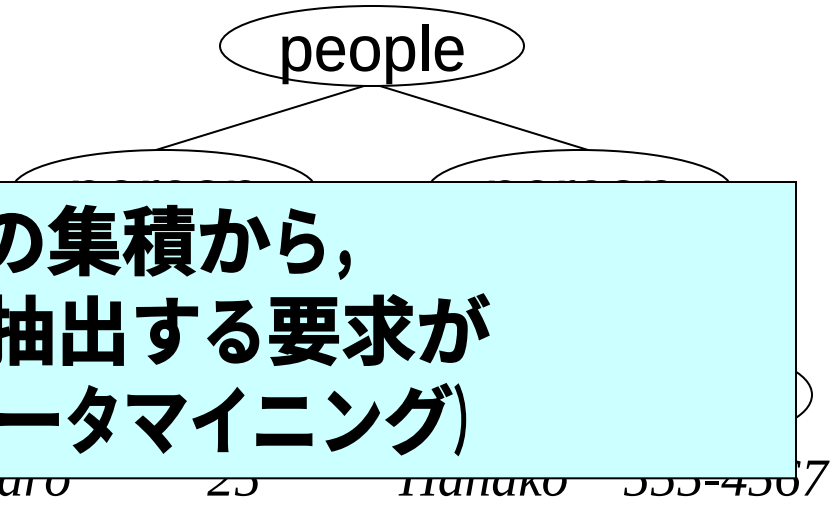
半構造データからの頻出パターン発見アルゴリズム

浅井達哉, 安部賢治, 川副真治,
坂本比呂志, 有村博紀, 有川節夫
九州大学 大学院システム情報科学府・研究院

背景：半構造データ

- **大量のウェブページ，XML文書が蓄積**
- **半構造データ**[Abiteboul, Buneman, Suciu 00]
- XML [2nd ed, W3C 00]

```
<people>
  <person>
    <name>Taro</name>
    <age>25</age>
  </person>
  <person>
    <name>Hanako</name>
    <tel>555-4567</tel>
  </person>
</people>
```

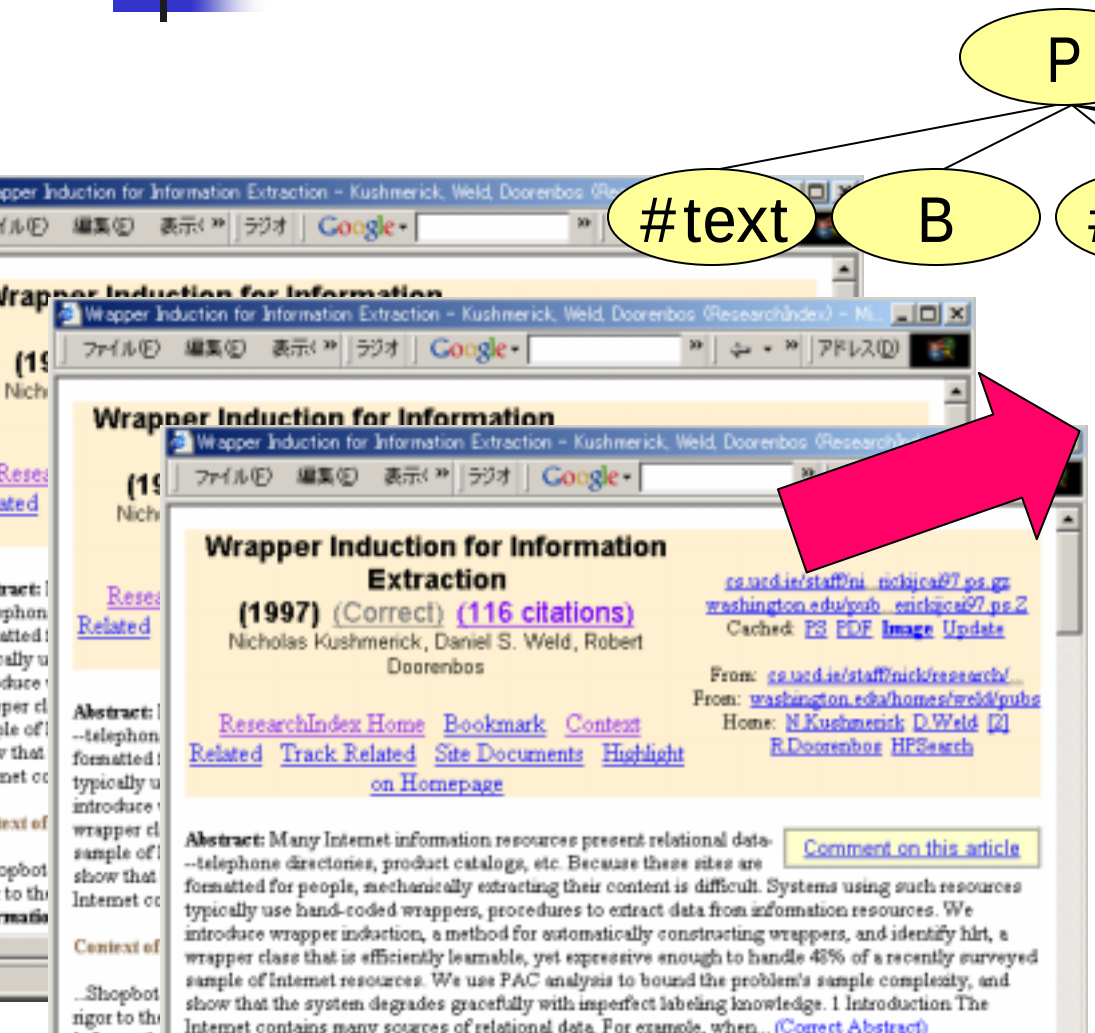


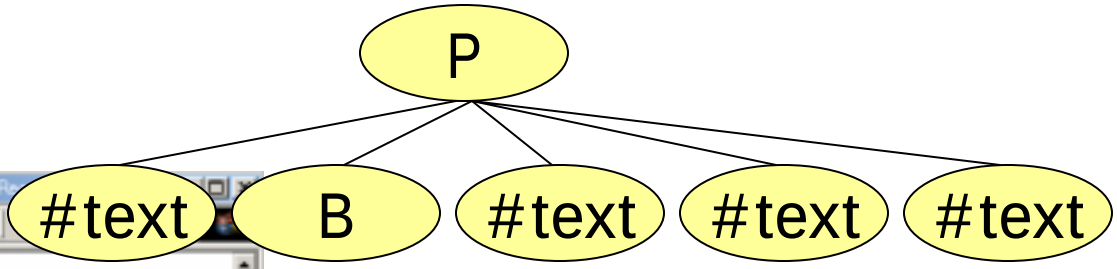
- **大規模な半構造データの集積から，**
  有益な情報を自動的に抽出する要求が
  高まっている (半構造データマイニング)

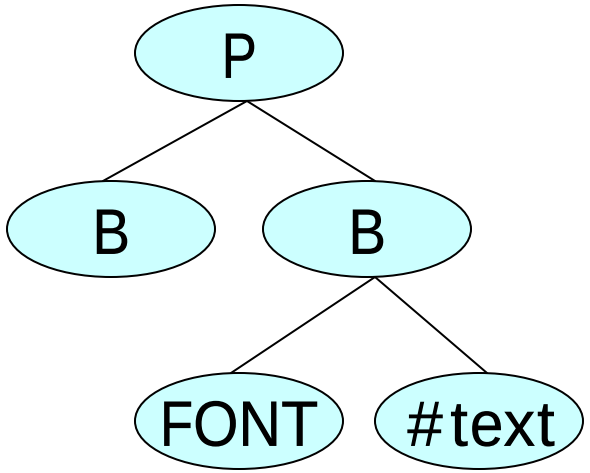
背景：半構造データマイニング

- **半構造データマイニング**
  - [Wang, Liu 97]
    - **木を経路の集合として扱う**
    - **各経路をアイテムとみなした相関規則マイニング手法**
  - [Miyahara, et al. 01]
    - **タグ木パターンと呼ばれる構造パターンを用いて，頻出パターン発見問題を考察**

研究の目標


Wrapper Induction for Information Extraction - Kushmerick, Weld, Doorenbos (ResearchIndex)
Wrapper Induction for Information Extraction
(1997) (Correct) (116 citations)
Nicholas Kushmerick, Daniel S. Weld, Robert Doorenbos
ResearchIndex Home Bookmark Context Related Track Related Site Documents Highlight on Homepage
Comment on this article
Abstract: Many Internet information resources present relational data--telephone directories, product catalogs, etc. Because these sites are formatted for people, mechanically extracting their content is difficult. Systems using such resources typically use hand-coded wrappers, procedures to extract data from information resources. We introduce wrapper induction, a method for automatically constructing wrappers, and identify hlrt, a wrapper class that is efficiently learnable, yet expressive enough to handle 48% of a recently surveyed sample of Internet resources. We use PAC analysis to bound the problem's sample complexity, and show that the system degrades gracefully with imperfect labeling knowledge. 1 Introduction The

