統合・解析技術の研究開発」による．

文　献

[1] ISO/IEC 2016. Information technology — database languages — sql technical reports — part 5:row pattern recognition in sql. Technical report, ISO copyright office, 2016.

[2] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. Ep-sparql: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pages 635–644. ACM, 2011.

[3] Alan J Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, Walker M White, et al. Cayuga: A general purpose event monitoring system. In *CIDR*, volume 7, pages 412–422, 2007.

[4] Daniel Gyllstrom, Jagrati Agrawal, Yanlei Diao, and Neil Immerman. On supporting kleene closure over event streams. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1391–1393. IEEE, 2008.

[5] Ilya Kolchinsky, Izchak Sharfman, and Assaf Schuster. Lazy evaluation methods for detecting complex events. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pages 34–45. ACM, 2015.

[6] Mo Liu, Elke Rundensteiner, Kara Greenfield, Chetan Gupta, Song Wang, Ismail Ari, and Abhay Mehta. E-cube: Multi-dimensional event sequence processing using concept and pattern hierarchies. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 1097–1100. IEEE, 2010.

[7] Mo Liu, Elke Rundensteiner, Kara Greenfield, Chetan Gupta, Song Wang, Ismail Ari, and Abhay Mehta. E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 889–900. ACM, 2011.

[8] Eric Lo, Ben Kao, Wai-Shing Ho, Sau Dan Lee, Chun Kit Chui, and David W Cheung. Olap on sequence data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 649–660. ACM, 2008.

[9] Yuan Mei and Samuel Madden. Zstream: a cost-based query processor for adaptively detecting composite events. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 193–206. ACM, 2009.

[10] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.

[11] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM, 2006.

[12] Haopeng Zhang, Yanlei Diao, and Neil Immerman. Recognizing patterns in streams with imprecise timestamps. *Proceedings of the VLDB Endowment*, 3(1-2):244–255, 2010.