XMLとスキーマ

日本IBM (株) 東京基礎研
国際大学研究所
村田　真

目標

- スキーマ設計とはどんなものか
  - XMLを用いて複雑な情報を管理するには，スキーマによって秩序を見出すことが必須だが，スキーマを完成させることにこだわりすぎると藪知らずに落ち込む．
- XMLスキーマ言語の理論的な基礎
- スキーマのモジュール化技法
- プログラミング言語研究への期待

XML

Simplicity wins, Efficiency loses

XMLとは

- インターネット時代の文書/データ記述言語
- SGML(ISO8879)のサブセット
- アプリ固有の要求に合わせて拡張可能
- WWWクライアント側でのアプリやデータベースの操作対象

XML文書

- タグを含んだテキスト
- 要素: 開始タグと終了タグの対
  - 段落
- 要素の木構造
  - <文書>
    <章> <節>…</節> <節>…</節> </章>
    <章> <節>…</節> <節>…</節> </章> … </文書>
  - 幅についての制限なし、深さについての制限なし
  - 要素の順番は意味をもつ
- 要素は属性を持つ
  -

なぜXMLにするか

- XMLでいちおう表現できる
- 完成度の高いXMLパーサがどこにでも無料で入手できる
- 徹底的な国際化がされている
- 人間にとって読みやすく、理解しやすい。
- XMLはテキストとして扱える

S式との違い

- S式にはintegerがあるがXML 1.0にはない
- S式はLISPからだけしか扱えないが，XMLは誰でも使える．
- S式は) で終わるが，XMLは</foo>で終わる．
- S式の用途は狭い
  - 電子政府、電子商取引に使われたか？
  - 文書作成に使われたか？
- 後述するスキーマ言語はS式にはない

なぜスキーマが必要か？

- XMLそれ自体は何でも許してしまう
  - 要素や属性の名前にほとんど制限はない
    - <だんらく>これは段落</だんらく>
    - <直線>長さは3cm</直線>
  - 要素の階層構造も何でも許される
    - <html><meta>これもXML文書</meta></html>

スキーマを使う三つの理由

- 厳密かつ簡潔であって、人間に読める記述で、マークアップ語彙(XHTML, SVG, etc.)を規定するため. [文書化]
- XML文書が、このマークアップ語彙で確かに書かれているかどうか確かめるため [*検証*].
- このマークアップ語彙のための応用プログラムを容易に構築するため [*データバインディング*].

スキーマと意味？

- スキーマは、文書の意味を規定しない
- XML文書全体の集合の部分集合を規定しているだけ
- DTDに適合する文書を扱うスタイルシート、ソフトウェア、人間の総体が意味を規定する