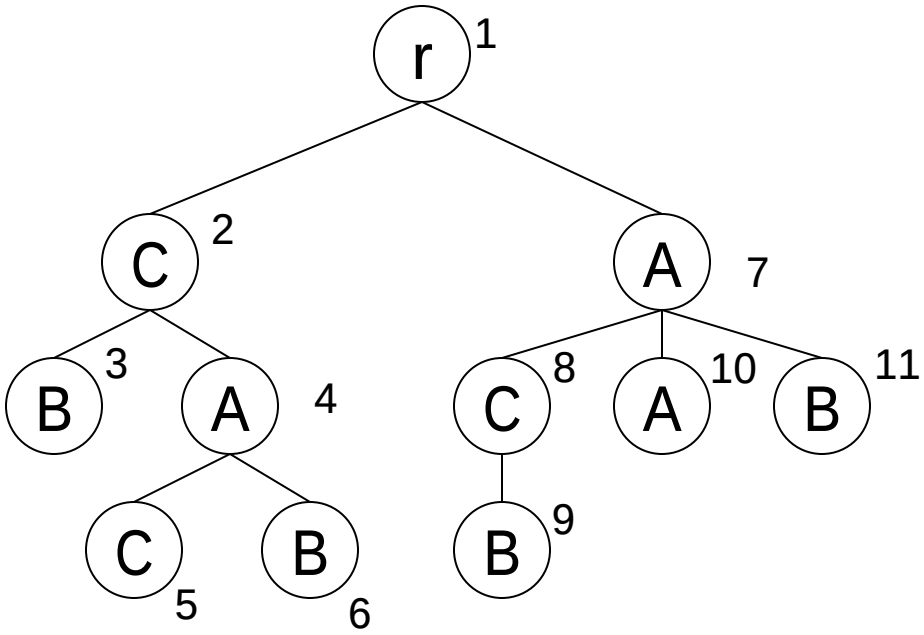

P
#text
B
#text
#text
#text


P
B
B
FONT
#text

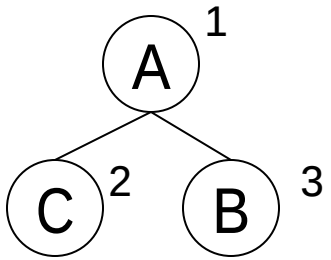
半構造データのモデル

ラベル付き順序木

データ木D

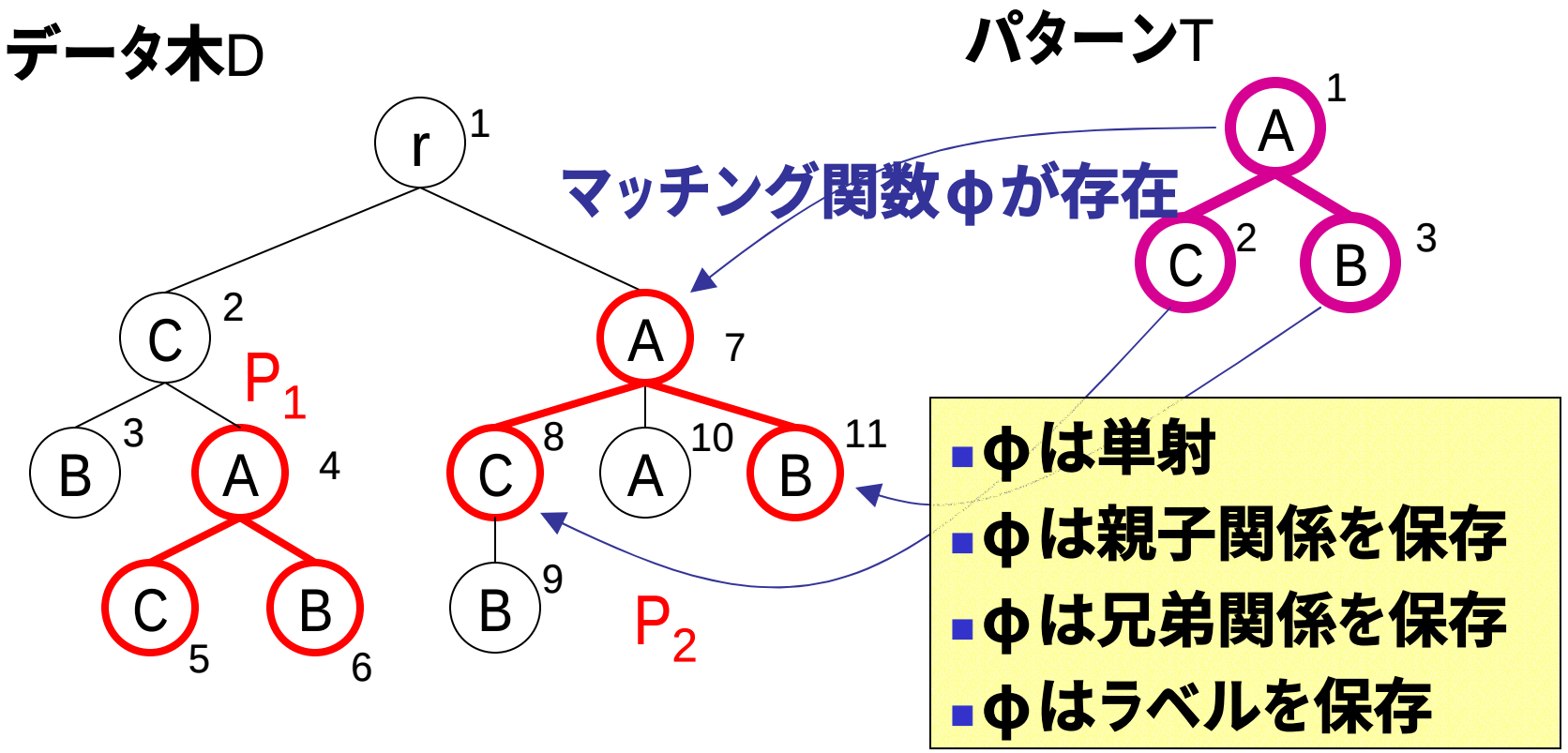


パターンT



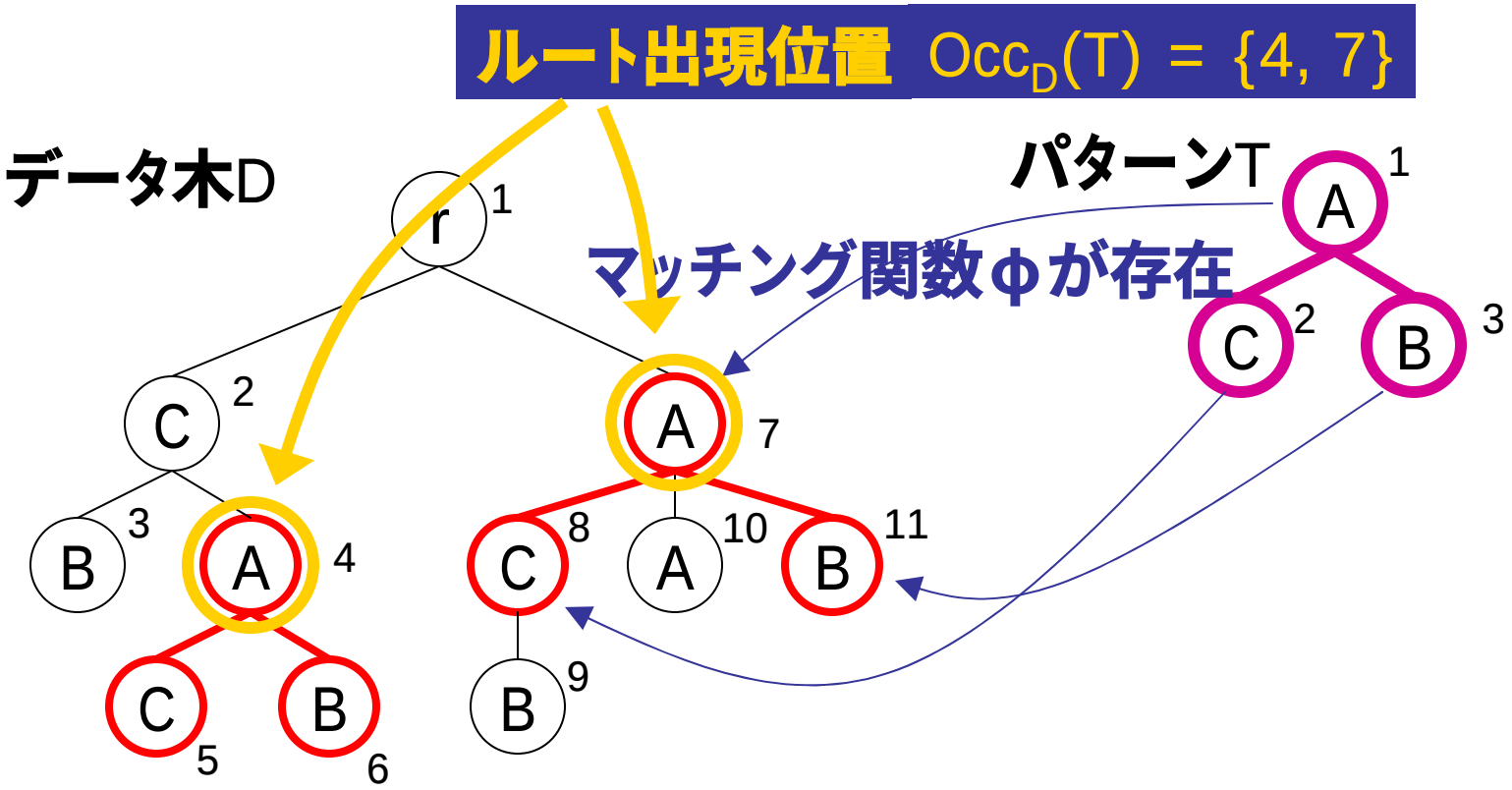
パターンの出現

パターンTがデータ木Dに出現する



パターンの出現数

パターンTの出現数は，ルートが写像する節点で数える



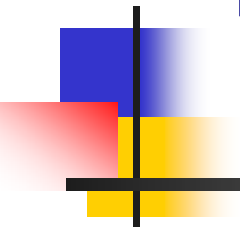
半構造データマイニング問題

頻出パターン発見問題

データ木Dと最小サポート $0 < \sigma \leqq 1$ に対し，ルート出現に関してσ頻出

$$\#Occ_D(T)/\#D \geqq \sigma$$

であるすべての順序木パターンTをみつけよ．



アルゴリズム

アルゴリズムの方針

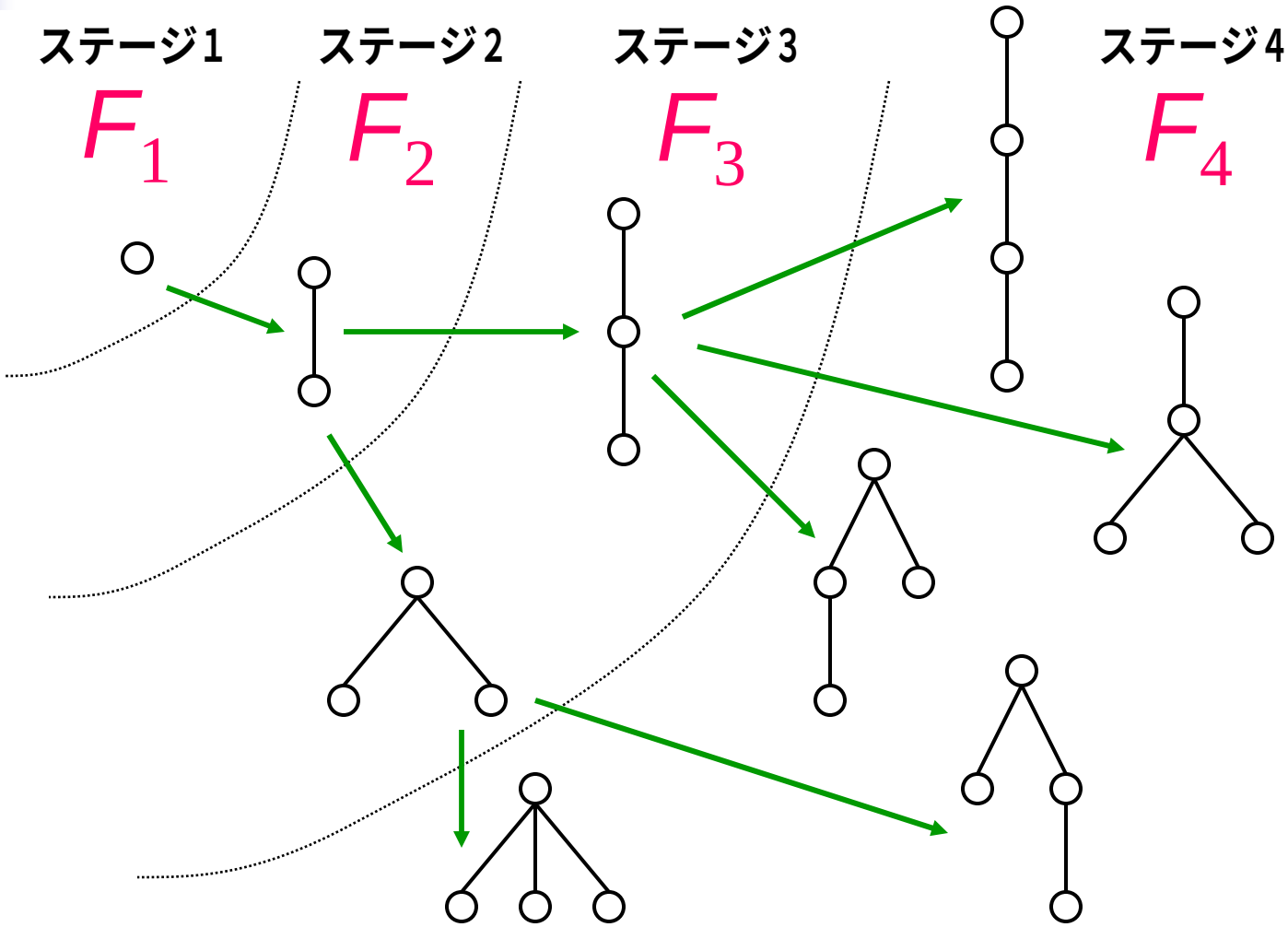
