DEIM Forum 2011 D2-6

抽象的な記述が可能なWebからのデータ抽出言語の提案

石井　悠太† 　森嶋　厚行†,†† 　杉本　重雄†,†† 　北川　博之†††

† 筑波大学大学院 図書館情報メディア研究科　〒 305–8550 茨城県つくば市春日 1–2
†† 筑波大学大学院 図書館情報メディア研究科/知的コミュニティ基盤研究センター　〒 305–8550 茨城県つくば市春日 1–2
††† 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
E-mail: †{yishii,mori,sugimoto}@slis.tsukuba.ac.jp, †††kitagawa@cs.tsukuba.ac.jp

あらまし　近年，Web コンテンツを通じた情報発信が広く普及したことに伴い，大量の Web コンテンツの効率的な再利用が重要な問題となっている．本稿は，Web コンテンツの効率的な再利用を実現するためのラッピング言語の提案および評価を行う．本稿で提案する言語の新規性は，既存のラッピング言語でサポートしているような，対象ページの詳細な構造に依存した具体的なラッピング規則だけでなく，対象ページの詳細な構造に依存しない抽象的なラッピング規則の記述が可能であるという点である．これによりユーザーの持つ知識やラッピング規則の再利用に対する要求に応じたラッピング規則の記述が可能となる．

キーワード　Web コンテンツ，ラッピング技術，データ抽出

A Wrapping Language that Allows us to Write Abstract Wrapping Rules for Extracting Data from Web Contents

Yuuta ISHII†, Atsuyuki MORISHIMA†,††, Shigeo SUGIMOTO†,††, and Hiroyuki KITAGAWA†††

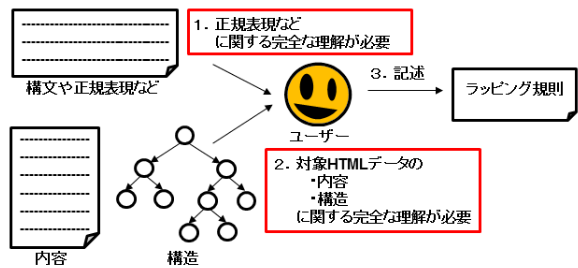
† Grad. Sch. of Library, Information and Media Studies, Univ. of Tsukuba　1–2 Kasuga, Tsukuba, Ibaraki,Japan 305–8550 Japan
†† Grad. Sch. of Library, Information and Media Studies/Research Center for Knowledge Communities, Univ. of Tsukuba., Univ. of Tsukuba　1–2 Kasuga, Tsukuba, Ibaraki,Japan 305–8550 Japan
††† Grad. Sch. of Sys. and Info. Eng., Univ. of Tsukuba　1–1–1 Tennohdai, Tsukuba, Japan 305–8573
E-mail: †{yishii,mori,sugimoto}@slis.tsukuba.ac.jp, †††kitagawa@cs.tsukuba.ac.jp

1. はじめに

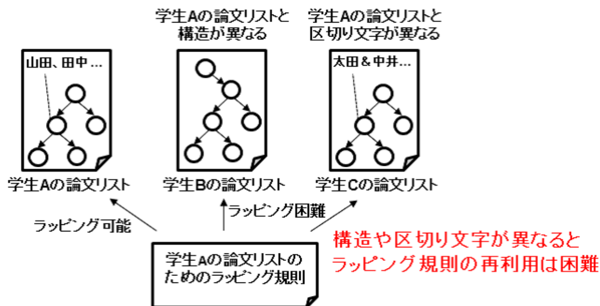
近年，Web コンテンツを通じた情報発信が広く普及したことに伴い，大量の Web コンテンツの効率的な再利用が重要な問題となっている．Web コンテンツの効率的な再利用を実現する方法としては次の 2 つの方法がある．(a)Web コンテンツの管理者が Web コンテンツの構築時に再利用を十分に考慮して DB や API を設計し利用者に提供する方法，(b) 構築済みの Web コンテンツに対して（管理者または利用者が）再利用しやすいように処理する方法．これらのうち，(a) の方法は，Web コンテンツを再利用してもらうことにより利益が発生し得る組織 (例えばショッピングサイトやニュースサイトを持つような組織) 以外ではあまり用いられていない．そのため，既存の Web コンテンツの再利用を行う際には，多くの場合で (b) の方法による処理が必要となる．本稿のトピックであるラッピング技術は，(b) に関する主なアプローチの一つである．

これまでラッピング技術に関しては，多くの研究 [2]〜[11] が行われてきた．ラッピング技術とは，ある形式のデータから他の形式のデータへの変換処理を行うための技術である．また，特定の部分のデータの抽出処理を行う場合もある．ラッピング技術を利用することで，例えば，複数の HTML 形式の論文データから特定の著者の論文データを抽出する処理や，複数の HTML 形式の商品データを統合する処理などを行うことが可能となる．ラッピングを行うソフトウェアをラッパーと呼ぶ．例えば，HTML 形式のデータから一部のデータのみを XML 形式のデータとして抽出するラッパーなどが存在する．ラッピング言語とは，ラッパーの処理を具体的に記述した**ラッピング規則**を記述するための専用の言語である．

本稿では，具体的なラッピング規則だけでなく，抽象的なラッピング規則も記述可能な新たなラッピング言語



問題点 1：正規表現や対象データ等に関する完全な知識が必要



問題点 2：構造や区切り文字が異なると規則の再利用が困難

図 1　既存のラッピング言語の問題点

Wraplet/A(Wraplet with Abstraction mechanisms) を提案する．図 1 に，既存のラッピング言語が完全に具体的なラッピング規則しか書けないために生じる問題を示す．すなわち，(1) ユーザーは正規表現や対象データの詳細な構造 (タグの入れ子構造や区切り文字等) の把握などに関する完全な知識が必要 (図 1(上))，(2) 記述したラッピング規則が対象データの詳細な構造に依存しているため，記述したラッピング規則の再利用が困難 (図 1(下))，という問題がある．一方，Wraplet/A は次の特徴を持つ．(1) ユーザーが正規表現や対象データの詳細な構造などに関する知識が完全でない場合でも知識に応じた利用が可能，(2) 対象データの詳細な構造に依存しないラッピング規則の記述が可能でラッピング規則の再利用が容易．これら 2 つの特徴の詳細については 3. 章で述べる．

