DEIM Forum 2019 H4-3

# シーケンスOLAPのための複数行パターンマッチング問合せ処理

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，シーケンスデータ上の特定のパターンの出現箇所 (パターンオカレンス) を抽出し，集約次元の粒度を切換える drill-down・roll-up などの OLAP 演算や，パターンを切換える pattern-drill-down・pattern-roll-up などの OLAP-Pattern 演算を使用して多次元集約を実行する集約分析技術である．最も汎用的な DB の 1 つである RDB 上でシーケンス OLAP の OLAP-Pattern 演算を効率的に処理するためには，異なる複数のパターンについての行パターンマッチング結果をあらかじめ抽出しておく必要がある．一般に，このような複数パターンの行パターンマッチングはそれぞれ個別に実行されるが，それらのパターン同士が共通部分を持つ場合に冗長な処理が発生する．そこで，RDB 上でシーケンス OLAP の OLAP-Pattern 演算をサポートするために，複数行パターンマッチングとパターンオカレンスの親子関係検出を効率的に処理するパターン階層モデルと問合せ処理手法を提案する．さらに，RDB の UDF として提案手法を実装し，人工データを用いた実験で提案手法の性能の検証を行う．

**キーワード**　時系列データ処理，SQL/RPR，OLAP

## 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データベースに蓄積されている．特に，RFID による移動ログデータや株取引データなど，特定の順序に従って生成されるデータをシーケンスデータと呼ぶ．そのようなシーケンスデータに対して，データの並びのパターンを照合し，特定パターンの発生箇所 (パターンオカレンス) を抽出・集約する分析技術が求められている．

データベースに蓄積されたシーケンスデータに対するパターンオカレンスの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パターンオカレンスを抽出し，データの次元の階層構造に基づいて集約条件を変更する drill-down や roll-up などの OLAP 演算を実行することで多次元集約分析を行うことができる．また，シーケンス OLAP 分析では集約結果を取得後，現在のパターンとは異なるパターンの集約結果を確認したいという要求がある．このような要求に対して，シーケンス OLAP では pattern-drill-down や pattern-roll-up などの集約対象パターンを切換える OLAP-Pattern 演算を実行することで，異なるパターンの集約結果を取得することができる．OLAP-Pattern 演算を実行するためには，複数のパターンについてパターンマッチングを行い，それらのパターンオカレンス間の親子関係を検出する必要がある．一般に，このような複数パターンのパターンマッチングはそれぞれ個別に実行されるが，それらのパターン同士が共通部分を持つ場合に冗長な処理が発生する．

そこで本研究では，最も汎用的なデータベースの 1 つである RDB(関係データベース) 上でのシーケンス OLAP における OLAP-Pattern 演算をサポートするために，複数行パターンマッチング問合せ処理手法を提案する．提案手法では，拡張前のパターンを親，拡張後のパターンを子とするパターンの階層構造を定義し，複数パターンを統合した NFA によって複数パターンの行パターンマッチングを実行する．さらに，パターンオカレンス抽出時に逐次的にパターンオカレンス間の親子関係を検出する．提案手法によって，複数パターンの行パターンマッチングとパターンオカレンス間の親子関係検出を効率的に実行することができる．本研究では，RDB の UDF(ユーザー定義関数) として提案手法を実装し，人工データを用いた実験で提案手法の性能の検証を行う．

## 2　シーケンス OLAP

シーケンス OLAP はデータベースに蓄積されたシーケンスデータに対するパターンオカレンスの集約分析手法である．本研究では，最も汎用的なデータベースの 1 つである RDB に蓄積されたシーケンスデータを処理対象として考える．シーケンス OLAP はシーケンスデータからパターンオカレンスを抽出し，データの次元の階層構造に基づいて集約条件を変更する OLAP 演算を活用して多次元集約分析を実行する．シーケンス OLAP に関わる表の例を図 1 に示す．シーケンス表には，パーソントリップデータとして sid(シーケンス ID)，city，venue，time を持つタプルが格納されている．シーケンス OLAP ではまず，シーケンスデータに対してパターンを指定して行パターンマッチングを行い，パターンオカレンスを抽出する。パターンは (X Y) などのいくつかの変数 (パターン変数と呼ぶ) の並びで表現される．また，パターンで指定されている各パターン変数に対して条件を指定することができる．例えば，X.venue=Airport，Y.venue=Airport の条件を指定したパターン (X Y) は Airport

シーケンス表

| sid | city | venue | time |
|---|---|---|---|
| 1 | Vancouver | Airport | 2019-01-04 10:20:00 |
| 1 | New York | Airport | 2019-01-04 16:35:00 |
| 1 | New York | Hotel | 2019-01-04 17:05:00 |
| 1 | New York | Restaurant | 2019-01-04 18:00:00 |
| 1 | New York | Airport | 2019-01-05 07:15:00 |
| 1 | London | Airport | 2019-01-05 14:10:00 |
| 1 | Paris | Airport | 2019-01-05 22:30:00 |
| 1 | Paris | Hotel | 2019-01-05 23:05:00 |
| … | … | | … |