- すべての順序木パターンを，小さなサイズから順に，できるだけ無駄なく生成する．
- パターンの出現位置を漸増的に計算し，かつ，できるだけコンパクトに保持する．

アルゴリズムFREQT

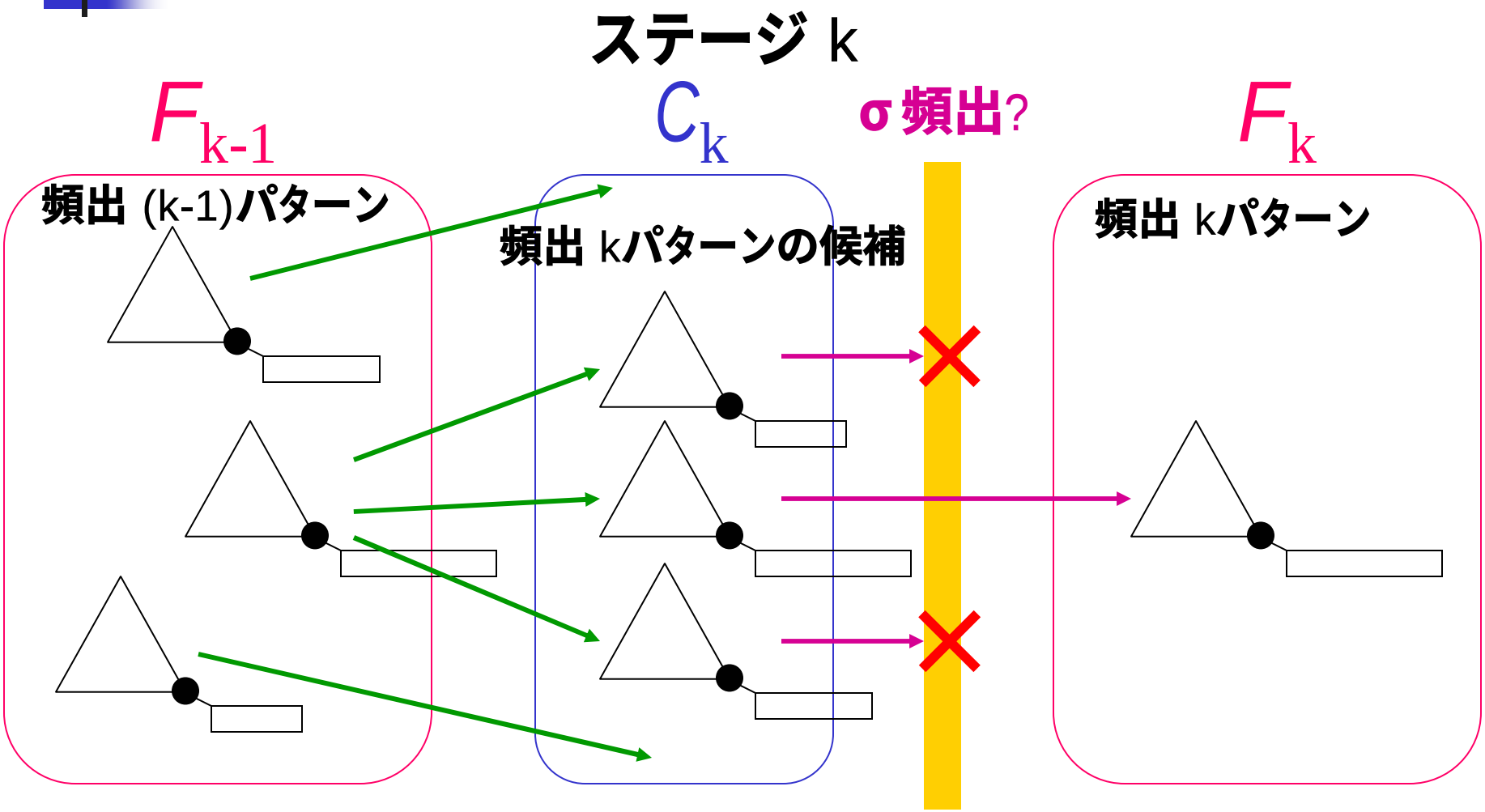
F_k：サイズkのすべて頻出パターンの集合
C_k：F_kの候補集合

- F_1を計算
- F_{k-1} からF_k を計算
  - 効率のよい順序木の枚挙
    (F_{k-1} から C_k)
  - 同時に，その出現位置リストを更新
  - 頻出パターンを選択
    (C_k から F_k)

アルゴリズムFREQT



F_{k-1} からF_k を計算



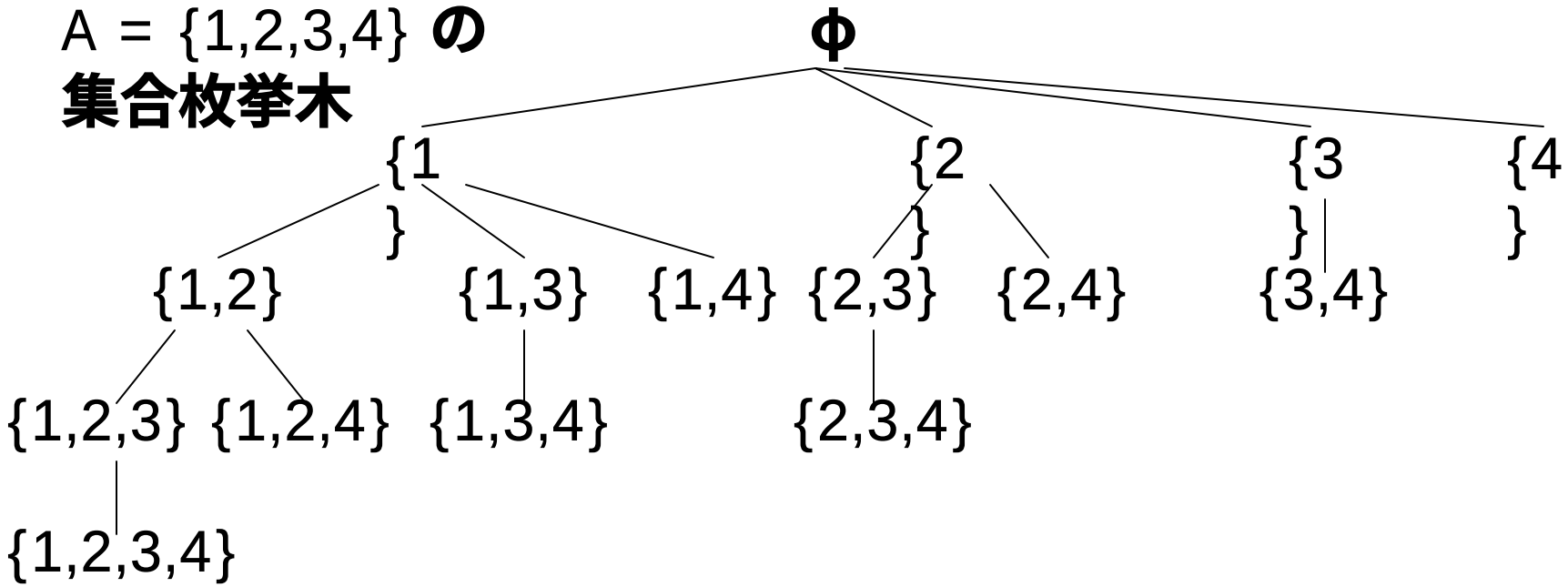
アルゴリズムFREQT

F_k：サイズkのすべて頻出パターンの集合
C_k：F_kの候補集合

- F_1を計算
- F_{k-1} からF_k を計算
  - 効率のよい順序木の枚挙
    (F_{k-1} から C_k)
  - 同時に，その出現位置リストを更新
  - 頻出パターンを選択
    (C_k から F_k)