Wraplet/A は HTML 形式のデータから XML 形式のデータを抽出するためのラッピング言語である．Wraplet/A の構文は我々の研究グループが以前に提案したラッピング言語 Wraplet [2] の構文がベースとなっている．Wraplet/A ではユーザーの知識や要求に応じて**具体的な (specific) ラッピング規則**または**抽象的な (abstract) ラッピング規則**を混在して記述することができる．Wraplet/A の処理系はこのようなラッピング規則を処理しなければならず，その実現は自明ではない．本稿では，GNFA (Generalized Nondeterministic Finite Automaton) [1] を用いた Wraplet/A の処理系を提案する．

本稿の構成は次の通りである．2. 章では関連研究について述べる．3. 章では提案言語である Wraplet/A について述べる．4. 章では提案言語の評価のための実験と，その結果を示す．5. 章はまとめと今後の課題である．

2. 関連研究

Web コンテンツのラッピング技術については，これまで多くの研究が行われてきた．既存の研究は大きく分けて (A) ラッパーの構築支援，(B) ラッパーの自動構築，という 2 つに分類することができる．次からそれぞれの関連研究を説明する．

(A) ラッパーの構築支援に関する研究．まず，ラッパー構築を支援するためのラッピング言語に関する研究として，Wraplet [2], [3] や W4F [4] などがある．これらの研究で提案されているラッピング言語では，具体的なラッピング規則しか記述できないという問題点があった．それに対して，本稿で提案する言語は，具体的なラッピング規則だけでなく，抽象的なラッピング規則の記述が可能である．

次に，教師あり学習によるラッパー構築に関する研究として，Kushmerick らの研究 [5] や三井らの研究 [6] などがある．これらの研究では，ユーザーは抽出したいデータの範囲を指定したテストデータを用意し，そのテストデータを使って学習させたラッパーを利用することにより Web コンテンツからデータを抽出することになる．これらの研究では，大量の定型 Web コンテンツが存在することを前提としているのに対して，本研究などのラッピング言語アプローチではそのような前提は置かないため，定型・非定型問わずに Web コンテンツからのデータ抽出が可能である．

また，GUI を用いたインタラクティブなラッパー構築に関する研究として，XWRAP [7] や Irmak らの研究 [8] などがある．これらの研究では，正規表現や対象 Web コンテンツの構造などの，ラッピングに関する知識があまりないようなユーザーでも利用できるように設計されている場合が多く，利用に関する敷居は比較的低い．しかし，次のような点で本研究とは異なる．(1) 構築されたラッパーは，常に具体的なラッピング規則に基づいたラッパーであり，類似のサイトに対してそれらを再利用することは困難である．(2) これらはあくまでもインタラクティブなラッパー構築の研究であり，その生成物 (ラッピング規則を記述した式等) は，利用者が直接書いたり変更したりすることは想定されていない．

(B) ラッパーの自動構築に関する研究．教師なし学習によるラッパーの自動構築に関する研究として，次のようなものが存在する．すなわち，文字列の繰り返しに焦点を当てて Web コンテンツの区切り文字を発見しコンテンツ部分を抽出することを目的とした山田らの研究 [9] や，定型 Web コンテンツからテンプレート部分とコンテンツ部分を分離させるためのラッパーの自動構築を行うことを目的とした Arasu らの研究 [10]，ニュースサイトや商品カタログなど特定のドメインに絞ることによりラッパーの自動構築を行うことを目的とした増山らの研究 [11] などがある．これらの研究では定型 Web コンテンツや特定のドメインなどに対するラッパーしか構築できないが，それに対して本稿で提案する言語では，定型・非定型や特定のドメインの Web コンテンツなどに縛られずに柔軟にラッパーを構築することが可能である．

3. ラッピング言語 Wraplet/A

本章では，本稿で提案するラッピング言語である Wraplet/A について説明する．Wraplet/A は，HTML 形式のデータから

```
1:    <html>
2:        ...
3:        <h3>プロフィール</h3>
4:        <ul>
5:            <li>名前：筑波太郎</li>
6:            <li>出身地：茨城県つくば市</li>
7:        </ul>
8:        ...
9:        <h3>論文一覧</h3>
10:       <ul>
11:           <li>筑波太郎「×××の提案」2009 年 3 月.</li>
12:           <li>筑波太郎「△△△の提案」2008 年 12 月.</li>
13:           ...
14:       </ul>
15:       ...
16:   </html>
```

```
1:    <論文一覧>
2:        <論文>
3:            <著者>筑波太郎</著者>
4:            <タイトル>×××の提案</タイトル>
5:            <日付>2009 年 3 月</日付>
6:        </論文>
7:        <論文>
8:            <著者>筑波太郎</著者>
9:            <タイトル>△△△の提案</タイトル>
10:           <日付>2008 年 12 月</日付>
11:       </論文>
12:       ...
13:   </論文一覧>
```

図 2　ラッピングの対象 h(上) 抽出したいラッピング結果 x(下)

```
1: <論文一覧>:#ul(2)/{
2:     <論文>:#li/[
3:         <著者>:_val([^「]*?)「,
4:         <タイトル>:_val([^」]*?)」,
5:         <日付>:_val([^\.]*?)
6:     ]
7: }
```

(a) 具体的な Wraplet/A 式の一例

```
1: <論文一覧>:/{[
2:     <論文>:/(
3:         <著者>:$name,
4:         <タイトル>:$title,
5:         <日付>:$date
6:     )
7: ]}
```

(b) 抽象的な Wraplet/A 式の一例

図 3　図 2(下) のラッピング結果を抽出するための Wraplet/A 式の例

XML 形式のデータを抽出するためのラッピング言語である．先述したように，具体的なラッピング規則だけでなく，抽象的なラッピング規則を混在して記述することができる．Wraplet/A を用いて記述したラッピング規則のことを Wraplet/A 式と呼ぶ．以下では，利用例，記述可能な抽象的な Wraplet/A 式，処理系について順に説明する．Wraplet/A の具体的な構文に関しては付録に示す．

3.1　Wraplet/A の利用例

利用例として，ある大学の研究室に所属する学生が作成した Web コンテンツ（図 2(上) の HTML 形式のデータ h）から，論文情報（図 2(下) の XML 形式のデータ x）を抽出する例を説明する．