パターンオカレンス表 : (X Y)

| sid | X_city | Y_city | X_venue | Y_venue | travel_hour |
|---|---|---|---|---|---|
| 1 | Vancouver | New York | Airport | Airport | 6 |
| 1 | Vancouver | New York | Airport | Airport | 21 |
| 1 | Vancouver | London | Airport | Airport | 28 |
| 1 | Vancouver | Paris | Airport | Airport | 36 |
| … | … | … | … | … | … |
| 1 | New York | New York | Airport | Airport | 15 |
| 1 | New York | London | Airport | Airport | 22 |
| … | … | … | … | … | … |

次元表 : 都市毎の地域区分

| city | world_region |
|---|---|
| Vancouver | North America |
| New York | North America |
| London | Europe |
| Paris | Europe |
| … | … |

図 1: シーケンス OLAP で用いる表

から Airport への移動パターンを表す．このパターンについて行パターンマッチングが実行されると図 1 のパターンオカレンス表が出力される．

次に，出力されたパターンオカレンス表に対して集約処理が実行される． シーケンス OLAP の集約分析の結果は，図 2 に示すようなデータキューブで表現することができる．図 2 の左のキューブは (X_city，Y_city，travel_hour) の属性値によるカウントのクロス集計結果を示している．このような集約結果に対して，次元表を結合することによって集約の粒度を変更する OLAP 演算 (roll-up，drill-down) を実行することが可能となる． 例えば，図 1 のパターンオカレンス表の X_city と次元表の city を結合することで，同じ region の属性値を持つ集約結果をまとめることができる．このような集約の粒度を粗くする演算を roll-up と呼び，反対に，集約の粒度を細くする演算を drill-down と呼ぶ．図 2 では，左のデータキューブを roll-up(X_city → X_region）して，右のデータキューブを取得している．

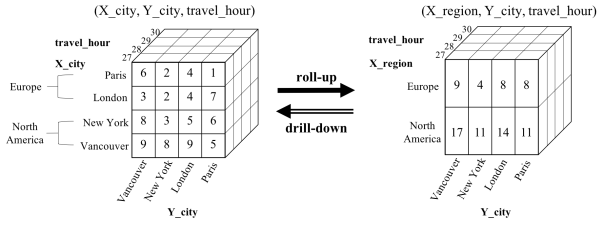


図 2: シーケンス OLAP のデータキューブ

次に，シーケンス OLAP では集約の次元だけでなく，パターンそのものを変更する機能も必要となる．例えば，Airport(X) から Airport(Y) への移動パターンを分析した後に，さらに，その間に会場 W を通過したパターンを分析するというシナリオが考えられる．このような要求に応えるためには，パターン (X Y) とは別にパターン (X W Y) のパターンオカレンスを抽出しておく必要がある．本研究では特に，2 つのパターンについて，片方のパターンに新たなパターン変数を追加することで，もう片方のパターンが得られる場合に，元のパターンを親パターン，新たなパターンを子パターンとする．このように，集約対象のパターンを異なるパターンに切換える演算を OLAP-Pattern 演算と呼ぶ．特に，(X Y) → (X W Y) のように親パターンから子パターンに切換える演算を pattern-drill-down と呼び，(X W Y) → (X Y) のように子パターンから親パターンに切換える演算を pattern-roll-up と呼ぶ．

さらに，シーケンス OLAP ではパターンを切換えるだけではなく，特定のパターンオカレンスについて異なるパターンにおける詳細な分析を行いたいという要求がある．例えば，図 3 に示すようにパターン (X Y) について集約結果を確認した後に，New York の Airport から New York の Airport へ移動したパターンオカレンスについて，その間に通過した会場 W を分析するというシナリオを考える．このような要求に応えるためには，パターン (X W Y) のパターンオカレンスの中から条件に合致するもの ({X_city, Y_city}=New York) のみを抽出する必要がある．図 3 の右側の表からは，New York の Airport から New York の Airport へ移動する間に，Hotel と Restaurant の両方を通過したシーケンス (sid=1)，どこも通過していないシーケンス (sid=2)，Cafe を通過したシーケンス (sid=3) が存在することが分かる．このように，特定のパターンオカレンスについて異なるパターンにおける詳細な集約結果を得る演算を pattern-occurrence-drill-down と呼ぶ．

aggregation result of (X Y)

| X_city | Y_city | X_venue | Y_venue | count |
|---|---|---|---|---|
| Vancouver | New York | Airport | Airport | 5 |
| Vancouver | London | Airport | Airport | 2 |
| … | … | … | … | … |
| New York | New York | Airport | Airport | 3 |
| … | … | … | … | … |

pattern occurrences of (X W Y)
X_city=New York, Y_city=New York

| sid | … | X_venue | W_venue | Y_venue |
|---|---|---|---|---|
| 1 | … | Airport | Hotel | Airport |
| 1 | … | Airport | Restaurant | Airport |
| 2 | … | Airport | | Airport |
| 3 | … | Airport | Cafe | Airport |

図 3: pattern-occurrence-drill-down

これらの機能を RDB 上で実現する方法について考察する．まず，RDB における行パターンマッチングは SQL2016 [1] で標準化されている MATCH_RECOGNIZE 句を用いた問合せで実行することができる．また，drill-down，roll-up を含む OLAP 分析機能も RDB で提供されている機能で実現することができる．次に，OLAP-Patten 演算について考える．OLAP-Patten 演算を実行するためには，親パターンと子パターンのそれぞれについてパターンオカレンスを抽出する必要がある．この時，MATCH_RECOGNIZE 句は複数のパターンの問合せを実行することができないため，複数回の問合せを行う必要がある． 次に，pattern-occurrence-drill-down について考える．pattern-occurrence-drill-down を効率的に実行するためには，同一の発生箇所を示すパターンオカレンス同士を異なるパターンの間でリンクさせる必要がある．しかし，RDB はそのよう