別の見解

- スキーマは，integerすら持たないXML文書を、型情報を備えたオブジェクトへと昇華させる
  - 効率は負ける
  - 誤ったレイヤリングである

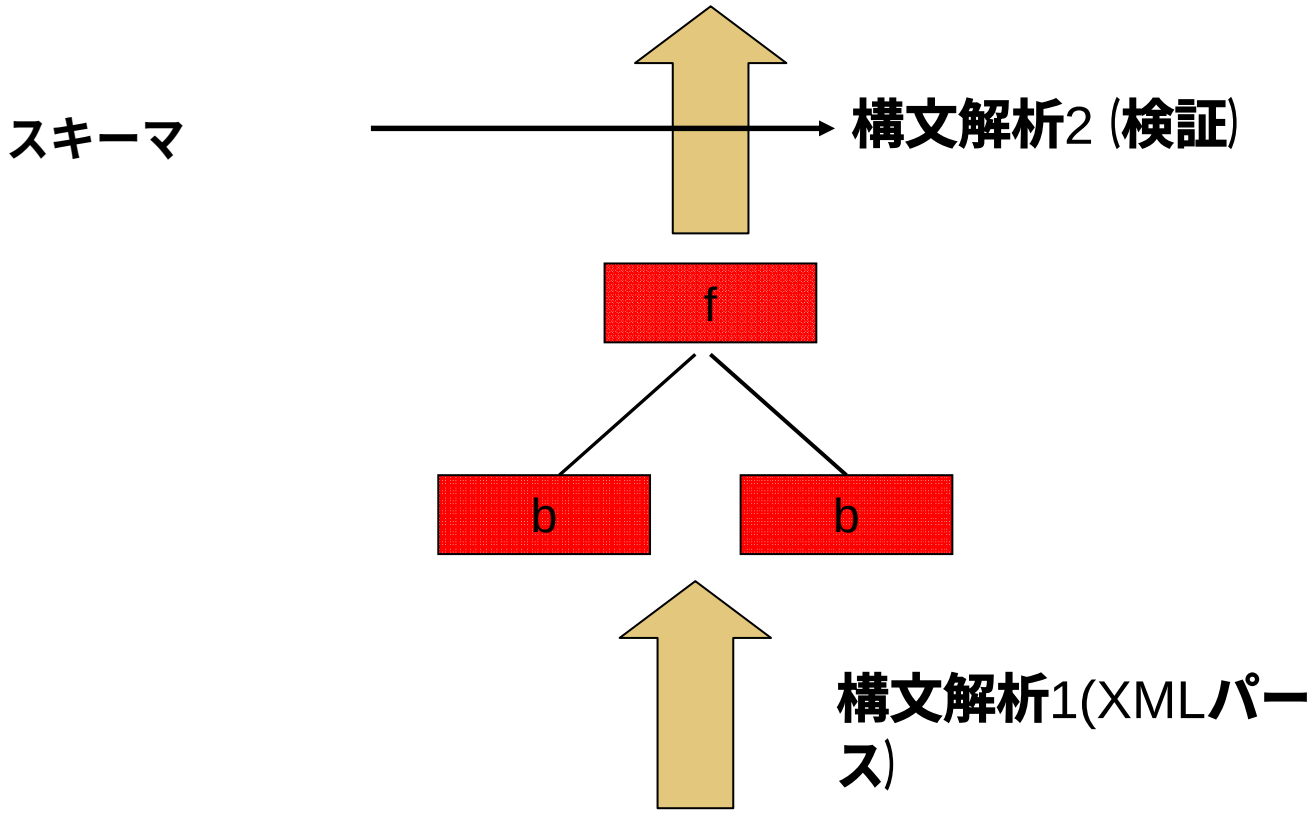
スキーマは本当に重要？

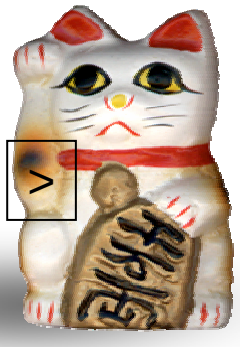
- 重要ではない
  - どうせプログラミングの役にはそれほど立たない
- 重要である
  - XMLで扱う情報は桁違いに複雑だし、オープンな環境でまったく違う人が操作する

スキーマの理論

- 正規木言語、木オートマトン
  - 厳密な形式言語理論がもっとも実用的である!

二つの構文解析





第一レベルの文法 (固定)

```
document  ::=  element
element   ::=  EmptyElemTag | STag content ETag
STag      ::=  '<' Name (S Attribute)* S? '>'
ETag      ::=  '</' Name S? '>
content   ::=  CharData? (element CharData?)*
EmptyElemTag  ::= '<' Name (S Attribute)* S? '/>'
Attribute  ::=  Name S? '=' S? AttValue
AttValue  ::=  '"' [^<&"]* '"' |  "'" [^<&']* "'"
Name  ::=  (Letter | '_' | ':') (NameChar)*
CharData  ::=  [^<&]* - ([^<&]* ']]>' [^<]&
```

第二レベルの文法 (スキーマ)

- 木を扱う
- マークアップ語彙ごとに異なる文法

- 正規木文法を使う

正規文法（復習）

- 生成規則は、
  n ::= a n’
  の形であるか
  n ::= a
  の形である
  - n
    非終端記号
  - a
    終端記号
  - n’
    非終端記号

正規木文法

- 生成規則は、n ::= a [exp] の形である
  - n
    非終端記号
  - a
    終端記号
  - exp
    非終端記号からなる正規表現