他のラッピング言語とは異なり，Wraplet/A はユーザーの知識や再利用の要求に応じて具体的なラッピング規則と抽象的なラッピング規則を混在して記述することができる．したがって，h から x を抽出するための Wraplet/A 式として複数の Wraplet/A 式が考えられる．例えば，図 2(上) の HTML 形式のデータから図 2(下) のラッピング結果を抽出するための Wraplet/A 式としては，図 3 のように複数の例が考えられる．図 3(a) は具体的なラッピング規則のみで記述された Wraplet/A 式 (以下では**具体的な Wraplet/A 式**と呼ぶ) であり，図 3(b) は具体的なラッピング規則と抽象的なラッピング規則の混在した Wraplet/A 式 (以下では**抽象的な Wraplet/A 式**と呼ぶ) である．

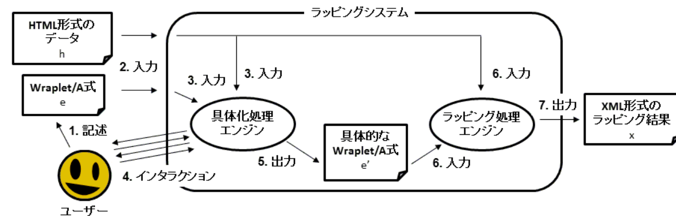


図 4　Wraplet/A を用いたラッピングの手順

図 3 の式について詳しく説明する前に，まず Wraplet/A を用いたラッピングの手順 (図 4) について説明する．(1) 利用者が，ラッピング対象となる HTML 形式のデータ h に対する Wraplet/A 式 e を記述し，e と h をシステムに入力する．(2) 入力された e が抽象的であった場合，システムは利用者にインタラクティブに問合せを行いながら，具体化した Wraplet/A 式 e' を作成する．ただし，最初の入力 e が既に具体的であった場合は，$e' = e$ となり何もしない．また，Wraplet/A 式に含まれる抽象的なラッピング規則が多いほど，インタラクションの回数やシステムの処理時間が増加する．(3) 具体化された Wraplet/A 式 e' と HTML 形式のデータ h がラッピング処理エンジンに渡され，ラッピング結果を出力する．

次に，図 3 の Wraplet/A 式について，順番に簡単に説明する．

具体的な Wraplet/A 式 (図 3(a))． まず，先頭から始まる<論文一覧>:#ul(2)/{...} は次を表す．(1) 入力 h の先頭から 2 番目の <ul> 要素 (図 2(上) の 10〜14 行目) にマッチし，出力 x に < 論文一覧 >(図 2(下) の 1 行目) を追加する．(2) マッチした文字列に対して，/{...} 中の式を繰り返し適用する．

次に，2 行目から始まる<論文>:#li/[...] は次を表す．(1) 親の Wraplet/A 式でマッチした文字列中の <li> 要素 (図 2(上) の 11 行目) にマッチし，x の < 論文一覧 > の子要素に < 論文 >(図 2(下) の 2 行目) を追加する．(2) その <li> にマッチした文字列に対して，[...] 中の式を順に一度ずつ適用する (これは繰返しではない)．

次に，3 行目の<著者>:_val([^「]*?)「, は次を表す．(1) 要素の中の文字列の先頭から“「 ”以外の文字の繰り返しの直後に“「 ”が現れる文字列 (図 2(上) の 11 行目の“ 筑波太郎「 ”) にマッチする．(2) x の < 論文 > の子要素に < 著者 >((図 2(下) の 3 行目) を追加する．(3) _val(...) の... 部分のマッチ結果である“ 筑波太郎 ”を < 著者 > の値とする．

最後に，4 行目の<タイトル>:_val([^」]*?)」, および 5 行目の<日付>:_val([^\.]*?) は 3 行目と同様の意味を表す．そして 6 行目の] および 7 行目の } はそれぞれ対応する左括弧の右括弧である．

このように，具体的な Wraplet/A 式は，HTML タグの入れ子構造や具体的な正規表現を利用してラッピングを行う．したがって，内容が類似していても HTML タグの構造や区切り文字などが若干でも異なるような HTML 形式のデータに対しては図 3(a) の式をそのまま再利用する事はできない．

抽象的な Wraplet/A 式 (図 3(b))． 一方，抽象的な Wraplet/A 式は，具体的な式と似ているが，パターンの具

体的な指定を省略したり，後述するライブラリでサポートしているパターン部品を指定することができる．例えば，図 3(a) の式の 2 行目に存在した`#li`の指定が省略されており，順序列を表す `[ ... ]` の変わりに順不同の列を表す `( ... )` が指定されている．また，正規表現の代わりに人名 (`$name`), タイトル (`$title`), 日付 (`$date`) といった，ライブラリを用いた記述がある．ここで，“ <著者>:$name ”というのは“ :#d(*), <著者>:_val(#name) ”の省略記法であり，任意の区切り文字列 (:#d(*)) の後に，名前辞書 (#name) にマッチする文字列が現れた場合，マッチした文字列を <著者> 要素の値として出力 (_val(#name)) する，という意味である．このように，図 3(b) の式は，図 3(a) の式とは異なり，図 2(上) のデータに依存していないため，HTML タグの構造や区切り文字などが異っていても，同様の抽象的な構造を持つ Web コンテンツ全体を対象としたラッピング式となっている．抽象的な Wraplet/A 式は，後述する具体化処理時に，インタラクティブに具体的な Wraplet/A 式へ変換される．Wraplet/A で記述可能な抽象的な Wraplet/A 式については次節で説明する．

3.2 抽象的な Wraplet/A 式

抽象的な Wraplet/A 式を次のように定義する．

定義． Wraplet/A 式 e が，部分式として以下のいずれかの**抽象式**を含む時，e を**抽象的な Wraplet/A 式**であると言う．

（1） 列展開：抽出データの出現回数を省略した式

（2） 列挙： 抽出データの出現順序を省略した式

（3） 区切り記号：具体的な区切り文字列の省略 □

以下では，それぞれの抽象式について説明する．

(1) 抽出データの出現回数の省略． これは，Wraplet/A の**列展開**“ {[...]} ”を用いて表現する．列展開を用いることで，出現回数を抽象化した規則を記述できる．例えば，“ <商品ランキング>:/{[<商品>:/[<順位>:$num, <商品名>:$goods]]} ”という抽象的な Wraplet/A 式は，長さ未定の固定長の列を表す．上記の式を用いることにより，抽出対象 h_a からは h_a に掲載されている商品ランキングを全て抽出したいが，抽出対象 h_b からは h_b に掲載されている商品ランキングの上位数件のみをを抽出したい，というような要求を一つの抽象的な Wraplet/A 式でカバーすることが可能である．