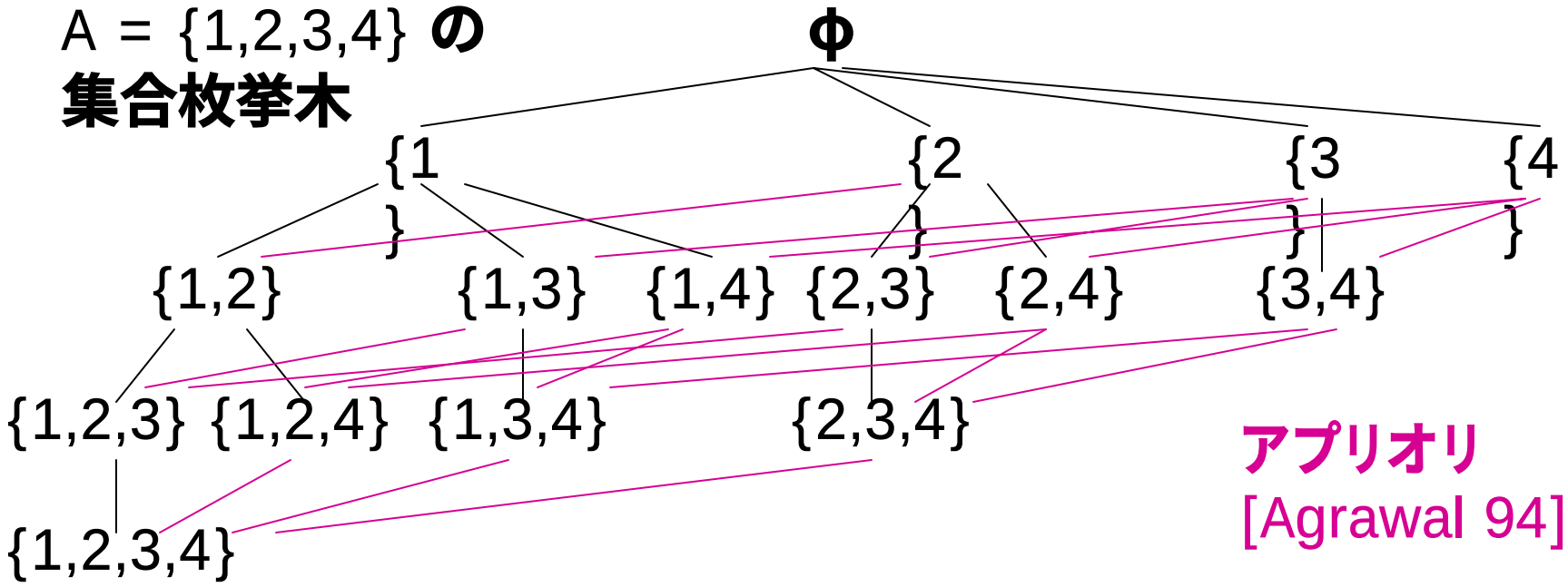
集合枚挙木 [Bayardo (SIGMOD'98)]

- **有限集合Aのすべての部分集合を重複なく枚挙する手法**



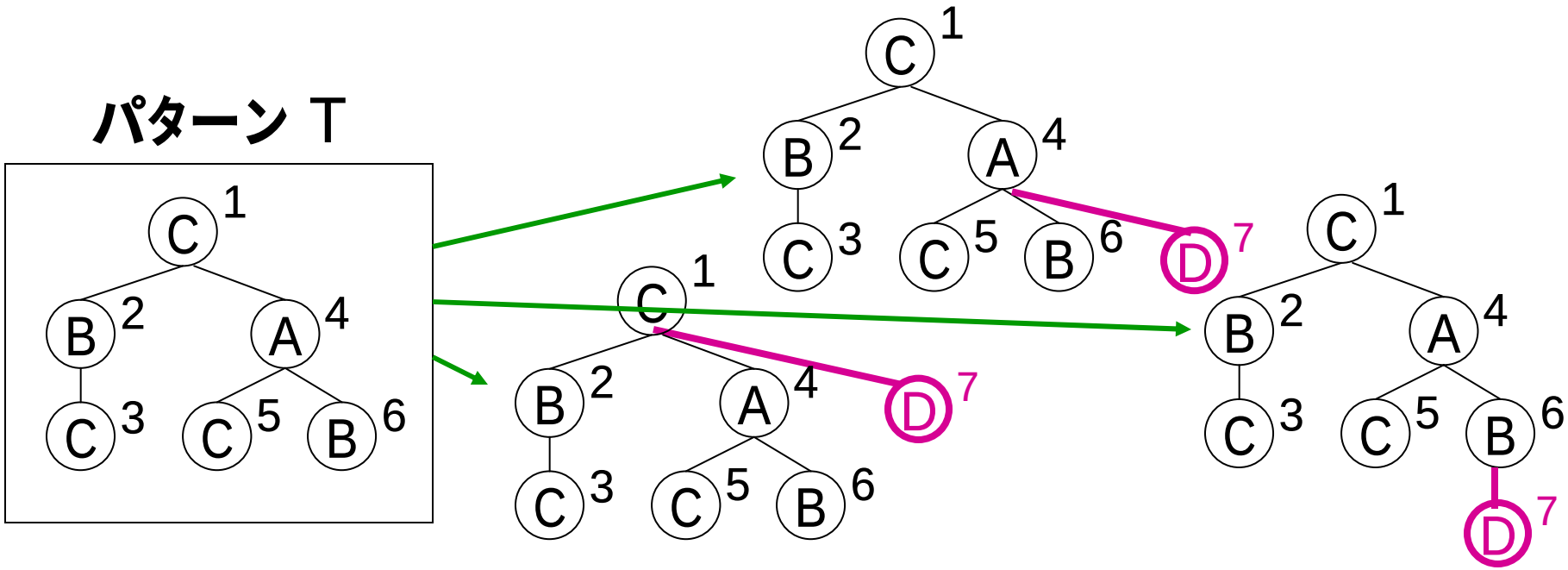
集合枚挙木 [Bayardo (SIGMOD'98)]