な参照を検出して保持する機能を持たない．そこで本研究では，RDB を用いてシーケンス OLAP における OLAP-Pattern 演算をサポートするために，複数パターンの行パターンマッチングとパターンオカレンス間の親子関係検出を効率的に実行する手法を提案する．

## 3 パターン階層モデル

本章では，パターンの親子関係からなるパターン階層モデルについて定義する．我々は，あるパターン (例:(X Y)) とそれに新しいパターン変数を追加した拡張パターン (例:(X W Y)) がある場合，拡張パターンのパターンオカレンスは元のパターンの条件も満たしているという性質に着目した．この性質から，元のパターンが親，拡張パターンが子となるような親子関係を定義する．まず，パターンを構成するパターン要素を定義する．

［定義 1］ (**パターン要素**) **パターン要素** $p$ はパターン変数 $v$(1 文字)，または，パターン要素からなる正規表現である．正規表現は以下のものを考える．

- シーケンス (denoted by $p_1 \dots p_n$)
- クリーネ閉包 (denoted by p*)
- 選択 (denoted by $p_1|p_2$)

ここで，$p$ または $p_i$ パターン要素であり, 各パターン要素には同じパターン変数が複数回出現することはない．特に，シーケンスのパターン要素のことを**シーケンスパターン要素**と呼ぶ．

［定義 2］ (**パターン**) **パターン** $P$ は 1 つのシーケンスパターン要素である．ここで，パターンを $P = \{p_1 \dots p_n\}$ と表す．

［定義 3］ (**パターンの親子関係**) もし，パターン $P_{child}$ がパターン $P_{parent} = \{p_1, p_2, \dots, p_n\}$ に対して，新たなパターン要素 $p_{new}$ を前 ($P_{child} = \{p_{new}, p_1, p_2, \dots, p_n\}$)，間 ($P_{child} = \{p_1, p_2, \dots, p_{new}, \dots, p_n\}$)，後 ($P_{child} = \{p_1, p_2, \dots, p_n, p_{new}\}$) に挿入するという処理を有限回繰り返すことによって構築可能である場合，パターン $P_{child}$ は親パターン $P_{parent}$ の子パターンである．

［定義 4］ (**パターンの親子関係の分類**) 親パターン $P_{parent}$ とその子パターン $P_{child}$ が存在するとき，$P_{parent}$ と $P_{child}$ の最後のパターン要素が一致する場合，その親子関係を **suffix match case** に分類する．また，一致しない場合，その親子関係を **prefix match case** に分類する．

［例 1］ (**パターンの親子関係**) 図 4 は親パターン (X Y) に対する子パターンと孫パターンの例を示す図である．各パターンを結ぶ線に付随する文字はそれぞれの親子関係の分類を示している．子パターンを新しく定義するときは，別の子パターンですでに使用されているパターン変数を新しく追加するパターン要素として使用することはできない．例えば，以下の (X W Y) の子パターンを定義する際に，すでに他のパターンで使用されている Z や V などを用いたパターンである (X W Y Z) や (V X W Y) などは定義することができない．

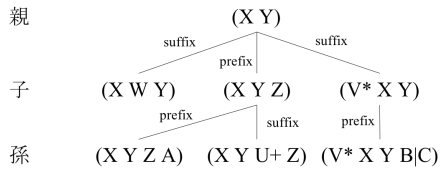


図 4: パターンの親子関係

複数行パターンマッチングを実現するために，上記のパターンおよびパターン階層の定義を行のシーケンスとして関係モデルの表に適用する． さらに，複数行パターンマッチングに以下の表現を用いる．

- パターン $P$ にパターン変数 $\{v_1, \dots, v_m\}$ が出現する場合，$pattern\_variables(P) = \{v_1, v_2, \dots, v_m\}$ と表す．
- パターン変数 $v_i$ に一致した行の集合 $R_i$ を $R_i = \{r_1, \dots, r_k\}$ と表す．

［定義 5］ (**パターンオカレンス**) $pattern\_variables(P) = \{v_1, \dots, v_m\}$ であるパターン P のパターンオカレンス $o_P$ を $o_P = \{R_1, \dots, R_m\}$ とする．

［定義 6］ (**パターンオカレンスの親子関係**) 親パターン $P_{parent}$ のパターンオカレンス $o_{P_{parent}}$ と子パターン $P_{child}$ のパターンオカレンス $o_{P_{child}}$ に対して，$\forall R_i, R_i \in o_{P_{parent}} \Rightarrow R_i \in o_{P_{child}}$ が成り立つ時，$o_{P_{child}}$ は $o_{P_{parent}}$ の子パターンオカレンスである．

［例 2］ (**パターンオカレンスの親子関係**) 図 4 に示されている (X Y) のパターンオカレンスと (X W Y) のパターンオカレンスは，いずれも図 1 のシーケンス表の 2 行目が X，5 行目が Y に一致しているため，それぞれ親子関係を持つ．