(2) 抽出データの出現順序の省略． これは，Wraplet/A の**列挙**“ (...) ”を用いて表現する．列挙を用いることで，出現順序を抽象化した式を記述する事ができる．列挙について，図 3(b) の抽象的な Wraplet/A 式の例を用いて説明する．図 3(b) の式は列挙型の子要素として人名 (`$name`), タイトル (`$title`), 日付 (`$date`) という 3 つの要素を記述している．列挙型の子要素は順不同の列なので，抽出対象 h の中に“「○○○の提案」筑波太郎 2010 年 3 月 ”のようなデータがあるか，もしくは“ 2010 年 3 月「○○○の提案」筑波太郎 ”のようなデータがあるか，といった順序には言及していない．

(3) 具体的な区切り文字列の省略． これは，Wraplet/A 式の**区切り記号**“ #d(*) ”を用いて表現する．区切り記号を用いることで，区切り文字列を抽象化した規則を記述する事ができる．例えば，<商品>:/[<発売日>:_val(#date), :#d(*), <価格>:_val(#price)] という抽象的な Wraplet/A 式を記述した場合には，抽出対象のデータによって発売日と価格の間の区切り文字列が具体的に何かについては言及していない．したがって，上記の式を用いることにより，ラッピング対象のデータが”...2010/09/03，2000 円...”のようなデータであった場合でも”...
2010/09/03
2000 円 ...”のようなデータであった場合でもデータが抽出される．

以上の，列展開，列挙，区切り記号は，単一で用いるだけでなく，それぞれを組み合わせて記述することも可能である．以下では，部分式としてこれらの抽象式を多数含む Wraplet/A 式を**抽象度の高い** Wraplet/A 式と呼び，あまり含まない Wraplet/A 式を**抽象度の低い** Wraplet/A 式と呼ぶ．

3.3 Wraplet/A の処理系

ここでは，図 4 の具体化処理エンジンとラッピング処理エンジンが行うそれぞれの処理について説明する．

3.3.1 具体化処理

具体化処理では，入力として与えられた Wraplet/A 式 e が抽象的な Wraplet/A 式であった場合，e をラッピング対象の HTML 形式のデータ h に対する具体的な Wraplet/A 式 e' に変換する．

前節で述べたように，Wraplet/A では，抽出対象のデータの出現回数，出現順序，区切り文字列の 3 点に関して省略した抽象的なラッピング規則が記述可能である．これらの抽象的な記述が存在する場合には，出現回数や出現順序，区切り文字列などが異なる複数のラッピングが考えられるため，それらの中から適切な物を選択する必要がある．本処理では，それぞれの場合のラッピング結果 (以下では**ラッピング結果の候補**と呼ぶ) をユーザーに提示し，ユーザが選択するというアプローチを取る．

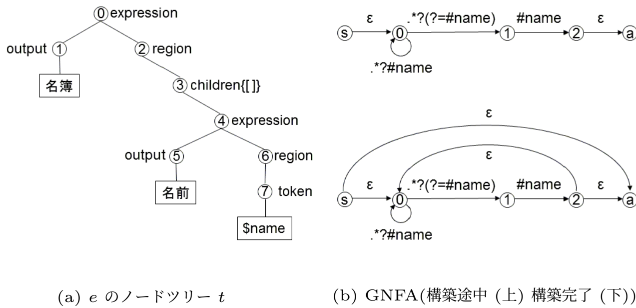
Wraplet/A 式 e とラッピング対象 h が与えられたとき，ラッピング結果の候補数 $|C_e|$ は次のようになる．

$$|C_e| = \begin{cases} 1 & (e \text{ が具体的な Wraplet/A 式の場合}) \\ n \ (>1) & (e \text{ が抽象的な Wraplet/A 式の場合}) \end{cases}$$

すなわち，e が h に対する具体的な Wraplet/A 式の場合は，ラッピング結果は一つしか存在しないため，ラッピング結果の候補数 $|C_e|$ は 1 になる．e が h に対する抽象的な Wraplet/A 式の場合は，ラッピング結果として複数の可能性があるため，ラッピング結果の候補数 $|C_e|$ は 1 より大きな整数になる．

本処理系では，ラッピング結果の候補の集合 C_e を計算するために，GNFA(各状態遷移が任意の正規表現に対応する非決定性有限オートマトン) を用いた処理を行う．

GNFA について説明する．GNFA は (S, Σ, T, s_0, a) から構成される．各構成要素について順に説明する．S は状態の有限集合である．Σ は入力文字の有限集合である．T は状態遷移の集合で，各状態遷移 $t \in T$ は $(s_{from} \in S,\ s_{to} \in S,\ r)$ のトリプルである．ここで，s_{from} は遷移元の状態，s_{to} は遷移先の状態，r は遷移条件を表す．入力文字が遷移条件 r を満たした場合，s_{from} から s_{to} に遷移する．遷移条件 r は正規表現もしくは ε であり，ε は入力文字を受け取らずに遷移することを表す．$s_0 \in S$ は初期状態である．$a \in S$ は受容状態である．GNFA



(a) e のノードツリー t　　(b) GNFA(構築途中 (上) 構築完了 (下))

図 5　GNFA 構築処理の例

は，初期状態 s_0 に対して文字列を与え，状態遷移 t に従って状態遷移を行い，受容状態 a まで遷移した場合，入力文字列を受容したことになる．

本処理系では，次の手順で処理を行う．

(1) GNFA 構築処理．まず，入力された Wraplet/A 式 e に対して構文解析を行い，e に含まれる抽象式 a_i ごとに対応する GNFAg_{a_i} を構築する．**(2) 候補列挙処理**．次に，構築した GNFAg_{a_i} に対して HTML 形式のデータ h を与えてラッピング結果の候補の集合 C_{a_i} を列挙する．候補列挙処理のポイントは，GNFA に対して h を入力として与えた際に，受容状態まで遷移するパス p_j が存在すれば，そのパスに対応するラッピング結果の候補 c_{p_j} が存在する，ということである．ここで，パス p_j は状態遷移 t のリストである．**(3) 式変換処理**．最後に，ラッピング結果の候補 C_{a_i} をユーザーに提示し，ユーザーに適切な候補の選択を求め，選択された候補のパス情報を元に，抽象的な Wraplet/A 式を具体的な Wraplet/A 式に変換する．