- **有限集合Aのすべての部分集合を重複なく枚挙する手法**

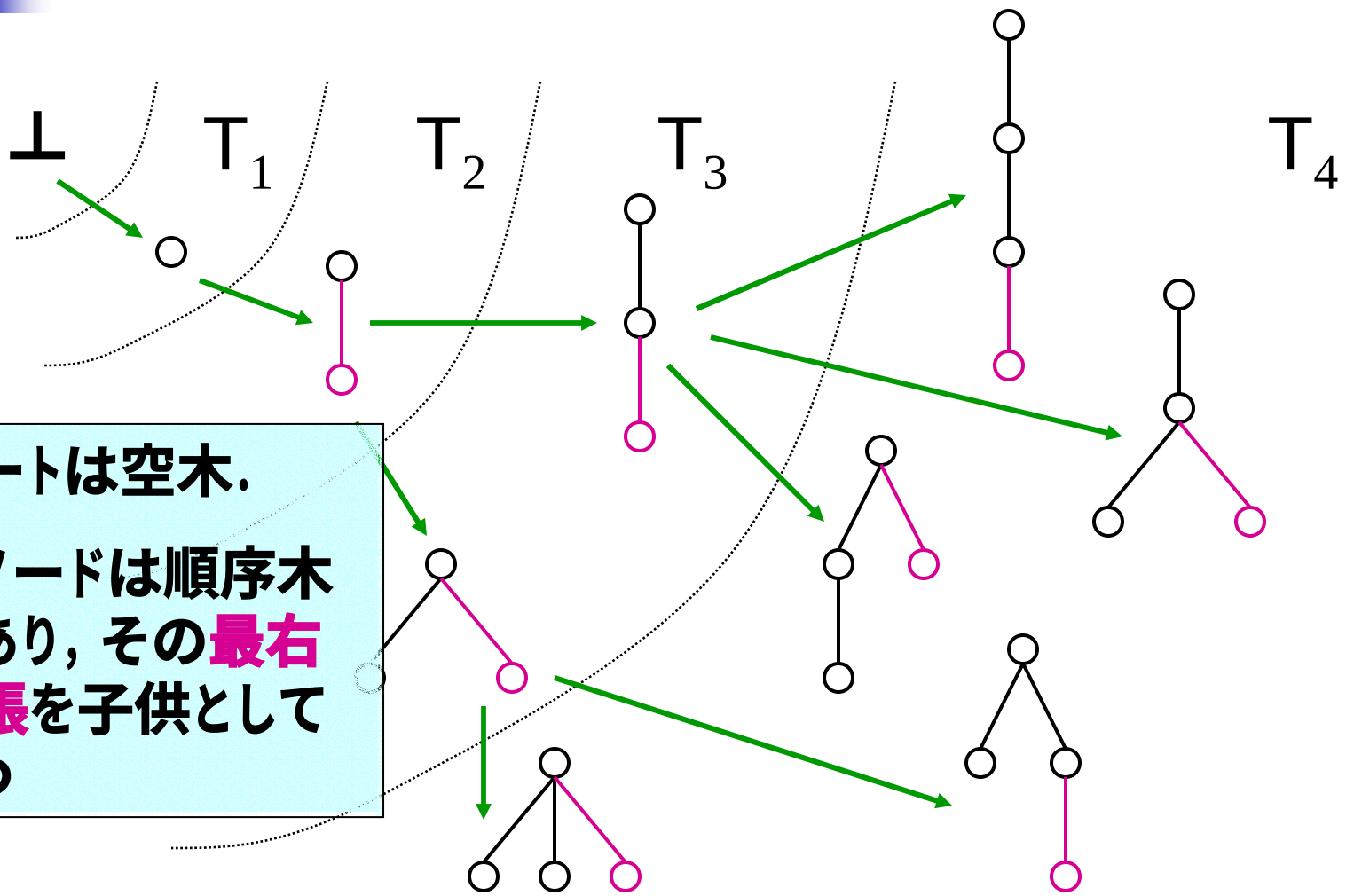


最右拡張

- パターンTの**最右拡張**T'とは，Tに**新たな節点**を追加して得られる順序木
  - **一番右の枝**に追加
  - **末弟**になるように追加



順序木枚挙グラフ



- ルートは空木.
- 各ノードは順序木であり，その**最右拡張**を子供としてもつ

補題１

T：すべての順序木パターンの集合

補題１

順序木枚挙グラフGの節点集合は$T \cup \{\perp\}$であり，さらに，Gは木である．

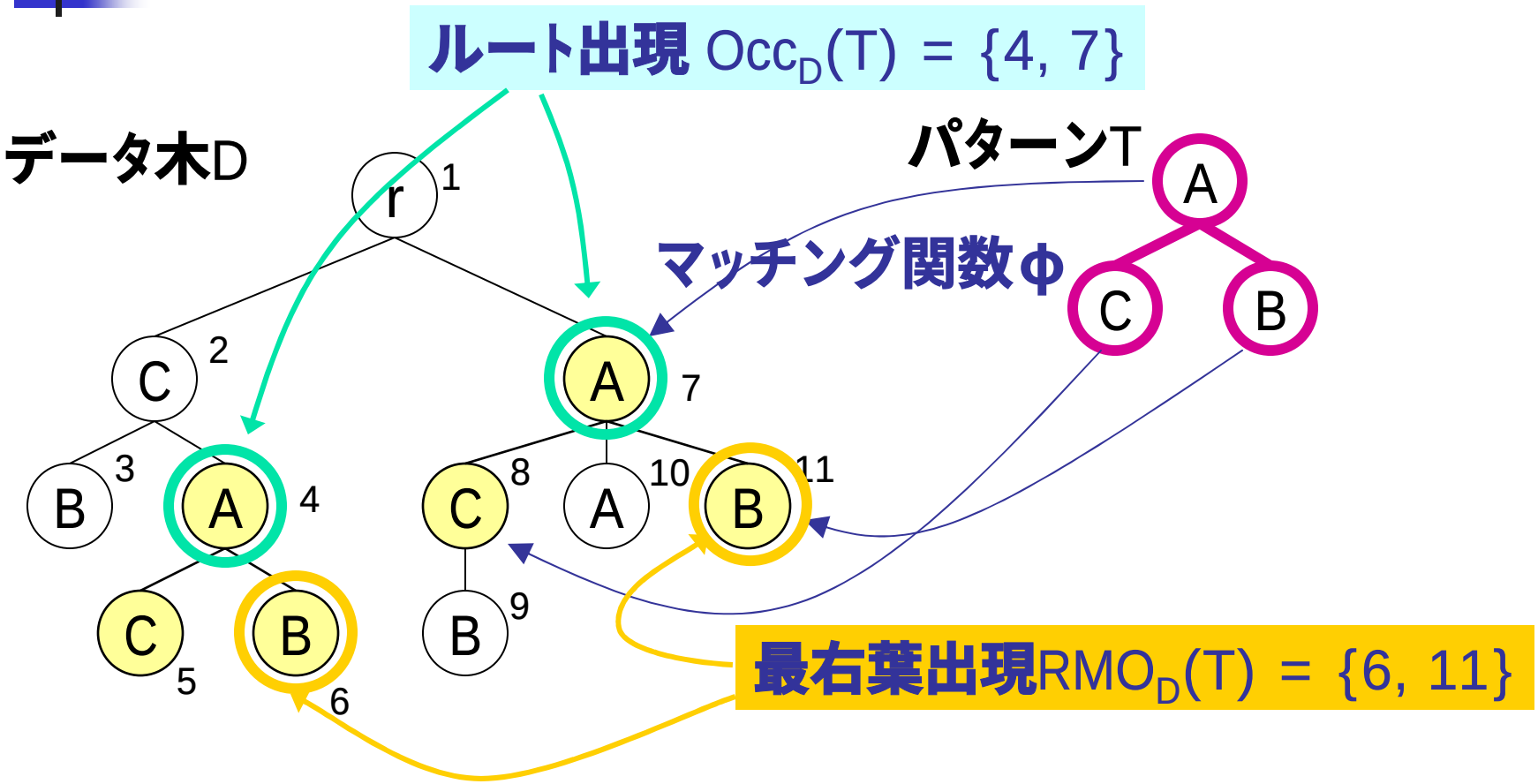
すなわち，
Gはすべての順序木パターンを重複せずに枚挙する．

アルゴリズムFREQT

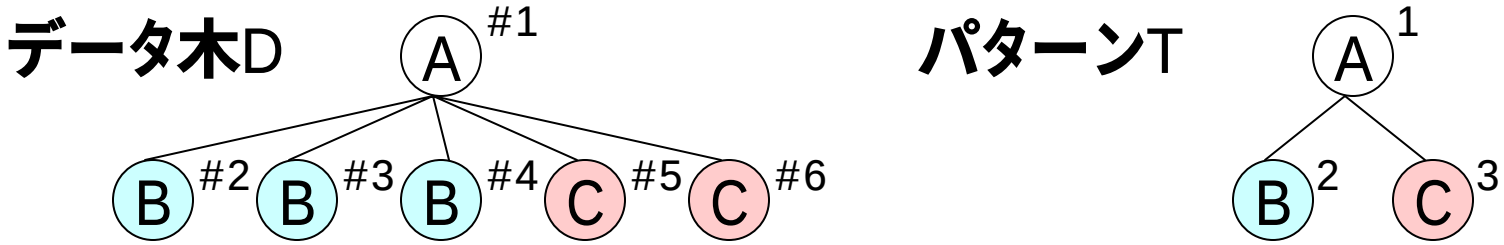
F_k：サイズkのすべて頻出パターンの集合
C_k：F_kの候補集合

- F_1を計算
- F_{k-1} からF_k を計算
  - 効率のよい順序木の枚挙
    (F_{k-1} から C_k)
  - 同時に，その出現位置リストを更新
  - 頻出パターンを選択
    (C_k から F_k)

パターンの出現数の計算



出現位置のコンパクトな保持



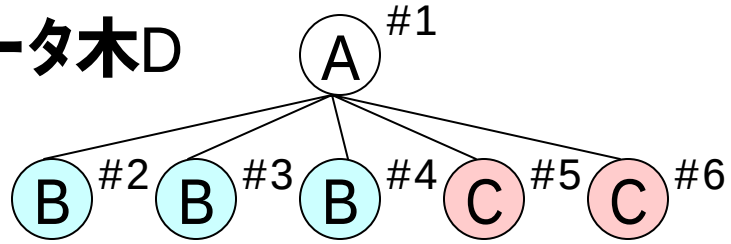
素朴な定義：
(1,2,3)**の出現位置をすべて保持**

(#1,#2,#5), (#1,#2,#6),
(#1,#3,#5), (#1,#3,#6),
(#1,#4,#5), (#1,#4,#6).

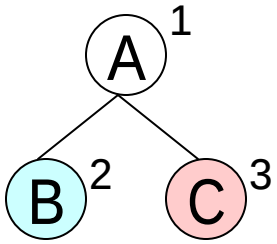
最悪の場合， $O(m^k)$**個の出現が存在**
(m=|D|，k=|T|)

