Araucariaツールの形式論証から拡張論理プログラムの自動生成

望月 昇[1] 若木 利子[2]
芝浦工業大学 システム工学部*

1 はじめに

最近, araucaria ツール [5] を用いて自然言語表現の論証ダイアグラムから (人手で) 形式論証のダイアグラムを構成した後、その形式論証から EALP のルール集合を生成する方法 [1] が提案された. さらに形式化された論証ダイアグラムを Araucaria の機能を用いて AML(Argument Markup Language) 表現で保存した後, その AML 表現の形式論証から EALP のルール集合を自動生成するアプローチが文献 [2] で言及されているが, 具体的な手続きが示されていない. 本研究では後者の研究課題に関して, AML の文書型定義 (以後 DTD と称す) [5] に基づく ELP/EALP 自動生成の手続きを示す. 形式論証の AML 文書は XML の木構造と見なされるが, 本研究の手続きでは, この XML の木構造を AML の DTD で定義された生成規則に基づく構文解析木とみなし, JAVA API である DOM を用いてこの XML 木を根からトップダウンに辿って, EALP/ELP 表現のルール集合を自動生成する. さらに提案手法を実現した変換プログラム aml2ealp を作成するとともに, 多くの事例を用いて, 提案手法とプログラムの正当性の検証を行った.

2 システム構成

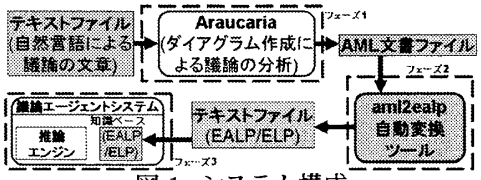

図 1: システム構成

本システム (図 1) は, 次の 3 つのフェーズからなる.

- (フェーズ 1) Araucaria を用いて作成された自然言語の論証ダイアグラムから, 手作業で形式論証のダイアグラムに変換し, AML 文書として保存.
- (フェーズ 2) 本研究の提案手法で実装された aml2ealp 変換ツールを用いて, AML 表現の形式論証から ELP/EALP 表現のルール集合を生成し, ファイルに保存.
- (フェーズ 3) 生成した EALP/ELP を知識ベースに格納して議論エージェントシステムを稼動.

以下に, 文献 [1] に述べられたフェーズ 1 の Araucaria を用いた変換処理を示す.

〈 **Araucaria を用いた変換** 〉

step1: 自然言語の論証ダイアグラムを作成

step2: 足りない前提を補う

step3: 使用する注釈を決める

step4: ノードに格納された自然言語の文章を (注釈付) リテラル表現にし, 形式論証のダイアグラムを構成

ここでは, step1 と step2 で作られたダイアグラムの各ノードに格納されている自然言語の文を手作業で述語論理 (又は命題論理) のリテラルに置換する. 他方, EALP を生成する場合は, リテラル表現された各ノードの対応するラベルフィールド [5] に注釈を登録する.

step5: AML 文書として形式論証を保存

例 1. 図 2 は step4 で得られた形式論証のダイアグラムの例である. この図は, 以下の EALP のルールを表現している.

$$a : t \leftarrow b : t$$
$$b : t \leftarrow d : t,\ not\ e : f$$
$$d : t \leftarrow$$

step5 では図 2 の形式論証ダイアグラムが, 図 3 の AML 木表現を持つ AML 文書としてファイル保存される. 図 3 では複数存在する同じ要素名 (AU 等) のノードをそれぞれ区別するために「要素名 番号」のようにノードを表わしている. なお, 図 3 の木では変換で使用する部分のみを表している.

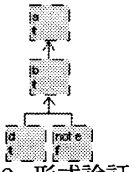

図 2: 形式論証ダイアグラム

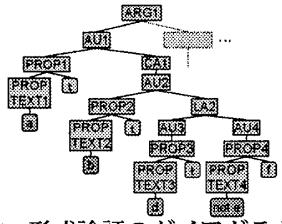

図 3: 形式論証のダイアグラムを保存したの AML 文書木

3 AML 文書から EALP への変換

ツール aml2ealp は形式論証の AML 文書を入力し, EALP/ELP 表現の論証 (ルール集合)[3] を生成する.