以下では，それぞれの処理ついて説明する．

(1) GNFA 構築処理． 本処理では，入力された式 e に含まれる抽象式 a_i ごとに，対応する GNFAg_{a_i} を構築する．ただし，抽象式が入れ子になって現れる場合は，それらに対してまとめて一つの GNFA が構築される．具体的には，次の手順で処理を行う．(1) Wraplet/A 式 e を構文解析 (構文は付録に添付) し，ノードツリー t を構築，(2) t を深さ優先で探索し，抽象式に対応する部分木のを発見した場合は，GNFA 構築ルールに従って対応するノードの GNFA を構築．

例として，“ < 名簿 >:/{[< 名前 >:$name]} ”という抽象的な Wraplet/A 式が入力された場合に構築される GNFA を考える．この Wraplet/A 式の意味は，“ < 名前 >:$name ”の {[...]}(列展開) を < 名簿 > の子要素として出力する，となる．

この Wraplet/A 式を構文解析すると図 5(a) のようなノードツリー t が得られる．t を走査しながら，ノード n が抽象式を表す部分木のルートノードに対応するとき，n に次で説明する GNFA 構築アルゴリズム (makeGNFA) を適用すると，抽象式を持つノード　をルートとする部分木に対して，図 5(b) 下のような GNFA が構築される．

GNFA 構築アルゴリズム． 図 6 に GNFA 構築アルゴリズムを示す．makeGNFA は，抽象式のノードに対応する GNFA を返す関数である．makeGNFA の引数は次の 2 つである．すなわち，(1) 走査対象ノード n，(2) 区切り記号の状態遷移の作成待ち状態 (後述) であることを表すフラグ f (以下，区切り記号フラグ)，ただし，区切り記号フラグは，makeGNFA を再帰的に呼び出す場合のみ true にする可能性があり，t の走査中に呼び出すときは，false である (すなわち，外からは makeGNFA(n, false) を実行する)．makeGNFA の返り値は，GNFA と，入力と同じく区切り記号フラグの組である．このフラグは，再帰的に makeGNFA を適用するときの入力として利用される．

```
1:    Pair(GNFA, Flag) makeGNFA(n, f){
2:        g:={};
3:        if (f){
4:            g.add(n を用いて図 7(c) の GNFA を構築したもの);
5:            f:=false;
6:        }
7:        switch type(n) {
8:        case 列展開:
9 :           g':={}:
10:           for n' in n.children() {
11:               r := makeGNFA(n', f);
12:               g'.add(r.getGNFA);
13:               f := r.getFlag;
14:           }
15:           g.add(g' を用いて図 7(a) の GNFA を構築したもの);
16:       case 列挙:
17:           g':={};
18:           for sequence in permutation(n.children()) {
19:               for n' in sequence.elements() {
20:                   r := makeGNFA(n', f);
21:                   g'.add(r.getGNFA);
22:                   f := r.getFlag;
23:               }
24:           }
25:           g.add(g' を用いて図 7(b) の GNFA を構築したもの);
26:       case 区切り記号:
27:           f:=true;(図 7(c) の GNFA を構築するためのフラグ);
28:       case 具体表現：
29:           g.add(具体表現に対応する GNFA を構築したもの);
30:       }
31:       return g, f;
32: }
```

図 6　GNFA 構築アルゴリズム

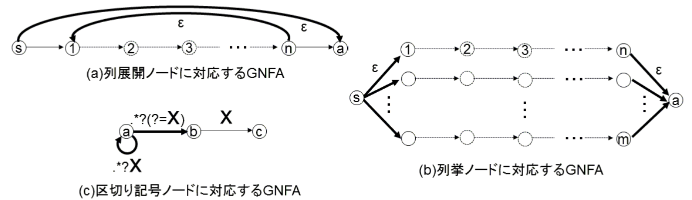


図 7　抽象的な記述に対応する GNFA の構築

makeGNFA は走査中のカレントノード n が表す式の種類に応じて，それぞれの式に対応した GNFA の構築処理を行う．2 行目の g は，出力する GNFA を 状態遷移の集合として表現している変数である．3〜6 行目は区切り記号の GNFA の構築待ちのフラグ f が true 時の処理であるが，これについては後ほど説明する．

7〜30 行目はノード n の種類に応じた処理である．それぞれの処理について順に説明する．

8〜15 行目は，n が列展開ノードだった場合の処理である．9〜14 行目で，列展開ノードを root とする部分木に対応する GNFA g' を構築する．15 行目では，9〜14 行目で構築された g' に対して，次のように 2 つの状態遷移を追加する．追加する 2 つの状態遷移について，図 7(a) を用いて説明する．① →...→ ⓝ は g' である．状態遷移 ① $\overset{\varepsilon}{\longleftarrow}$ ⓝ は，① →...→ ⓝ の状態遷移を繰り返すために追加する状態遷移であり，列展開ノード

の子要素の出現回数が 2 回以上のラッピング結果の候補を列挙するための状態遷移である．また，状態遷移 ⓢ $\xrightarrow{\varepsilon}$ ⓐ は，① →...→ ⓝ の状態遷移の内容に関わらず，初期状態から受容状態に遷移するために追加するの状態遷移であり，列展開ノードの子要素の出現回数が 0 回のラッピング結果の候補を列挙するための状態遷移である．

16〜25 行目は，n が列挙ノードだった場合の処理である．16〜24 行目で，列挙ノードの子要素の順列の計算を行い，各順列に対応する GNFA g' を構築する．25 行目では，17〜24 行目で構築された GNFA に対して，$2p$ (p は列挙ノードの子要素の順列の総数) 個の状態遷移を追加する．追加する状態遷移について，図 7(b) を用いて説明する．平行に並んだ ① →...→ ⓝ，◯ →...→ ◯，◯ →...→ ⓜ は各順列に対応する GNFA である．初期状態からそれぞれの GNFA の先頭の状態への状態遷移 (図 7(b) 左部分の太線) および，それぞれの GNFA の末尾の状態から受容状態への状態遷移 (図 7(b) 右部分の太線) は，各順列に対応した GNFA を統合し，ラッピング結果の候補を列挙するための状態遷移である．

26〜27 行目は，n が区切り記号ノードだった場合の処理で，次に現れるノードに応じた GNFA を構築するために，区切り記号の GNFA の構築待ちフラグを立てる．その理由は，図 7(c) に示すように，区切り記号ノードに対応する状態遷移 (太字部分) は，区切り記号ノードの後に出現するノードの内容 (大文字の X で示した正規表現) に依存するからである．