出現位置のコンパクトな保持

データ木D



パターンT



素朴な定義：
(1,2,3)の出現位置をすべて保持

(#1,#2,#5), (#1,#2,#6),
(#1,#3,#5), (#1,#3,#6),
(#1,#4,#5), (#1,#4,#6),

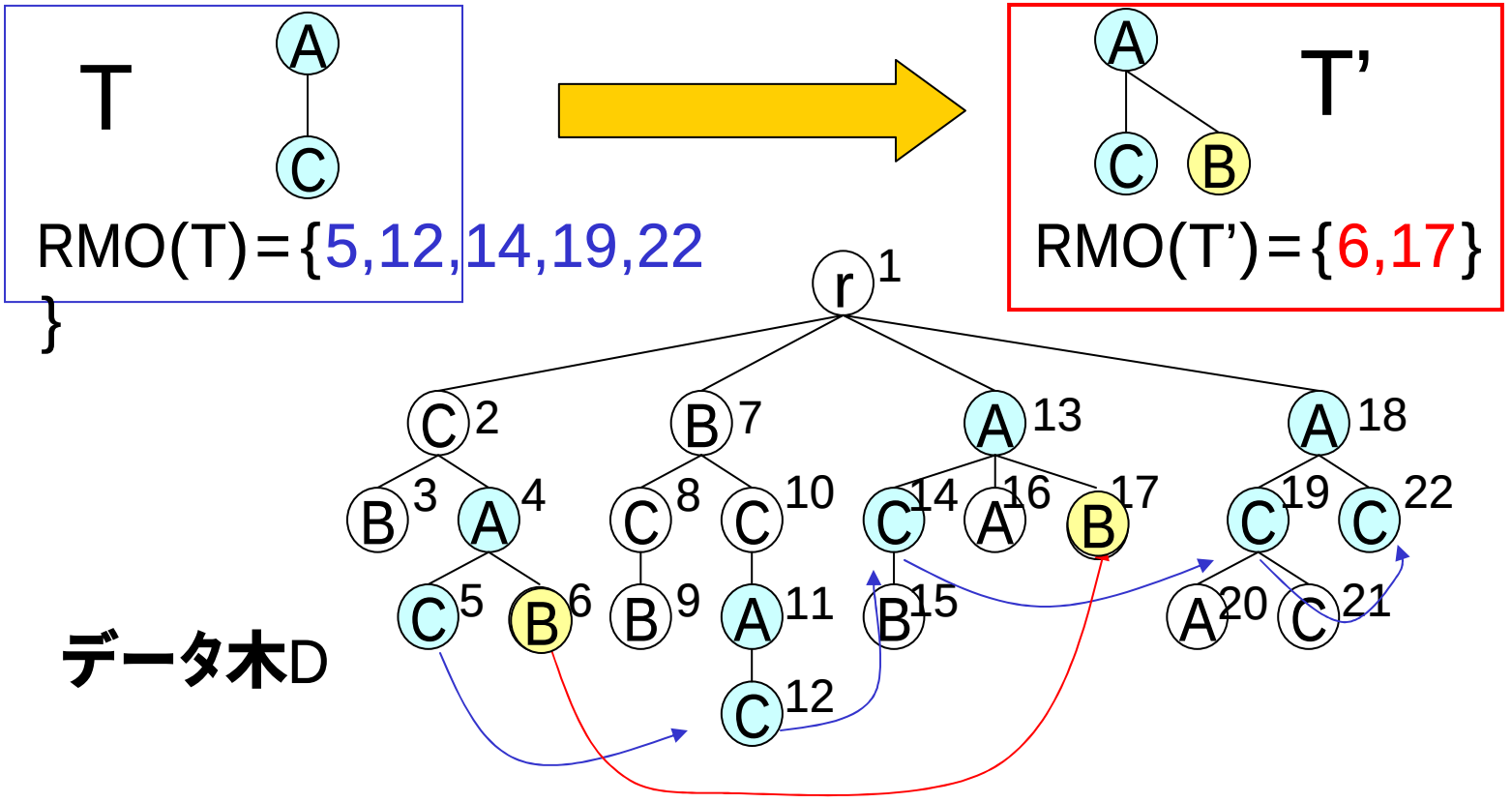
最悪の場合，$O(m^k)$個の出現が存在
(m=|D|，k=|T|)

最右葉出現：
最右葉(=3)の出現位置のみ保持

#5,#6

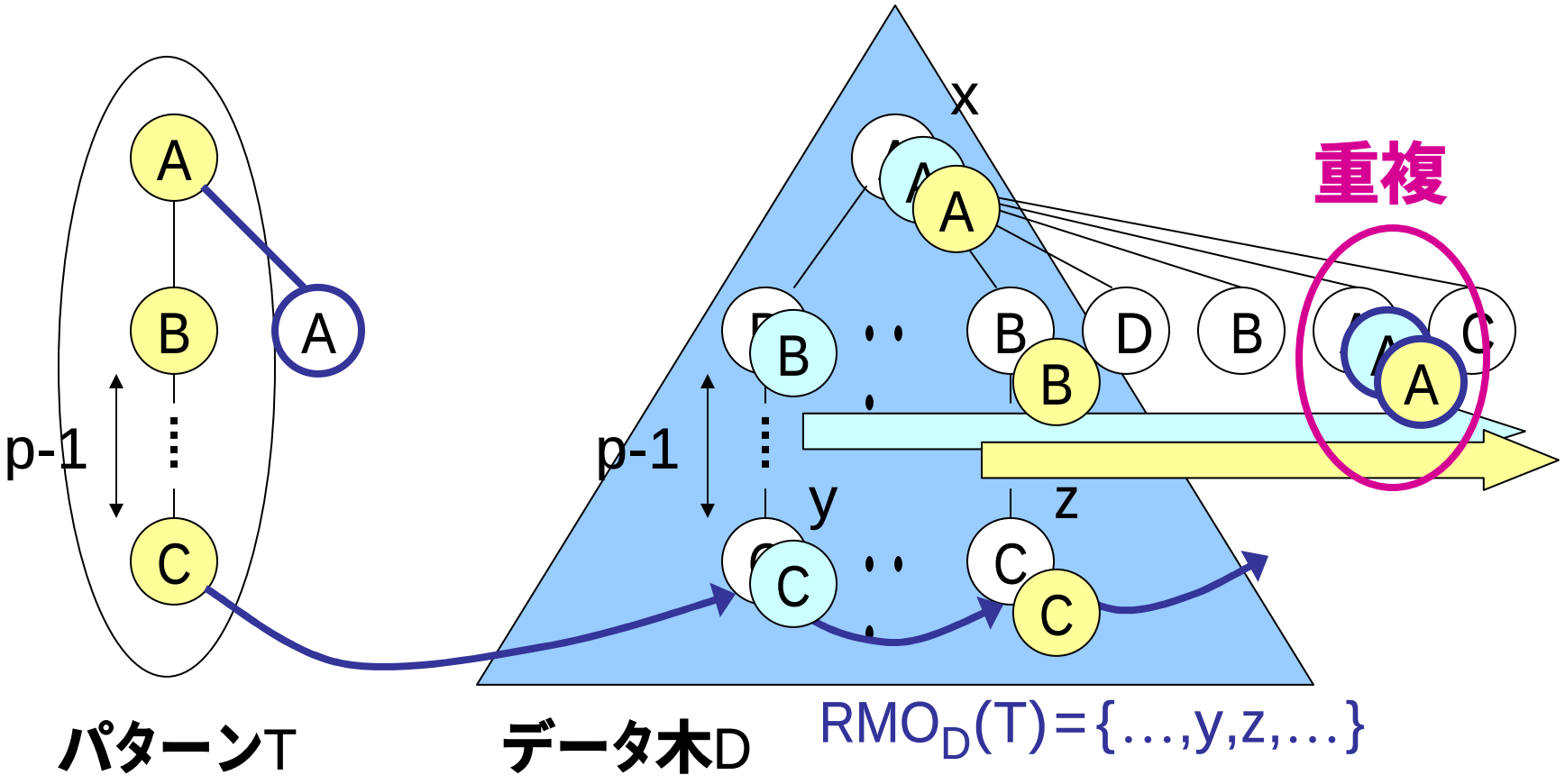
出現は，高々m個

出現リストの漸増的な計算



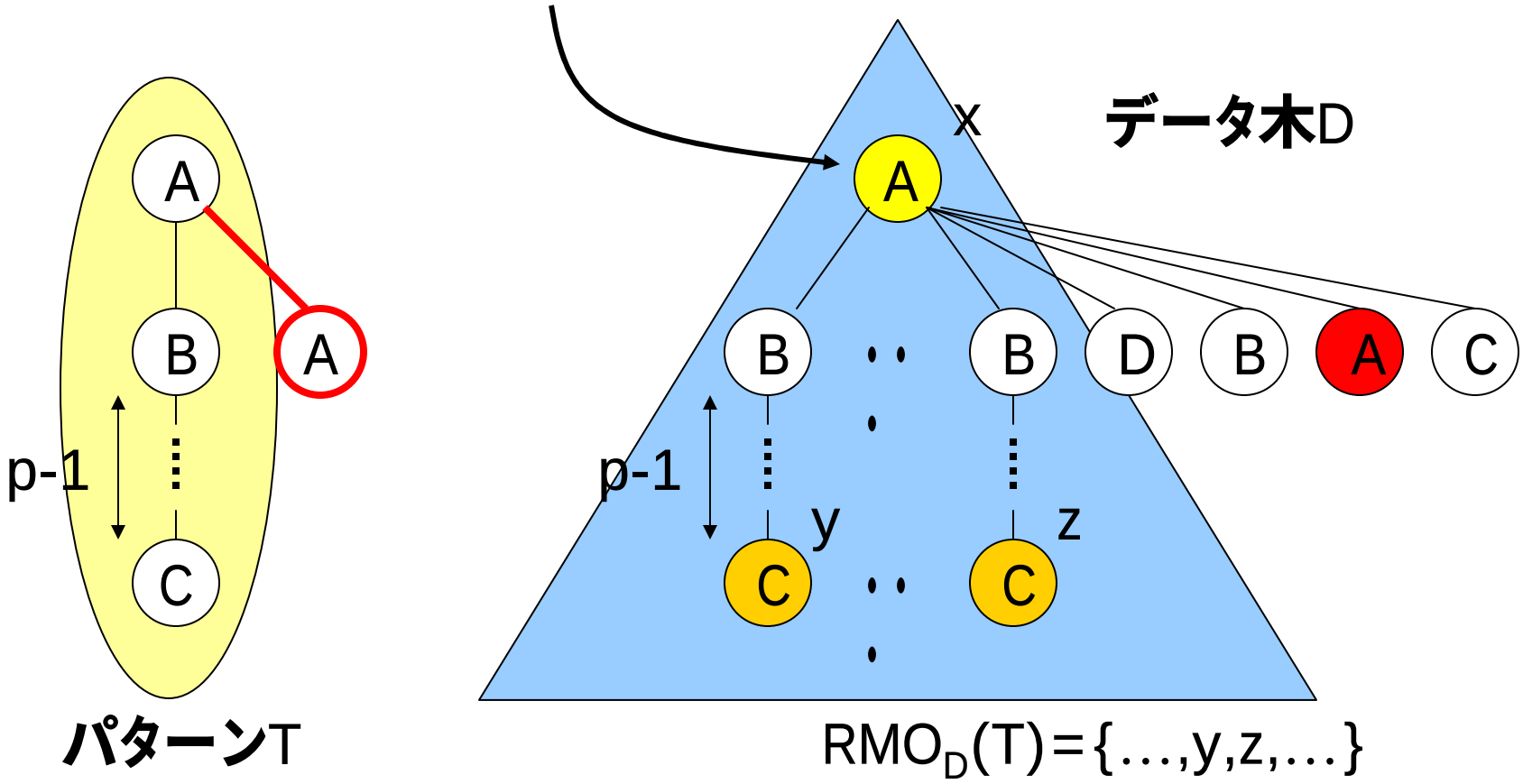
重複した走査の検出

- 更新した最右葉出現リストに重複が生じる場合



重複した走査の検出

走査の前に，p代前の親をチェック



最右葉出現更新アルゴリズム

```
アルゴリズム Update-RMO(OldRMO, p, label)
foreach last∈OldRMO do
        if p=0 then
                vはlastの長男;
        else then
                curはlastの(p-1)代前の先祖;
                vはcurの次弟;  /* vを走査列の先頭にセット*/
        while v ≠ NIL do begin /* vが走査列を右へ走査*/
                if L_D(v) = label then
                        NewRMO = NewRMO∪{v};
                vを次の弟へ;
        end
end
return NewRMO ;
```