## 4 問合せ言語

本章では，複数行パターンマッチングを実行するための問合せ言語について説明する．SQL2016 で標準化されている MATCH_RECOGNIZE 句では定義できるパターンが 1 つに限られており，複数パターンの行パターンマッチングを行うためには複数回の問合せが必要となる． そこで，MATCH_RECOGNIZE 句をベースとして，複数行パターンマッチングとパターンオカレンス間の親子関係検出が実行可能な MULTI_MATCH_RECOGNIZE 句を提案する．MULTI_MATCH_RECOGNIZE 句の構文を以下に示す．

リスト 1: MULTI_MATCH_RECOGNIZE 句のシンタックス

```
1 SELECT <select list>
2 FROM <table name>
3 MULTI_MATCH_RECOGNIZE (
4     PARTITION BY <column names>
```

```
    ORDER BY <column names>
    MEASURES <measures list>
    PATTERN SET <define patterns>
    DEFINE <define pattern variables>
    WINDOW <window param>
);
```

PARTITION BY 句ではシーケンスを形成するための識別子となる属性を指定する．ORDER BY ではシーケンス内での順序の基準となる属性を指定する．MEASURES 句では出力されるパターンオカレンスが持つ属性を定義する．PATTERN SET 句ではパターンを指定する．DEFINE ではそれぞれのパターン変数の条件を指定する．WINDOW 句では抽出するパターンオカレンスの最大長を指定する．また，図 1 のシーケンス表に対するクエリの例をリスト 2 に示す．

リスト 2: MULTI_MATCH_RECOGNIZE 句を用いたクエリ

```
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.city AS (X_city; P1, P2, P3),
        W.city AS (W_city; P2),
        Y.city AS (Y_city; P1, P2, P3),
        Z.city AS (Z_city; P3)
    PATTERN SET (P1; X Y (P2; X W Y) (P3; X Y
        Z))
    DEFINE X AS (X.venue = "Airport"),
        W AS (W.city = Xcity),
        Y AS (Y.venue = "Airport"),
        Z AS (Z.city = Y.city)
    WINDOW [Rows 5]
);
```

リスト 2 は親パターン (X Y) と 2 つの子パターン (X W Y), (X Y Z) の複数行パターンマッチングを実行するためのクエリである．MEASURES 句では各属性に対して，その属性を出力するパターンを指定する．リスト 2 では，全てのパターンで X, Y に関連する属性が出力されるように指定されている．また，P1 と P3 はパターン変数 W を持たないため，W に関連する属性は P2 のみ出力するように指定されている．WINDOW 句で指定されている [Rows 5] という記述は，最初に一致した行から 5 行未満の範囲に出現するパターンオカレンスのみを抽出する条件を表している．

MULTI_MATCH_RECOGNIZE 句の問合せ処理では，パターンオカレンスの親子関係検出を実行するために，複数行パターンマッチング実行前に行番号として rid という属性を追加する．rid を追加したシーケンス表に対して複数行パターンマッチングと親子関係検出を実行すると，図 5 のパターンオカレンス表のような結果が得られる．パターンオカレンス表の **oid**(パターンオカレンス表の主キーとなる属性)，**sid**(PARTITION BY 句で指定された属性)，**pid**(一致したパターンのパターン名を示す属性)，**pattern oid**(各パターン内での id となる属性) はデフォルトで出力される．

図 6 では，P1 に一致したパターンオカレンスの id が P1_oid として出力される． 同様に，P2 と P3 のパターンオカレンスでは p2 _oid と p3_oid が出力される．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 pattern oid を持つ．例えば，6 行目の P2 のパターンオカレンス ($oid = 6$) は 5 行目の P1 のパターンオカレンス ($oid = 5$) を親パターンオカレンスとして持つ．従って，パターンオカレンス ($oid = 6$) は親パターンオカレンスの P1_oid として $P1\_oid = 3$ を持つ．

パターンオカレンス表 : P1 (X Y), P2 (X W Y), P3 (X Y Z)

| oid | sid | X_rid | W_rid | Y_rid | Z_rid | X_city | W_city | … | pid | P1_oid | P2_oid | P3_oid |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | | 2 | | Vancouver | | | P1 | 1 | | |
| 2 | 1 | 1 | | 2 | 3 | Vancouver | | | P3 | 1 | | 1 |
| 3 | 1 | 1 | | 2 | 4 | Vancouver | | … | P3 | 1 | | 2 |
| 4 | 1 | 1 | | 5 | | Vancouver | | | P1 | 2 | | |
| 5 | 1 | 2 | | 5 | | New York | | … | P1 | 3 | | |
| 6 | 1 | 2 | 3 | 5 | | New York | New York | | P2 | 3 | 1 | |
| 7 | 1 | 2 | 4 | 5 | | New York | New York | | P2 | 3 | 2 | |
| … | … | … | … | … | … | … | | … | … | … | … | … |

図 5: MULTI_MATCH_RECOGNIZE 句の問合せ結果

## 5 問合せ処理

本章では，図 5 のようなパターンオカレンス表を生成するための問合せ処理方法を提案する．パターンオカレンス表を生成するためには，複数行パターンマッチングと親子関係検出を実行する必要がある．提案手法では，複数のパターンを統合的に扱う NFA(Nondeterministic Finite Automaton) [11] を用いて複数行パターンマッチングを行う．さらに，パターンオカレンス抽出時に逐次的に親子関係検出を実行することで，効率的に問合せ処理を行う．