両者の比較

正規木文法の規則

```
n ::= t [exp]
```

正規文法の規則

```
n ::= t n'

   or

n ::= t
```

問題

① 文字列abacだけを生成する正規文法を作れ。

② XML文書
<doc><a><b/><a><c/></doc>
だけを生成する正規木文法を作れ。

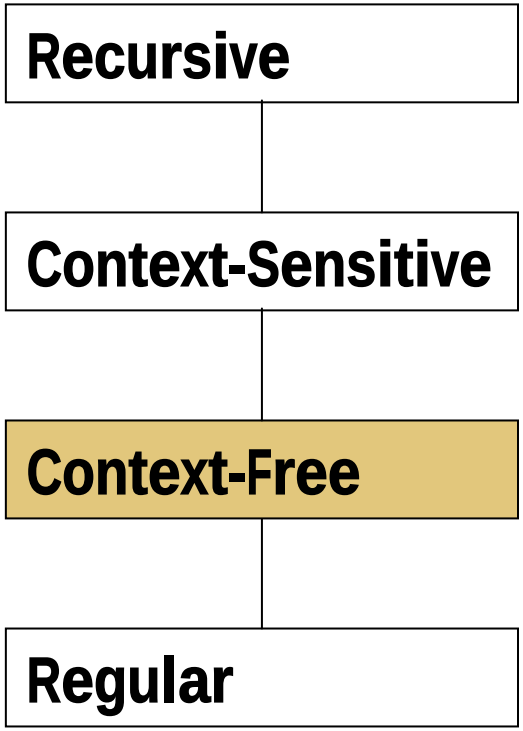
文脈自由(列)文法ではダメな理由

- 生成規則は
  `n ::= exp`
  の形
- 文脈自由文法で、構造化文書のスキーマは十分だと1995年頃まで信じられていた。

例

DTD	拡張文脈自由文法
<!ELEMENT 目次 (章*)>	目次 ::= 章*
<!ELEMENT 章 (#PCDATA)>	章 ::= #PCDATA

Chomsky Hierarchy



疑問点

- 文を解析して構文解析木を得るなら自然だった (プログラミング言語，自然言語)
- 構文解析木を解析して構文解析木を得る？？

深刻な問題点

- 重要で当たり前なのに書けないことがある
- ブール演算ができない
- スキーマ変換(or 型推論)ができない

書けないこと

- 間接的な入れ子の禁止/許可
  - …<a>…<a>… … … …
  - <form>… ..<form>…</form>.. …</form>
  - <form>… …<input>…</input>… …</form>
- 場所によって微妙に違う内容
  - <chapter><title>…</title>…</chapter>
  - <figure><title>…</title>…</figure>
  - <section><title>…</title><para>…</para>…</section>

書けないこと (2)

- 涙ぐましいハック
  - Exception (inclusion/exclusion) of SGML
    - HTML 2.0, HTML 3.2, HTML 4.01, XHTML 1.0の違い
    - TEI
    - DocBook
  - Assertion grammar

局所文法(local grammar)

- 左辺の非終端記号と右辺の終端記号の間に一対一の対応を強制したもの

局所木文法の規則

```
n(t) ::= t [exp]
```

局所文法の規則

```
n(t) ::= t n(t')
   or
n(t) ::= t
```

DTDは局所木文法

- <!ELEMENT foo (bar*)>

- `n(foo) ::=  foo [ n(bar)* ]`

局所木言語と文脈自由文法

局所木言語の規則

```
n(t) ::= t [exp]
```

を以下のように読み直すと

```
t ::= exp
```

文脈自由文法に見えた。

Chomsky Hierarchies

Tree Languages

- Context-Free
- Regular
- Local

String languages

- Context-Free
- Regular
- Local

問題

- aabが以下の有限オートマトンによって受理される様子を示せ。

① 初期状態はs1, 終了状態はs3、遷移関数は次の通り。

- (a, s1, s2)
- (a, s2, s2)
- (b, s2, s3)

② 遷移関数を以下のように変更。

- (a, s1 , s1)
- (a, s1 , s2)
- (b, s2 , s3)

③ 遷移関数をさらに以下のように変更。

- (a, s1, s1)
- (a, s1, s2)
- (b, s2 , s3)
- (a, s2 , s4)
- (b, s4 , s3)

問題

- 以下の文法をそのままオートマトンと見なして実行したとき、文書<a><a><b/>が受理される様子を示せ。木の上から実行する場合、下から実行する場合の両方を考えよ。

① 開始記号はs1, 生成規則は次の通り。

- s1 ::= a[s2]
- s2 ::= a[s2]
- s2 ::= b[ε]