補題2

次の(1),(2)を仮定すると，補題2が成り立つ．

(1) 任意の1-パターンTについて，RMO(T)はDのプリオーダー順で並んでおり，

(2) 任意のパターンTについて，アルゴリズムはRMO(T)の順に走査する．

補題2

アルゴリズムが計算する最右葉出現リストRMO(T)には，パターンTのすべての最右葉出現が重複せずに枚挙される．

定理

補題1，補題2より，以下の定理が成り立つ．

定理

データ木Dと最小サポートσに対して，

アルゴリズムFREQTは，すべてのσ頻出パターンを正しく計算する．

アルゴリズムの計算時間

T：大きさ k の順序木パターン (kパターン)

T'：Tの最右拡張

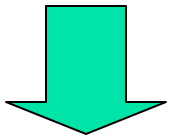
- RMO(T)からRMO(T')を計算する時間：O(kbn)
  - b：データ木Dの最大枝分かれ数
  - n = #RMO(T)

F_k：すべてのσ頻出k-パターンの集合

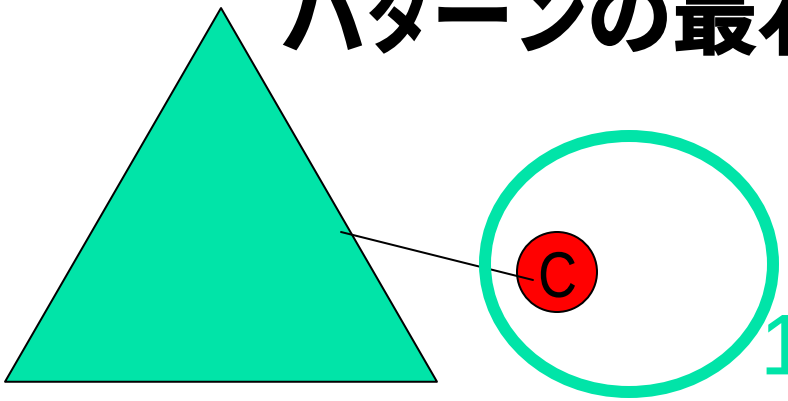
- F_k からF_{k+1} を計算する時間： $\Sigma_{T \in F_k} O(k^2Lbn)$
  - L：データ木Dの異なりラベル数

非頻出1-パターンを用いた高速化

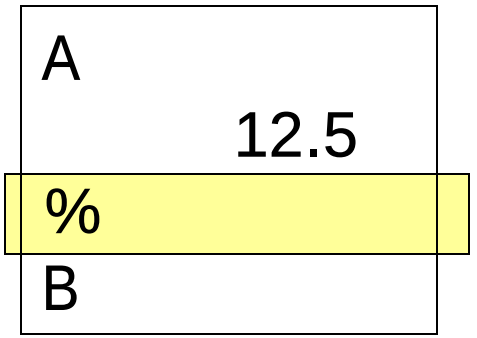
- ある k-パターン T’ は，非頻出1-パターンを含むならば，非頻出



非頻出1-パターンを用いて，非頻出パターンの最右出現リストの更新を省略

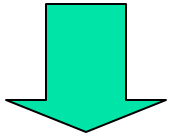


最小サポート
20%

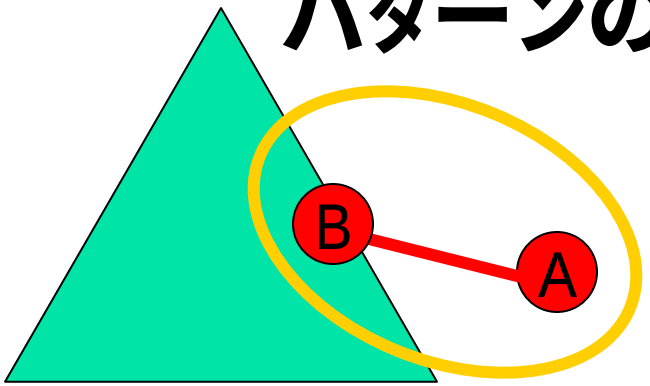


非頻出2-パターンを用いた高速化

- ある k-パターン T’ は，非頻出2-パターンを含むならば，非頻出



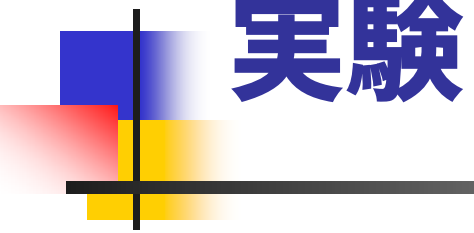
非頻出2-パターンを用いて，非頻出パターンの最右出現リストの更新を省略



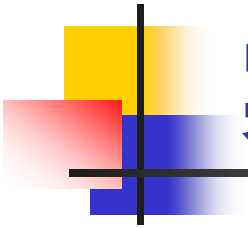
2-パターン

最小サポート
20%

(A, A)
12.5
%
(A, B)
35.0



実験



実装

- Java (SUN JDK 1.3.1)
- DOM**ライブラリ** (OpenXML)
- **実験環境**
  PentiumIII 600MH**z**, 512MB RAM,
  Linux 2.2.4

テストデータ

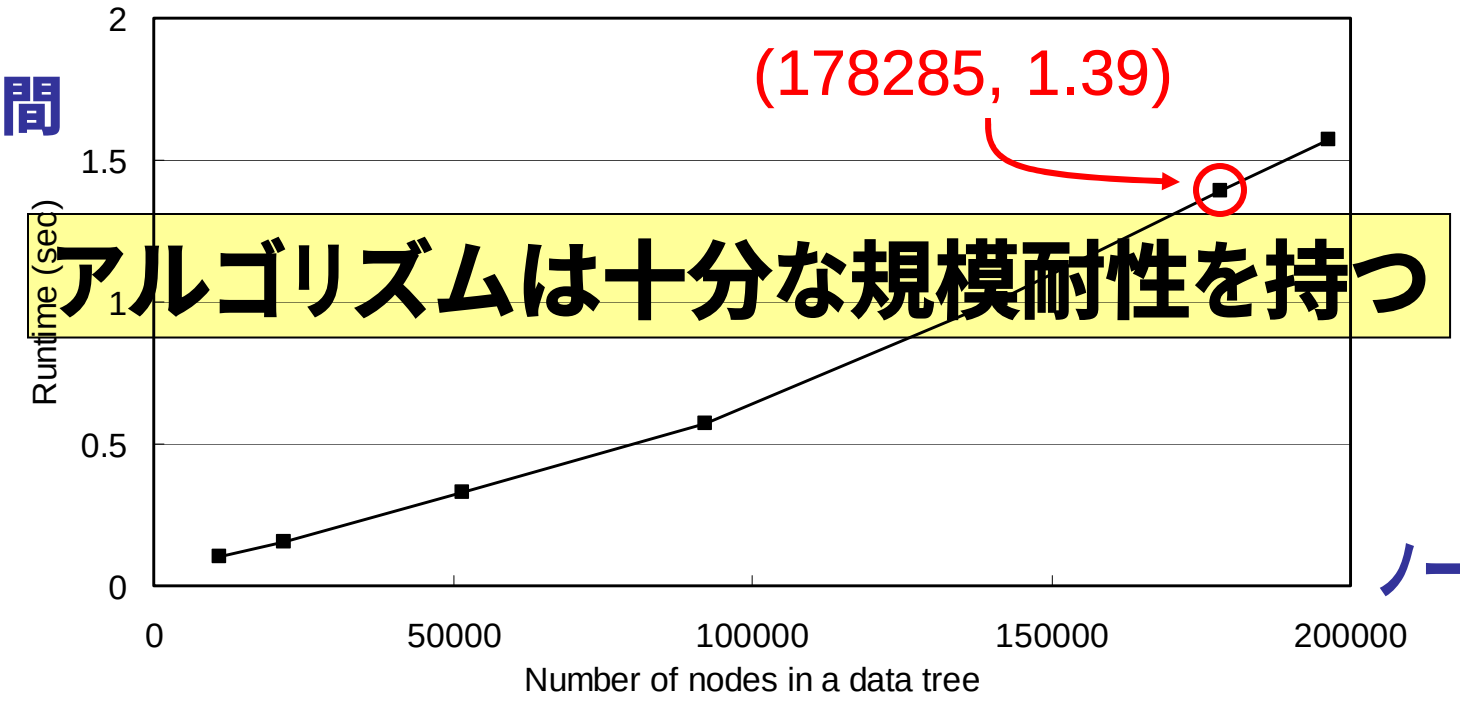
- citeseers：**オンライン論文ウェブサイトに キーワードを与えて得たHTML文書**
  - **文書数**：189
  - **データサイズ**：5.6MB
  - **節点数**：196,427
  - **ラベル数**：7,125