### 5.1 データ構造

提案手法では，複数行パターンマッチングを実行する NFA(MRPR-NFA) とパターン毎に抽出したパターンオカレンスを管理する pattern occurrence buffer を組合せたデータ構造を用いる．MRPR-NFA と pattern occurrence buffer は共に MULTI_MATCH_RECOGNIZE 句を用いたクエリをコンパイルすることによって構築する．まずは，MRPR-NFA について説明する．

#### 5.1.1 MRPR-NFA

MRPR-NFA は複数のパターンを統合的に扱い，複数行パターンマッチングを実行するためのオートマトンである．基本的な考え方は SASE [2] [5] を参考にした．MRPR-NFA は以下のように定義される．

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

ここで，$Q$ は状態の集合，$E$ はエッジの集合，$q_0$ は初期状態，$F$ は受理状態の集合を示す．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．

MRPR-NFA の評価は初期状態から始まる．状態の遷移は

**Algorithm 1** MRPR-NFA 構築

**Require:** pattern set PS, conditions from DEFINE and WINDOW C, initial state $q_0$

$Q \leftarrow \{q_0\}$, $E \leftarrow \emptyset$, $F \leftarrow \emptyset$
$e \leftarrow newEdge(q_0, q_0, "ignore", None, conditions)$
$E.append(e)$
$rootNode \leftarrow newNode(q_0)$
**for all** $P \in PS$ **do**
  $node \leftarrow rootNode$
  **for all** $p \in P$ **do**
    **if** $p$ not in $node.children$ **then**
      $node.children[p] \leftarrow newNode(p)$
      $Q.append(p)$
      $e \leftarrow newEdge(node.state, p, "take", p, conditions)$
      $E.append(e)$
      **if** $p$ is not $P.last$ **then**
        $e \leftarrow newEdge(p, p, "ignore", p, conditions)$
        $E.append(e)$
      **end if**
    **end if**
    $node \leftarrow node.children[p]$
  **end for**
  $F.append(p)$
**end for**
$A \leftarrow newNFA(Q, E, p_0, F)$ **return** $A$

行の入力によって引き起こされる．行の入力による遷移先が複数存在する場合，それぞれに遷移した状態を持つインスタンスが複製される．また，行の入力による遷移先が 1 つも無い場合は，メモリスペースの節約のためにそのインスタンスは削除される．各インスタンスは行パターンマッチングの中間結果を保持する match buffer に関連付けられている．MRPR-NFA への入力時に条件を満たした行へのポインタが対応する match buffer へ格納される．match buffer への行ポインタの格納は後述する *take action* を実行することによって行われる．

エッジに関連付けられた *action* は状態 $q_s$ を持つインスタンスに入力された行が *condition* の条件を満たし，状態遷移が発生した場合に実行される．MRPR-NFA では，*take* (*condition* の条件を満たした行へのポインタを match buffer に格納する．遷移先が受理状態であれば match buffer に格納された行ポインタを出力する) と *ignore* (*condition* の条件を満たした行をスキップする) の 2 つの *action* を用いる．

*condition* には MULTI_MATCH_RECOGNIZE 句の DEFINE 句で定義された条件を反映させる．また，遷移元の状態に初期状態を持つエッジを除く全てのエッジの *condition* に WINDOW 句で指定されたパターンオカレンスの最大長の制約条件を反映させる．また，*take action* 実行時に遷移先が受理状態であった場合，match buffer に格納されている行ポインタは 5.1.2 節で説明する pattern occurrence buffer へ出力される．

次に，MRPR-NFA の構築アルゴリズムについて説明する．MRPR-NFA は複数行パターンマッチングを効率的に実行するため，複数のパターンを統合的に扱うオートマトンを構築する．MRPR-NFA 構築アルゴリズムの疑似コードを Algorithm1 に示す．このアルゴリズムでは複数のパターンでマッチングを共有する部分を検出するために prefix tree の挿入アルゴリズムをベースとした手法を用いる．まず，アルゴリズムの入力としてクエリの PATTERN SET 句，DEFINE 句，WINDOW 句をコンパイルした情報を受け取る．次に，各パターンに対してパターン要素を元に状態とエッジを作成する．なお，Algorithm1 では説明を簡略化するために，パターン要素は単一のパターン変数であると仮定している．この時，接頭語が共通しているパターンがあれば，その部分は状態を共有するために状態とエッジの作成をスキップする．また，接尾語が共通しているパターンや途中のパターン要素が共通しているパターンについては共有せずに別々の状態としてオートマトンを構築する．

図 6 はリスト 2 のクエリから構築される MRPR-NFA の例を示している．各状態を結ぶ線はエッジを表し，エッジの近くに記述されている文字は，エッジに関連付けられている *action*, *name*, *condition* を表している．リスト 2 のクエリでは (X Y)，(X W Y)，(X Y Z) の 3 つのパターンが定義されている．(X Y) と (X Y Z) は接頭語の"X Y"が共通しているため，X と Y の状態を共有している．また，(X Y) と (X W Y) は接頭語の"X"が共通しているため，X の状態を共有している．