3.1 AML 文書と DTD

AML は論証を表現するために Araucaria で定義された Markup Language である. AML 文書の構文を定義する DTD(argument.dtd)[5] の一部を以下に示す.

```
<!ELEMENT ARG (SCHEMESET?, TEXT?, EDATA?, AU?)>
<!ELEMENT AU (PROP, REFUTATION?, (CA | LA)*)>
<!ELEMENT PROP (PROPTEXT, OWNER*,
                INSCHEME*, ROLE*, TUTOR?)>
<!ELEMENT REFUTATION (AU)>
<!ELEMENT CA (AU*)>
<!ELEMENT LA (AU, AU+)>
   …省略…
```

3.2 注釈付き拡張論理プログラム

aml2ealp-変換ツールの出力は, 注釈付き拡張論理プログラム (EALP), もしくは拡張論理プログラム (ELP) である. EALP は以下の形式のルールの集合である.

$$L_0 : \mu_0 \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m$$
$$, not\ L_{m+1} : \mu_{m+1}, \ldots, not\ L_{m+n} : \mu_{m+n}$$

但し, $m, n \geq 0$, L_i はリテラルで原子式 A, またはその論理否定 $\neg A$, not は NAF(negation as failure) のオペレータである. EALP のルールの各注釈付きリテラル $L_i : \mu_i$ (但し, μ_i は注釈) をリテラル L_i で置換したルールの集合が ELP である. (詳細は文献 [1] を参照)

Automatic Translation from Formal Arguments of Araucaria to Extended Annotated Logic Programs

[1]Noboru Mochiduki, [2]Toshiko Wakaki
*Shibaura Institute of Technology

3.3 AML文書からEALP/ELP生成手続き

AML 文書を入力し, AML の DTD に基づいて EALP/ELP ルールを生成する手続き p_ARG を図 4 に示す. p_ARG, p_AU, p_PROP, p_PROPTEXT は, それぞれ AML 木の ARG, AU, PROP, PROPTEXT の要素名 (ラベル) のノードを処理する手続きであり, n_ARG, n_AU, n_PROP, n_PROPTEXT は, それぞれ AML 木の ARG, AU, PROP, PROPTEXT をラベルとして持つノードを表す. p_AU は再帰的手続きである.

手続き p_ARG(root)
入力変数: root (AML 木の根)
step1. root の子の AU ラベルのノードを見つけ, それを n_AU とする.
step2. p_AU(n_AU) を実行. return.

手続き p_AU(Node_AU)
入力変数: Node_AU 返値: String 型
局所変数: Child, Head, BL_i
step1. Node_AU の子を Child とし, 全ての Child について step2 を実行.
step2. (i)Child のラベルが PROP ならば, p_PROP(Child) を実行しその返値を Head に代入. (i.e. Head:=p_PROP(Child))
(ii)Child のラベルが CA または LA ならば,
Child の子の各 AU ラベルのノード:n_AU_i $(1 \leq i \leq k)$ について, p_AU(n_AU_i) を実行, その返値を BL_i に代入.
Head "←" BL_1 ","..."," BL_k なるルールを出力.
(v)Child のラベルが REFUTATION ならば,
Child の子の AU ノードを見つけ, それを n_AU とする.
p_AU(n_AU) を実行.
(vi)Child に CA と LA のものが無く, Head が NAF リテラルでないなら, Head "←" なるルールを出力.
step3. return Head. %呼び出し元の p_AU (ii) で生成される %ルールの本体のリテラルになる.

手続き p_PROP(Node_PROP)
入力変数: Node_PROP 返値: string 型
局所変数: Lit, ano
step1. Node_PROP の子の PROPTEXT ラベルのノードをみつけそれを n_PROPTEXT とする.
step2. p_PROPTEXT(n_PROPTEXT) を実行し, その返値を Lit に代入.
step3. n_PROPTEXT の nodelabel 属性の値を ano に代入. %ELP ならば 注釈が無いので ano は空列になる
step4. Lit ":" ano を return. %呼び出し元の p_AU で生成される %ルールの頭部のリテラルになる.