- allsite：**30のウェブデータベースサイトから得た** HTML**文書**
  - **文書数**： 288
  - **データサイズ**：3.6MB
  - **節点数**：192,468
  - **ラベル数**：9,696

規模耐性実験

- テストデータ：citeseers
- 最小サポートを σ =2.0(%) に固定
- データサイズを0.3MBから5.6MBに変化させ，
  それぞれに対する計算時間を測定

実行時間
(秒)



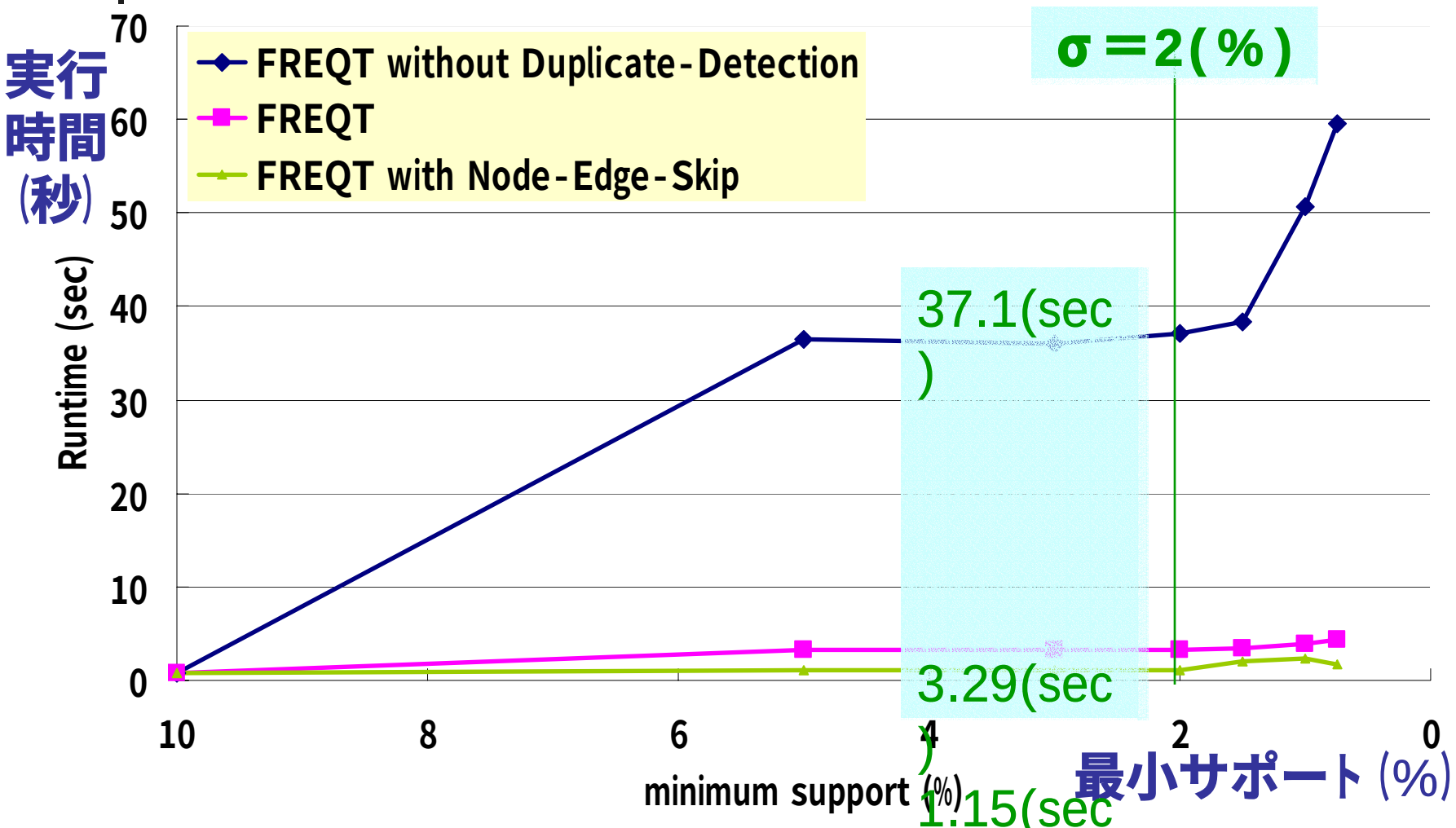
アルゴリズムは十分な規模耐性を持つ

ノード数

高速化手法の比較実験

- **3種類のアルゴリズムを実装**
  - FREQT without Duplicate-Detection
    (重複検出手法なし)
  - FREQT
  - FREQT with Node-Edge-Skip
    (1パターン, 2パターンを用いた高速化手法)
- **テストデータとして** allsites **を使用**
- **サポート値** σ =10.0(%)**から始めて,**
  σを減少させ,σに対する計算時間を測定

高速化手法の比較実験

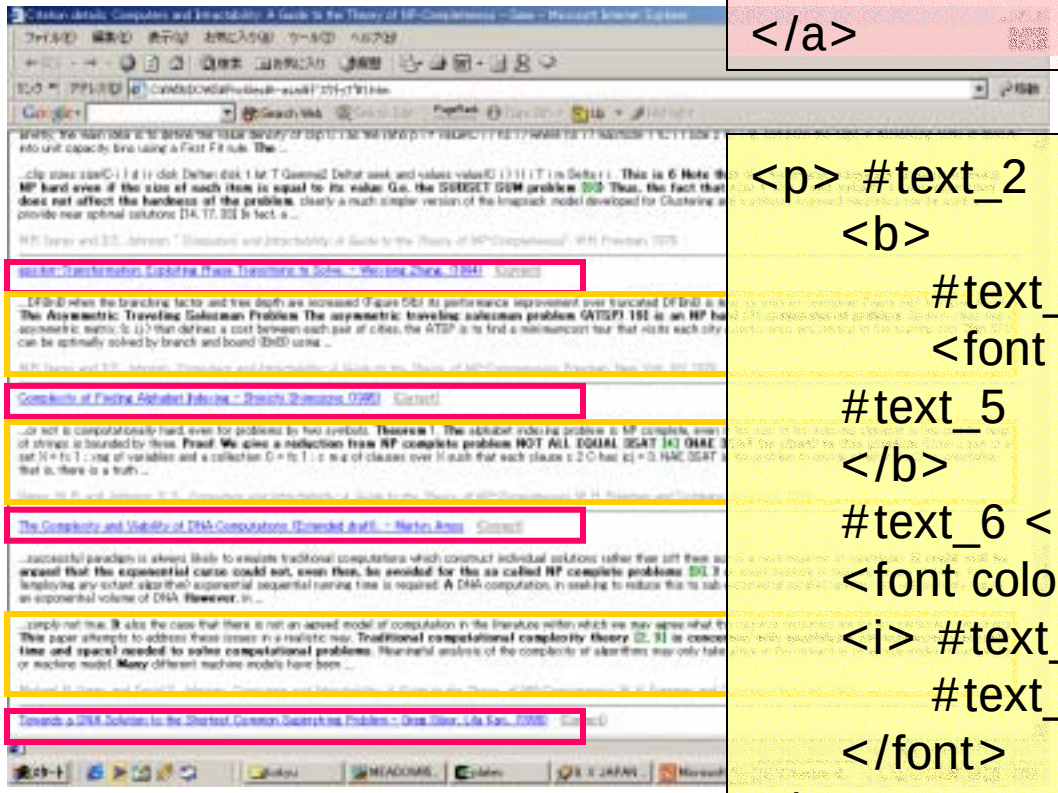

実行時間(秒)
Runtime (sec)
70
60
50
40
30
20
10
0
FREQT without Duplicate-Detection
FREQT
FREQT with Node-Edge-Skip
σ＝2(%)
37.1(sec)
3.29(sec)
1.15(sec
10
8
6
4
2
0
minimum support (%)
最小サポート(%)

高速化手法の比較実験

- σ≦5(%) の低サポート値のとき
  - 重複検出手法は，
    アルゴリズムを10倍以上高速にする．
  - 1-パターン，2-パターンを用いた高速化手法は，
    アルゴリズムを3倍ほど高速にする．
- σ＝10(%)のとき
  - 頻出パターンとして1-パターンしか発見されなかったため，
    計算時間は変わらなかった．

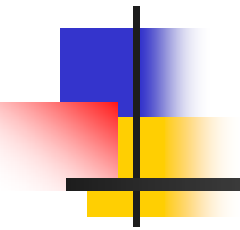
高速化手法は有効

発見された頻出パターン

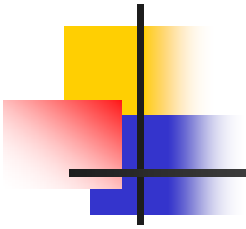


```
<a href="_">
    <font color="#6F6F6F"> #text_1 </font>
</a>
```

```
<p> #text_2
    <b>
        #text_3 <!-- CITE-->
        <font color="green"> #text_4 </font>
    #text_5
    </b>
    #text_6 <br /><br />
    <font color="#999999"> #text_7
    <i> #text_8 </i>
        #text_9
    </font>
</p>
```

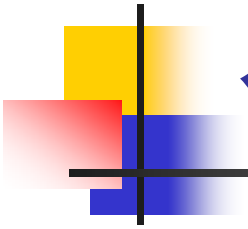


まとめ



まとめ

- 半構造データマイニング問題を定式化
- 効率のよい半構造データマイニング
  アルゴリズムを実現
  - ラベル付き順序木の効率のよい枚挙
  - 最右葉出現リストの更新



今後の課題

- データ木とパターン木の表現力の強化
  - タグの属性，属性値
  - テキスト
- ラベル集合に階層構造を導入
  - (ラベルの) ワイルドカード
- 最適パターン発見問題への拡張