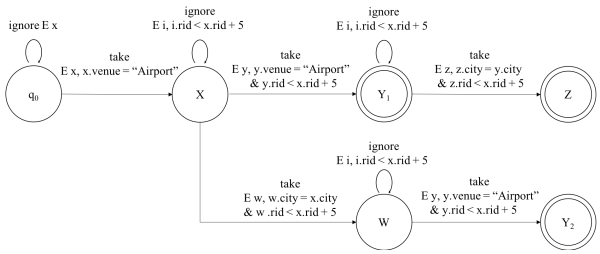


図 6: MRPR-NFA

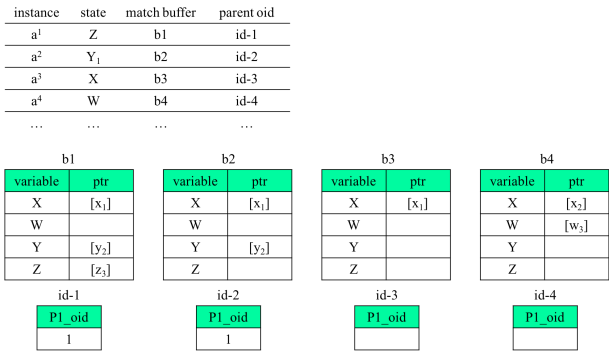


図 7: MRPR-NFA のインスタンス

図 7 に MRPR-NFA のインスタンスの例を示す．各インスタ

ンスはそれぞれ現在の状態，親パターンの pattern oid，match buffer に関連付けられている．親パターンの pattern oid については子パターンを持つパターン全てを対象とする．この値の取得方法については 5.2 節で説明する．また，図 7 は図 1 のシーケンス表の 3 行目まで $(r_1 \sim r_3)$ が入力された時のインスタンスの様子を表している．例えば，インスタンス $a^1$ は $q_0 \rightarrow X \rightarrow Y_1 \rightarrow Z$ のように遷移したインスタンスである．

#### 5.1.2 pattern occurrence buffer

pattern occurrence buffer は各インスタンスの match buffer から出力された行ポインタをパターンオカレンスとして管理するバッファである．図 8 に図 1 のシーケンス表の 5 行目まで $(r_1 \sim r_5)$ を入力した時の pattern occurrence buffer の様子を示す．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，そのパターンオカレンスを出力したインスタンスへのポインタ，パターンオカレンスの累計の count が格納される．新たなパターンオカレンスが格納される時に count が加算され，さらにその値を pattern oid としてパターンオカレンスに付与する．また，子パターンのパターンオカレンスは検出した親パターンオカレンスの pattern oid も保持する．さらに，子パターンを持つパターンの pattern oid はパターンオカレンスに付与される時，効率的に親子関係検出を行うために対応するインスタンスへ同じ値が付与される．例えば，図 8 の P1 に格納されている $(x_1, y_2)$ は $P1\_oid = 1$ を取得している．同様に，$P1\_oid = 1$ は $a^1$ にも付与されるため，図 7 の $a^1$ が $P1\_oid = 1$ の情報を保持している．また，pattern occurrence buffer は行パターンマッチング時に，新たな行が入力される前に保持されているパターンオカレンスをパターンオカレンス表へ出力し，count 以外の中身を空にする．図 8 では 5 行目が入力された時に，すでにパターンオカレンス表へ出力されているものは背景を灰色にしている．

P1

| count | instance | X | Y | P1_oid |
|---|---|---|---|---|
| 3 | $a^1$ | $[x_1]$ | $[y_2]$ | 1 |
| | $a^3$ | $[x_1]$ | $[y_5]$ | 2 |
| | $a^6$ | $[x_2]$ | $[y_5]$ | 3 |
| | | | | |

P2

| count | instance | X | W | Y | P1_oid | P2_oid |
|---|---|---|---|---|---|---|
| 2 | $a^7$ | $[x_2]$ | $[w_3]$ | $[y_5]$ | 2 | 1 |
| | $a^8$ | $[x_2]$ | $[w_4]$ | $[y_5]$ | 2 | 2 |

P3

| count | instance | X | Y | Z | P1_oid | P3_oid |
|---|---|---|---|---|---|---|
| 2 | $a^1$ | $[x_1]$ | $[y_2]$ | $[z_3]$ | 1 | 1 |
| | $a^2$ | $[x_1]$ | $[y_2]$ | $[z_4]$ | 1 | 2 |

図 8: pattern occurrence buffer

### 5.2 複数行パターンマッチングと親子関係検出

#### 5.2.1 複数行パターンマッチング

まずは，複数行パターンマッチングのアルゴリズムについて説明する．具体的なアルゴリズムを Algorithm 2 に示す．複数行パターンマッチングでは称号対象の表のスキャンを行う．その際に，参照した行を 1 行ずつ MRPR-NFA に入力し，各インスタンスの遷移を発生させる．あるインスタンスで *take action* を持つエッジによる遷移が発生した場合，入力行へのポインタをそのインスタンスの match buffer へ格納する．また，その際に，遷移先の状態が受理状態であれば対応する pattern occurrence buffer へパターンオカレンスの出力を行う．各インスタンスの遷移が終わり，次の行を参照する前に pattern occurrence buffer に保持されているパターンオカレンス同士で親子関係検出を実行し，パターンオカレンス表へ結果を出力する．また，パターンオカレンス表への結果出力後に pattern occurrence buffer に保持されているパターンオカレンスを全て削除する．その後，次の行の入力を行う．