28〜29 行目は，n が具体表現ノードだった場合の処理で，具体表現に対応する GNFA を追加する．

(2) 候補列挙処理． 本処理では，構築した GNFA を用いてラッピング結果の候補を列挙する．ラッピング結果の候補を列挙するために，GNFA 構築処理で構築した GNFA に対する入力として，HTML 形式のデータ h を与える．h により，初期状態から受容状態まで遷移するパス p_i が存在すれば，そのパスに対応するラッピング結果の候補 c_{p_i} が存在する．したがって，パスの総数 $|P|$ とラッピング結果の候補数 $|C|$ は等しくなる．

図 5(b) 下の GNFA を用いて説明する．⓪ → ⓪ および ⓪ → ① は，区切り記号 (:#d(*)) に対応する．この二つの状態遷移により任意の区切り文字列を持った複数のラッピング結果が生成される．① → ② は，出力する x の < 名前 > 要素の値 (< 名前 >:_val(#name)) に対応する．② → ⓪ および ⓢ → ⓐ は，列展開ノード ({[...]}) に対応する状態遷移であり，② → ⓪ は，x の < 名簿 > 要素の子要素の < 名前 > 要素を列構造として 2 個以上展開する場合に対応する．ⓢ → ⓐ は，x の < 名簿 > 要素の子要素の < 名前 > 要素を一つも持たない場合に対応する．

例えば，図 5(右) に対する入力 h の中に 2 つの名前 (A 氏と B 氏) が含まれていた場合，次の 4 つのラッピング結果の候補が生成される (“ [...] ”で囲まれた文字列は GNFA における状態遷移のパスを表す)．(c_1)A 氏のみを抽出する候補 [ⓢ → ⓪ → ① → ② → ⓐ]，(c_2)B 氏のみを抽出する候補 [ⓢ → ⓪ → ⓪ → ① → ② → ⓐ]，(c_3)AB 両氏を抽出する候補 [ⓢ → ⓪ → ① → ② → ⓪ → ① → ② → ⓐ]，(c_4)AB を両氏とも抽出しない候補 [ⓢ → ⓐ]．全ての部分文字列が処理された段階で，ラッピング結果の候補の集合 $C = \{c_1, c_2...\}$ が生成される．この例の場合 $|C|$=4 となる．

```
<名簿>:/[
     :.*?(?=#name),
     <名前>:_val(#name)
]
```

(b) c_1 を抽出するための式

```
<名簿>:/[
     :.*?(?=#name), <名前>:_val(#name),
     :.*?(?=#name), <名前>:_val(#name)
]
```

(b) c_3 を抽出するための式

図 8　具体的な Wraplet/A 式

次に，ラッピング結果の候補の集合 C をユーザーに対して提示し，適切な候補 $c_i \in C$ の選択を求める．ただし，C に含まれる全ての候補の中から単一の候補をユーザーに選択してもらうのは困難である．そこで，出現回数や出現順序，区切り文字列の内容などを順番に選択してもらい，候補の絞り込みを行う．

(3) 式変換処理． 本処理では，選択された候補 c_i を抽出するために抽象的な記述を具体的な記述に変換する．具体的な記述への変換は，選択された候補 c_i の状態遷移のパスの情報をもとに行う．上記で説明したように，図 5(b 下) に対する入力 h の中に A 氏と B 氏という 2 つの名前が含まれていた場合，4 つの候補が生成される．この 4 つの候補の中から，A 氏のみを抽出する候補 (c_1) を選択した場合および，AB 両氏を抽出する候補 (c_3) を選択した場合に変換される具体的な Wraplet/A 式 e' はそれぞれ図 8 のようになる．

(c_3) を選択した場合に変換される図 8(b) の式について説明する．前述したように，この候補のパスは [ⓢ → ⓪ → ① → ② → ⓪ → ① → ② → ⓐ] である．⓪ → ① は“ :.*?(?=#name)，”に対応する．① → ② は“ < 名前 >:_val(#name) ”に対応する．② → ⓪ は“ < 名前 > ”の展開個数に対応する．これらの対応関係および候補のパスに基づいて，入力された Wraplet/A 式を具体的な Wraplet/A 式に変換する．

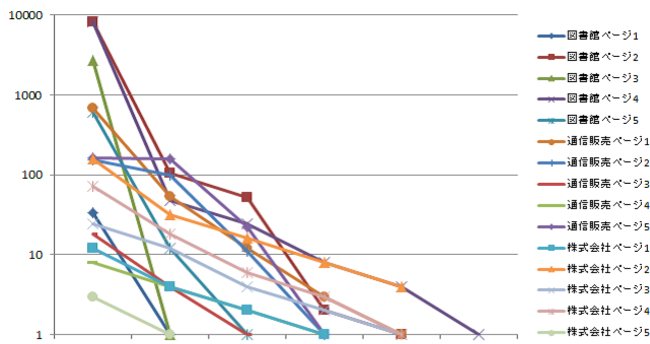
3.3.2　ラッピング処理

本プロセスでは，具体化プロセスで生成された具体的な Wraplet/A 式 e' を用いて，ラッピング対象 h に対してラッピングを行い，XML 形式のデータ x を抽出する．処理手順は Wraplet [2] に記載されている内容と同様である．具体的には，次の処理手順でラッピングを行う．(1) 具体的な Wraplet/A 式 e' を木構造に変換する，(2) 木構造の root ノードから深さ優先で走査する，(3)<delimiter> または <pattern> に対応するノードがあった場合はラッピング対象 h とのマッチングを行う (子ノードのための Wraplet/A 式の評価は親ノードでマッチングに成功した文字列をマッチングの対象として行う)，(4) マッチングが成功したら対応する <output> を XML 形式のデータ x に要素として追加 (<pattern> に_val() が含まれていた場合はマッチした文字列を要素の値として追加する)，(5) 全てのノードを走査し終わるまで (3)〜(4) を繰り返す．

4. 評　　価

ユーザーの持つ知識や再利用の要求に応じて，抽象度の高い Wraplet/A 式から具体的な Wraplet/A 式まで記述可能である，

表 1　実 験 結 果



という Wraplet/A の特徴について評価した．具体的には，4 種類の Web ページを対象として抽象度の高い Wraplet/A 式を記述し，次の評価を行った．(1) 記述した Wraplet/A 式が処理系を通じて具体的な Wraplet/A 式に変換可能か，(2) Wraplet/A 式の抽象度や Web ページの種類に応じて，インタラクション回数やラッピング結果の候補数にどのような変化があるか．本実験では抽象的な Wraplet/A 式の記述に必要なライブラリは備えているという状況で実験を行った．