② 生成規則を以下のように変更。

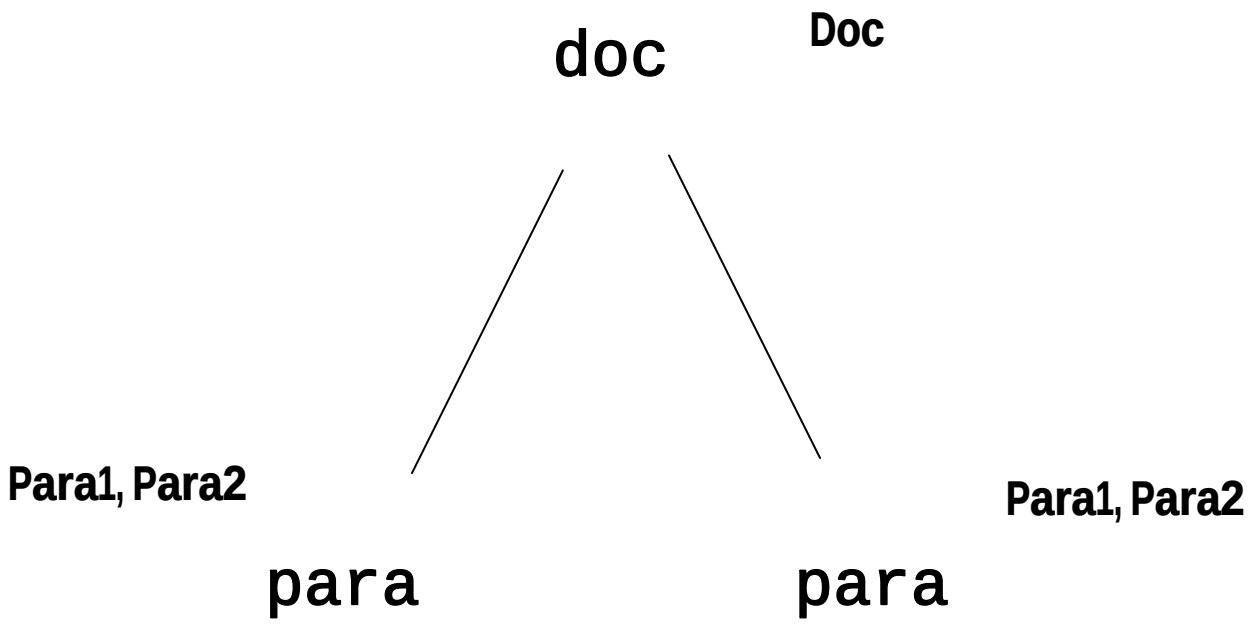
- s1 ::= a [s1]
- s1 ::= a [s2]
- s2 ::= b [ε]

③ 生成規則をさらに以下のように変更。

- s1 ::= a [s1]
- s1 ::= a [s2]
- s2 ::= b [ε]
- s2 ::= a [s4]
- s4 ::= b [ε]

曖昧な正規木文法の例

- `Doc ::= doc [Para1*, Para2*]`
- `Para1 ::= para [ε]`
- `Para2 ::= para [ε]`


doc
Doc
Para1, Para2
para
para
Para1, Para2

分かったこと

- 上から実行することも下から実行することもできる
  - 検証アルゴリスム
- 実行結果が一意になるとは限らない
  - 問題だという立場、問題ではないという立場の両方が存在する

正規木文法および局所木文法の比較

- 表現力は十分か？
- 検証は簡単に作れるか？
- 実行（検証）結果は一意か？
- ブール演算について閉じているか？

ブール演算の用途

- XMLプログラミング言語における型チェック
  - $L1 \subseteq L2$
  - $L1 \cap L2$
- スキーマの上位互換性判定
  - $L1 \subseteq L2$
- 文書変換
  - projection

クラスごとの閉包性

- **Regular**
  - ∩, ∪, ¬ , **projection**
- **Local**
  - ∩

Unionについて閉じない例

- Schema A

```
<!ELEMENT   doc   (sec*)>
<!ELEMENT   sec   (p*)>
<!ELEMENT   p
    (#PCDATA|FNote)*>
<!ELEMENT   FNote
          (#PCDATA)>
```

- Schema B

```
<!ELEMENT   doc   (div*)>
<!ELEMENT   div   (p*)>
<!ELEMENT   p
          (#PCDATA)>
```

Unionについて閉じない例(2)

- これは違う

```
<!ELEMENT   doc    (sec*|div*)>
<!ELEMENT   sec    (p*)>
<!ELEMENT   div    (p*)>
<!ELEMENT   p      (#PCDATA|FNote)*>
<!ELEMENT   FNote  (#PCDATA)>
```

文書変換とスキーマ変換(1)