**Algorithm 2** 複数行パターンマッチング

**Require:** rows R, MRPR-NFA A, pattern occurrence buffer B
  $patternOccurrenceTable \leftarrow \emptyset$, $Automata \leftarrow \emptyset$
  $Automata.append(newInstance())$
  **for all** $r \in R$ **do**
    $newAutomata \leftarrow \emptyset$
    **for all** $a \in Automata$ **do**
      $edges \leftarrow A.getTransitableEdges(a, r)$
      **for all** $e \in edges$ **do**
        $ac \leftarrow a.copy()$
        $ac.state \leftarrow e.q_d$
        **if** e.action is "take" **then**
          $ac.matchBuffer[e.name].append(r)$
          **if** $e.q_d$ in $A.F$ **then**
            $ac.matchBuffer.pushPatternOccurrence(B)$
          **end if**
        **end if**
        $newAutomata.append(ac)$
      **end for**
    **end for**
    $B.detectParentChildRelationships()$
    $B.pushPatternOccurrences(patternOccurrenceTable)$
    $Automata \leftarrow newAutomata$
  **end for**
  **return** $patternOccurrenceTable$

#### 5.2.2 親子関係検出

次に，パターンオカレンスの親子関係検出アルゴリズムについて説明する．パターンオカレンスの親子関係検出はパターンの親子関係が prefix match case の場合と suffix match case の場合で異なる手法を用いる．

まずは，パターンの親子関係が prefix match case の場合について説明する．パターンの親子関係が prefix match case の場合，子パターンオカレンスは必ず，親パターンオカレンスより後に抽出されるという性質を持つ．また，MRPR-NFA において prefix match case の子パターンは親パターンの受理状態を共有している．従って，prefix match case の親パターンオカレンスを出力したインスタンスはその後に子パターンオカレンスを出力する可能性がある．その場合，親パターンオカレンスの pattern oid はインスタンスが保持しているため，子パターンオカレンス出力時に共に親の pattern oid を出力することで親子関係検出を行うことができる．

次に，パターンの親子関係が suffix match case の場合について説明する．パターンの親子関係が suffix match case の場合，最後のパターン要素が共通しているため子パターンオカレ

ンスは必ず，親パターンオカレンスと同じタイミングで抽出されるという性質を持つ．この性質から，抽出されたタイミングで親子関係を持つ suffix match case のパターンオカレンスは全て pattern occurrence buffer に格納されているということが言える．そこで，suffix match case の親子関係検出は対応するパターンオカレンス同士の結合処理によって行う．結合の条件としてパターンオカレンスが保持している各行の rid を用いる．親子で共通するパターン変数にマッチした行の rid を 2 つのパターンオカレンスで比較し，全ての行の rid が一致したパターンオカレンス同士は親子関係を持つと判定する．ここで，子パターンオカレンスの親パターンオカレンスは孫パターンオカレンスの親パターンオカレンスも兼ねるという性質を考慮した場合，初めに親子の結合処理を実行しておけば，孫子の結合処理実行時に親と孫の親子関係も検出できることがわかる．この性質を利用して冗長な結合処理を削減するために PATTERN SET 句から親子関係検出計画を作成する．親子関係検出計画は結合処理を行う親子パターンのペアのリストである．親子関係検出では各検出計画に対して対応するバッファの結合処理を行う．また，親子関係検出ではパターンオカレンスの各行の rid を用いた等価結合を実行するため，各行の rid をキーとした Hash Join を用いて効率的に結合処理を実行する．

## 6 実験

本章では，提案手法の性能を評価するために行った実験について記す．本実験では，提案手法の性能を評価するために RDB 上に擬似的に提案手法と比較手法を実装し，クエリの処理時間を計測する．比較手法は，提案手法と同様に MRPR-NFA で複数行パターンマッチングを行い，パターンマッチング終了後にまとめて親子関係検出を行う Collective Detection とパターン毎にパターンマッチングを実行し，SQL の結合処理で親子関係検出を行う MATCH_RECOGNIZE の 2 つである．実験は全てで Ubuntu 16.04.3 LTS，Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz，32GB RAM で構成された PC 上で行った．実験に用いる RDB はオープンソースの PostgreSQL 9.5 を選択した．また，PostgreSQL の UDF(ユーザー定義関数) として提案手法と比較手法を C 言語で実装した．さらに，本実験ではシーケンスデータやクエリの特徴をコントロールするために人工データをデータセットとして用いる．人工データはシーケンス数が 100，各シーケンスが 10,000 行を持つ，合計 1,000,000 行の表である．提案手法を擬似的に実装した UDF を図 9 に示す．PostgreSQL では表を出力として扱うことができる表関数を使用できる．表関数は FROM 句に記述することで，関数で作成した表を SQL のセマンティクス上で扱うことができる．MULTI_MATCH_RECOGNIZE 句はパターンオカレンス表を出力するため，表関数として UDF を実装した．この UDF は 5 つの文字列を引数とする．1 つ目の引数は MULTI_MAT-CH_RECOGNIZE 句を用いたクエリの FROM 句，PARTITION BY 句，ORDER BY 句の指定を変換した SQL 文である．これらの 3 つの句の指定から，図 4 のような rid が付与されたソート済みの表を作成する SQL を関数内部で実行し，メモリに展開する．さらに，2 つ目から 5 つ目の引数はそれぞれ対応する句で指定されている文字列を受け取る．