4.1 実 験 方 法

実験は次の手順で行う．(1) 実験対象の Web ページの選択，(2) 実験対象の Web ページに対するラッピング要求の仮定，(3) ラッピング要求に応じた抽象度の高い Wraplet/A 式の記述，(4) 抽象度の高い Wraplet/A 式から具体的な Wraplet/A 式への処理系を用いた変換．それぞれの手順について次で説明する．

(1) 実験対象の Web ページの選択．本実験では公共と商用という大きく分けて 2 つのドメインの Web ページを対象とした．具体的には，公共 Web ページとして (p_1) 図書館，(p_2) 研究室，を対象とし，商用 Web ページとして (c_1) 通信販売，(c_2) 株式会社，を対象とした．具体的な Web ページの選択には検索エンジンを用いて，“ 図書館 ”，“ 研究室 ”，“ 通信販売 ”，“ 株式会社 ”という各キーワードに関連する Web ページを検索し，検索結果の上位 50 件の中から不適切な検索結果を除去した上で，無作為に選択した 5 件を対象 Web ページとした．

(2) ラッピング要求の仮定．本実験では次のようなシナリオに基づいたラッピング要求を仮定した．すなわち，(p_1) からの書籍情報の抽出，(p_2) からの論文情報の抽出，(c_1) からの商品情報の抽出，(c_2) からの企業情報の抽出，を行うというラッピング要求をそれぞれ仮定した．

(3) 抽象度の高い Wraplet/A 式の記述．手順 (1) で選択した各 Web ページに対して，手順 (2) で仮定したラッピング要求に基づいて抽象度の高い Wraplet/A 式の記述を行った．

(4) 具体的な Wraplet/A 式への変換．手順 (3) で記述した抽象度の高い Wraplet/A 式および対象 Web ページを入力として処理系に与え，具体的な Wraplet/A 式への変換過程におけるインタラクション回数やラッピング結果の候補数を記録した．

4.2 実 験 結 果

実験結果を表 1 に示す．表の横軸はインタラクションの回数で，縦軸はラッピング結果の候補数である．

(1) 抽象的な式から具体的な式への変換．実験の結果，(p_1)，(c_1)，(c_2) に関しては，仮定したラッピング要求に基づいた抽象的な Wraplet/A 式を具体的な式 Wraplet/A 式に変換可能なことが確認できた．しかし，(p_2) に関しては，仮定したラッピング要求に基づいた抽象的な Wraplet/A 式を具体化処理しようとしたところ，ユーザーとの間の現実的なインタラクションの時間内で，ラッピング結果の候補の列挙をすることができなかった．

(2) インタラクション回数とラッピング結果の候補数の変化．抽象的な Wraplet/A 式から具体的な Wraplet/A 式に変換可能であった各 Web ページにおいて，インタラクションの回数に応じてラッピング結果の候補数が減少し，抽象度の高い Wraplet/A 式が抽象度の低い (具体的な)Wraplet/A 式へと徐々に変換できていることが確認できた．

4.3 考　　察

(p_2) からの論文情報の抽出という要求を満たすような，抽象度の高い Wraplet/A 式を処理系に与えた場合に，ラッピング結果の候補の列挙処理が完了しなかった原因について考察する．(p_2) では論文情報を抽出するために“ < 論文情報 >:/{[< 論文 >:/(< 著者 >:/{[< 名前 >:$name]}, < タイトル >:$title ...)]} ”という抽象度の高い Wraplet/A 式を記述した．論文数を n とすると，< 論文 > 情報の抽出の組み合わせだけで少なくとも $\sum_{k=1}^{n} {}_nC_k$ 以上の組み合わせを計算する必要がある．今回実験対象とした 5 つのページには少なくとも数十件の論文情報があり，膨大な処理が必要になる．さらに，各 < 論文 > 情報に含まれる < 名前 > や < タイトル > に関しても，区切り記号および列挙を利用して記述しているため，膨大な数の組み合わせを計算する必要がある．

本論文では，抽象的な Wraplet/A 式に対するラッピング結果の候補をすべて計算した上で，ユーザーにインタラクションを求めるというアプローチを提案した．しかし，このアプローチは (p_2) からの論文情報の抽出のようにラッピング結果の候補数があまりにも膨大になるような抽象的な Wraplet/A 式の変換のためのアプローチとしては適切ではないと考えられる．

この問題に対する解決策としては，いくつかの方法が考えられるので，それぞれの方法について次で述べる．

まず，インタラクションのタイミングを変更するという方法が考えられる．具体的には，ラッピング結果の候補を計算する際に，ユーザーに適宜インタラクションを求めて候補の絞り込みを行い，ラッピング結果の候補の計算数を抑えるという方法である．これにより，すべてのラッピング候補を計算していた場合に比べて，ラッピング結果の候補の計算処理時間が短縮する．この方法の問題点は，次の 2 点が考えられる．1 点目は，本論文で提案した手法に比べてインタラクションの回数が増加してしまう可能性が高いという点である．2 点目は，インタラクションを行った結果，最終的にラッピング結果の候補が存在しないような場合が考えられるという点である．したがって，ラッピング対象のデータおよび Wraplet/A 式の内容によっては適切な方法ではない場合も考えられる．

次に，GNFA の分析を行い，ユーザーによる Wraplet/A 式の再記述をサポートするという方法が考えられる．これは，Wraplet/A 式の具体化処理の際に，ラッピング結果の候補数が一定数以上存在した時点で，GNFA の分析を行い，ラッピング結果の候補数の爆発の原因を特定し，爆発の原因となっている抽象的な記述の部分を具体化するようにユーザーに求めるという方法である．この方法の問題点は，ユーザーが正規表現などに関するある程度の知識がない場合には具体化するのが困難であるという点である．しかし，ユーザーが正規表現などに関するある程度の知識がある場合には有効な方法であると考えられる．