手続き p_PROPTEXT(Node_PROPTEXT)
入力変数: Node_PROPTEXT ノード 返値: string 型
局所変数: txt
step1. Node_PROPTEXT の子の ♯PCDATA を txt に代入.
step2. return txt %呼び出し先のリテラルになる

図 4: AML 文書から EALP/ELP 生成のアルゴリズム

例 2. AML 表現の形式論証から EALP の自動生成

図 4 のアルゴリズムを図 3 の AML 文書木に適用した実行履歴を以下に示す. まず p_ARG(ARG1) を呼び出し, 以下の様に順次手続きを呼び出しながら木を辿りルールを生成する. 但し, 自然数 i に対して $+i$ *call* p と $-i$ *return* p は深さ i レベルでの手続き p の呼び出しと戻りを表し, <<<<文字列>>>> はファイル出力される時点と出力文字列を表す.

```
+1 call p_ARG(ARG1)
  +2 call p_AU(AU1)
    +3 call p_PROP(PROP1)
      +4 call p_PROPTEXT(PROPTEXT1)
      -4 return a
    -3 return a:t
    +3 call p_AU(AU2)
      +4 call p_PROP(PROP2)
        +5 call p_PROPTEXT(PROPTEXT2)
        -5 return b
      -4 return b:t
      +4 call p_AU(AU3)
        +5 call p_PROP(PROP3)
          +6 call p_PROPTEXT(PROPTEXT3)
          -6 return d
        -5 return d:t
         <<<<output  d:t ←>>>>
      -4 return d:t
      +4 call p_AU(AU4)
        +5 call p_PROP(PROP4)
          +6 call p_PROPTEXT(PROPTEXT4)
          -6 return not e
        -5 return not e:f
      -4 return not e:f
       <<<<output  b:t ← d:t, not e:f>>>>
    -3 return b:t
     <<<<output  a:t ← b:t>>>>
  -2 return a:t
-1 return
```

上図の実行履歴より, 図 2 の論証ダイアグラムが表す論証 $[a:t \leftarrow b:t;\ b:t \leftarrow d:t, not\ e:f;\ d:t \leftarrow]$ を構成するルールが, 右から左の順に順次, 実行時に得られることが確認できる.

4 おわりに

AML 文書表現から EALP/ELP 表現の論証 (ルール集合)[3] を生成する方法の提案と変換ツールの実現, 及び, 種々の事例を用いた自然言語の論証から EALP/ELP への変換の検証を行った. なおツール aml2ealp は, ELP から論証集合と攻撃関係を計算する追加機能があるので, 図 1 で示した議論エージェントシステムのみならず, 議論の意味論を計算する文献 [4] のシステムへの自然言語インターフェースとして用いることも可能である. 今後の予定として, ELP/EALP 表現の論証 (つまりルール集合) から AML 文書表現の論証の生成方法とツールの開発を行うことを考えている.

参考文献

[1] Y.Takahashi , H.Sawamura , J.Zhang: Transforming Natural Arguments in Araucaria to Formal Arguments in LMA, Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology, pp 668-678, 2006.

[2] 高橋洋介, 澤村一: 議論分析ツール Araucaria における自然言語による議論の多値議論の論理における形式的な議論への変換, 合同エージェントワークショップ&シンポジウム 2007.

[3] R. Schweimeier, M. Schroeder: A Parameterised Hierarchy of Argumentation Semantics for Extended Logic Programming and its Application to the Well-founded Semantics, Theory andPractice of Logic Programming, pp.207-242, 2005.

[4] 伊藤 浩太 , 若木 利子: 解集合プログラミングによる議論の意味論の計算, 情報処理学会 第 70 回全国大会, 2008 年 3 月発表予定.

[5] http://araucaria.computing.dundee.ac.uk/