- 入力

```
<文書>
  <本体>
    <p/>
    <div><p/></div>
  </本体>
  <付録>
    <p/>
    <div><p/></div>
  </付録>
</文書>
```

- 出力

```
<文書>
  <本体>
    <本文段落/>
    <div><本文段落/></div>
  </本体>
  <付録>
    <付録段落/>
    <div><付録段落/></div>
  </付録>
</文書>
```

文書変換とスキーマ変換(2)

- 入力

<!ELEMENT 文書　(本体, 付録)>

<!ELEMENT 本体　(p*, div*)>

<!ELEMENT 付録　(p*, div*)>

<!ELEMENT div　(p*, div*)>

…

- 出力（不正確）

<!ELEMENT 文書　(本体, 付録)>

<!ELEMENT 本体 (本文段落*, div*)>

<!ELEMENT 付録 (付録段落*, div*)>

<!ELEMENT div ((本文段落 | 付録段落)*, div*)>

…

スキーマ作成の実際

簡単な例

- 氏名
  - <名前><姓>村田</姓><名>真</名></名前>
- 住所
  - <住所><郵便番号>0123-34</郵便番号><都道府県>神奈川</都道府県><市町村>川崎</市町村><続き>中原区…</続き></住所>
- この程度では済まない

OASISのXAL(Extensible Address Language)

- Postbox
- Floor
- Room
- StreetName
- AdditionalStreetName
- BuildingName
- BuildingNumber
- InhouseMail
- Department
- CityName
- PostalZone
- CountrySubentity
- CountrySubentityCode
- Region
- District
- TimezoneOffset
- AddressLine
- Country
- LocationCoordinate
- AddressLine
- Line

OASISのXAL(Extensible Name Language)

- PersonNameに含まれる情報
- NameLine（Given nameとChristian nameは属性で区別）,
- PrecedingTitle,
- Title,
- FirstName,
- MiddleName,
- NamePrefix,
- LastName,
- OtherName,
- Alias,
- GenerationIdentifier,
- Suffix,
- GeneralSuffix,
- FormerName,
- KnownAs

顧客名

- Customer Name and address Details
- Customer Identifier
- Organisation Details (Branches, Stocks, etc)
- Birth Details
- Age Details
- Gender
- Marital Status
- Language Details
- Nationality Details
- Occupation Details
- Qualification Details
- Passport Details
- Religion Details
- Ethnicity
- Telephone Details
- Facsimile Details
- Cellular Phone Details
- Pager Details
- E-mail Details
- URL
- Financial Account Details
- Identification card Details
- Person Physical Characteristics
- Tax number Details
- Vehicle Information Details
- Family Member Details
- Income Details
- Reference Contact Details
- Hobbies
- Habits
- Residency Details
- Visa Details

モジュール化技法

- DTD
  - パラメータ実体という一種のマクロを使ったトリック
  - 上書き機構のみ (二つの宣言のうち先にあるものが有効)
- RELAX NG
  - モジュール化機構
  - 共存機構 (二つの宣言のどちらも有効)
  - 上書き機構 (モジュールを読み込むときに一部を上書き)

課題1: スキーマの汎用化と特殊化

- 広い範囲の人が合意するスキーマは複雑すぎて作れない
  - 日本の電子政府で必要な範囲の人名スキーマ
- 狭い範囲の人が合意するスキーマは広範囲の情報交換には使えない
  - UN/CEFACTやOASISで制定された人名（国際化）されたもの）とどう折り合いをつければいいのか？
- 双方向変換？

課題2: スキーマ進化

- 古いスキーマ (及びそれに基づくシステム) と新しいスキーマが (及びそれに基づくシステム) インターネット上で共存するには
  - 古いスキーマに基づく文書は，新しいスキーマに照らして妥当か？
  - 新しいスキーマに基づく文書が，古いスキーマに照らして妥当であるようにするにはどうするか？
  - 古いスキーマに従って書かれたプログラムは，新しいスキーマに基づく文書に適用できるか？

参考文献

- XYZ第一回の資料 (http://arbre.is.s.u-tokyo.ac.jp/~hahosoya/xyz/xyz1.html)
- 村田, 川口: XMLとスキーマ言語, コンピュータソフトウェア (掲載予定)
- Murata, Lee, Murali, Kawaguchi: Taxonomy of XML Schema Language using Formal Language Theory, ACM TOIT (投稿中)