最後に，より抽象度の低い省略記法をサポートするという方法が考えられる．本論文で提案した Wraplet/A は，抽象的なラッピング規則の記述を実現するために，3 つの省略記法を持つ．これら 3 つの省略記法を用いて記述した抽象的なラッピング規則は，ラッピング対象が複雑な構造や多くのデータを持つ場合，抽象度が高すぎる記述になってしまい，ラッピング結果の候補数が爆発してしまう．そこで，3 つの省略記法だけでなく，より抽象度の低い省略記法を提供するという方法が考えられる．これにより，ラッピング対象のデータの内容やユーザーの要求などに応じて，より細かい粒度で抽象的な Wraplet/A 式から具体的な Wraplet/A 式までの記述を実現できるようになり，ユーザーの工夫次第でラッピング結果の候補数の爆発を抑えた抽象的な Wraplet/A 式の記述が可能になると考えられる．

5. まとめと今後の課題

本稿では，具体的なラッピング規則だけでなく，抽象的なラッピング規則を記述可能なラッピング言語 Wraplet/A の提案を行った．抽象的なラッピング規則の記述を許すことで，ユーザーの持つ知識やラッピング規則の再利用に関する要求に応じた柔軟なラッピング規則の記述が可能になる．今後の課題としては，抽象的な Wraplet/A 式を具体的な Wraplet/A 式へ変換するための仕組みの再検討や，提案言語の利用の容易さを評価するための実験などが挙げられる．

謝　辞

ゼミ等においてコメントいただきました，筑波大学大学院図書館情報メディア研究科 阪口哲男准教授，永森光晴講師に感謝いたします．本研究の一部は科学研究費補助金特定領域研究 (#21013004)，基盤研究 (A)(#21240005)，基盤研究 (B)(#19300081)，若手研究 (B)(#20700076) による．

文　献

[1] Bakhadyr Khoussainov,Anil Nerode. Automata theory and its applications. Birkhäuser Boston, 2001.

[2] Natsumi Sawa, Atsuyuki Morishima, Shigeo Sugimoto, Hiroyuki Kitagawa. Wraplet: Wrapping Your Web Contents with a Lightweight Language. Proc. IEEE SITIS' 2007.

[3] 石井悠太，森嶋厚行，杉本重雄，北川博之．ラッピング技術を用いた Web サイトの再構築手法の提案．第 149 回 DBS 研究会, 2009-DBS-149(13), pp. 1-8, 2009-11.

[4] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy web data-source using W4F. In The VLDB Journal, 1998.

[5] Nicholas Kushmerick. Wrapper Induction: Efficiency and Expressiveness. Articial Intelligence, Vol. 118, pp. 15-68, 2000.

[6] 三井健，岩沼宏治，鍋島英知．老若男女だれでも簡単に使える HTML 文書ラッパ自動合成システム．電子情報通信学会技術研究報告, 人工知能と知識処理 103(243), 43-48, 2003.

[7] Ling Liu, Calton Pu, Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In IEEE Int. Conf. on Data Engineering 2000.

[8] Utku Irmak and Torsten Suel. Interactive Wrapper Generation with Minimal User Effort. International World Wide Web 15th, 2006.

[9] 山田泰寛，池田大輔，廣川佐千男．半構造化文書に対する木構造と文字列を組み合わせたラッパーの自動生成法．情報処理学会研究報, Vol.2003, No.98, pp.115-122, 2003.

[10] Arvind Arasu, Hector Garcia-Molina. Extracting Structured Data from Web Pages. In ACM SIGMOD, pp. 337-348, 2003.

[11] 増山友美，松井藤五郎，大和田勇人．Wrapper を用いたデジタルカタログからの情報抽出．第 22 回 人工知能学会全国大会, 3B1-03, 2008.

付　　録

Wraplet/A の構文

```
基本となる構文の定義
    <expression>::=[<output>]':'[<region>]
output に関する定義 (抽出結果の形式を指定)
    <output>::='<'(<letter> | ':' | '_')
                    {<letter> | <digit> | <combine> | <extender>
                    | ':' | '_' | '-' | '.'} '>'
    <letter>::=半角英字 | 全角ひらがな | 全角カタカナ | 表意文字（漢字など）
    <digit>::=半角数字
    <combine>::=結合文字（濁点やウムラウトなど）（CombiningChar）
    <extender>::=カタカナに使う長音、々、ゝなど、他の文字の間やうしろに付く文字
region に関する定義 (抽出対象の範囲を指定)
    <region>::=[<pattern>]['/'<children>] | <token> | <delimiter>
pattern に関する定義
    <pattern>::=<pattern>+
    <pattern>::='#'['('<number>')']
                  ( <defined> | 'not('<pattern>')' | '<'<tag>'>' )
    <pattern>::='_val('<pattern>')'
    <pattern>::='_children('<pattern>')'
    <pattern>::=<char>[<quantity>]
    <pattern>::=<pattern> '|' <pattern>
    <pattern>::='('<pattern>')'[<quantity>]
    <tag>::=HTML タグ s.t body, div, ul
    <char>::=<basechar> | '['<basechar>+']' | '[^'<basechar>+']'
    <quantity>::='+'['?'] | '*'['?'] | '?'
                  | "{<number>"[","[<number>]]["?"]"}"
    <number>::=<digit>+
    <basechar>::=任意の 1 文字 (メタ文字除く) | \メタ文字
children に関する定義 (入れ子構造を指定)
    <children>::='{'<expression>'}' //繰り返し
                    |'{['<expression>{',' <expression>}']}' //列展開
                    |'['<expression>{',' <expression>}']' //列 (順序固定)
                    |'('<expression>{',' <expression>}')' //列挙 (順序可変)
token に関する定義
    <token>::='$'<defined>
    <defined>::=定義済みのトークン名 (name,number,date,price etc...)
delimiter に関する定義
    <delimiter>::='#d(*)'|'#d('<number>')'
補足事項
    <output> と <region> に関する補足事項
        <output> がない場合:
            <region> に記述可能なのは <delimiter> のみ
        <region> がない場合:
            <region>=".*"と解釈 (正規表現の全文一致の意味)
    <token> に関する補足事項
        <token> は区切り文字を補完しつつ値を出力する
        <token> が連続して出現した場合:
            #d(*), _val(#defined), #d(*), _val(#defined), #d(*), ... と解釈
        <token> に続けて <pattern> が出現した場合:
            #d(*), _val(#defined), #d(*), <pattern> ... と解釈
    繰り返し“ {...} ”に関する補足事項
        繰り返しの子要素に抽象的な表現は許さない
        抽象的な表現を繰り返したい場合は列展開構造を利用する
```