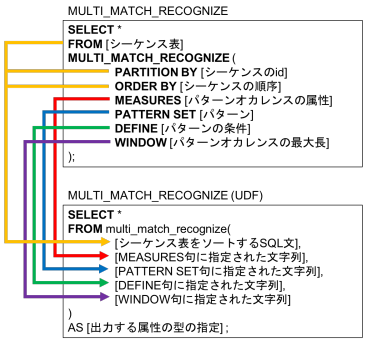


図 9: MULTI_MATCH_RECOGNIZE 句の UDF

図 10 に prefix match case でパターン数を変化させた実験結果を示す．グラフの x 軸はパターン数，左の y 軸はクエリの処理時間，右の y 軸は出力されたパターンオカレンスの数を示している．さらに，CD は Collective Detection，MR は MATCH_RECOGNIZE 句を示している．パターン数 5 における MR の処理時間は他と比べて非常に大きくなってしまったため，グラフからは除外している．実験結果から，どのパラメータでも提案手法が最も高速であることがわかる．特に，MATCH_RECOGNIZE 句と比較すると 2 倍近く高速に実行できている．これは，MATCH_RECOGNIZE 句がパターンの数だけ UDF を呼び出すことで，複数行パターンマッチングの処理時間が増加しているためである．また，Collective Detection と比較すると提案手法は 10%程度高速に実行できていることがわかる．これは，親子関係が prefix match case の場合，親の pattern_oid を MRPR-NFA のインスタンスで伝搬することで，結合処理を実行せずに親子関係を検出できるためである．

さらに，図 11 に suffix match case での実験結果を示す．実験結果から，どのパラメータでも提案手法が最も高速であることがわかる．prefix match case と比べて全体的に処理時間が大きくなっているのは，suffix match case の方が MRPR_NFA の state，インスタンスの数が多く，複数行パターンマッチングの処理時間が大きくなるためである．Collective Detection と比較した場合，こちらも 10%程度高速に実行できていることがわかる．これは，親子関係が suffix match case の場合，複数行パターンマッチング中に逐次的に結合処理を実行することで，ある程度区切られた範囲内で結合処理を実行できるためである．

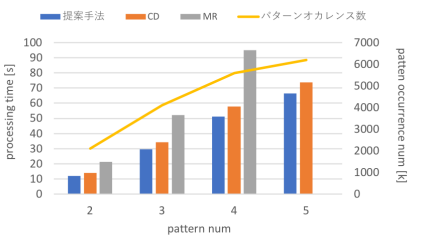


図 10: prefix match case における処理時間

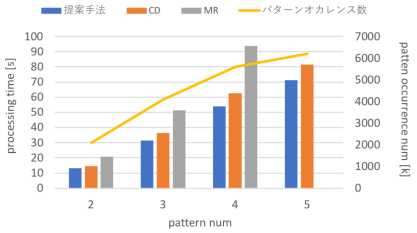


図 11: suffix match case における処理時間

## 7 関連研究

過去にはシーケンスデータに対してパターンマッチングを実行する手法が数多く研究されている．その多くは SASE [13] [4]，Cayuga [3]，Zstream [9] などのストリームデータに対してリアルタイムにパターンマッチングを行う CEP(Complex Event Processing) として提案されている．さらに，クエリワークロードとして登録された複数パターンのパターンマッチングを部分的に共有して効率的に問合せ処理を行う手法も提案されている [12] [14] [10]．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] や E-Cube [6] [7] が知られている．さらに，S-OLAP，E-Cubu は複数のパターンを効率的に処理する機能を持っている．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．本研究では汎用的なデータ管理システムである RDB を用いることでこの問題を解決している．

## 8 おわりに

本研究では，RDB 上でのシーケンス OLAP をサポートする複数行パターンマッチングの問合せ処理手法を提案した．まずは，パターンの親子関係を元にパターン階層モデルを定義した．また，複数行パターンマッチングを実行するための SQL 拡張記述 MULTI_MATCH_RECOGNIZE 句を示した．MULTI_MATCH_RECOGNIZE 句の問合せ処理として，提案手法では複数パターンを統合的に扱う MRPR-NFA を用いて効率的に複数行パターンマッチングとパターンオカレンス間の親子関係検出を実行し，パターンオカレンス表を出力することができる．出力されたパターンオカレンス表でシーケンス OLAP の OLAP-Pattern 操作をサポートすることが可能となる．また，今後の課題として prefix match pattern，suffix match pattern を組合せたパターン，クリーネ閉包を用いたパターンの実験，実データを用いたシーケンス OLAP 分析の実証実験や，シーケンス OLAP 用 GUI の開発があげられる．

## 謝　辞

本研究の一部は，平成 30 年度共同研究 (SKY 株式会社) (CPE30034))「データエンジニアリングの知見の応用による SKYSEA Client View のログ及び資産情報の処理の高速化・軽量化・高度化」の助成を受けたものである．

## 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] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 147–160. ACM, 2008.

[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] Olga Poppe, Allison Rozet, Chuan Lei, Elke A Rundensteiner, and David Maier. Sharon: Shared online event sequence aggregation. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 737–748. IEEE, 2018.

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

[12] Medhabi Ray, Chuan Lei, and Elke A Rundensteiner. Scalable pattern sharing on event streams. In *Proceedings of the 2016 International Conference on Management of Data*, pages 495–510. ACM, 2016.

[13] 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.

[14] Shuhao Zhang, Hoang Tam Vo, Daniel Dahlmeier, and Bingsheng He. Multi-query optimization for complex event processing in sap esp. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1213–1224. IEEE, 2017.