商品ルール記述の動的制約表現への変換の実装とその評価

志賀 隆之† 岩井原 瑞穂† 小澤 正幸‡

† 京都大学大学院情報学研究科 〒606-8501 京都市左京区吉田本町
‡ 凸版印刷株式会社 〒110-8560 東京都台東区台東 1-5-1
E-mail: (tshiga, kozawa)@db.soc.i.kyoto-u.ac.jp, iwaihara@i.kyoto-u.ac.jp

あらまし 商品ごとに付された多種多様な形式のルールを適用することで価格や利用条件といった商品特性が変化するような性質を持つルール付商品では，ルールは通常，自然言語で記述されている．しかし電子商取引においては，ルールを用いた商品の検索や比較，検証を行う為，当該ルールは計算機による処理が行える形式でも表現されることが望ましい．このために日本語で記述された航空券のルールを例にとり，計算機処理のための記述である動的制約表現への変換を行う．本稿では，変換の概略，特に変換の際に用いるテンプレートに関し包括型・意味型といった 2 通りの手法及びまたそれを実現するアルゴリズムを提案し，その実装・評価を行う．

キーワード E-Commerce，情報検索，データマイニング，動的制約代数，複合商品

Implementation for Converting Business-Rule to Dynamic Constraint Expression

Takayuki SHIGA† Mizuho IWAIHARA† and Masayuki KOZAWA‡

† Department of Social Informatics Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501 Japan
‡ Toppan Corporation 1-5-1 Taito, Taito-ku, Tokyo, 110-8560 Japan
E-mail: (tshiga, kozawa)@db.soc.i.kyoto-u.ac.jp, iwaihara@i.kyoto-u.ac.jp

Abstract E-commerce is becoming popular, product rules, which describe various conditions such as prices, applicability, discount conditions are still expressed in natural languages. In the area of e-commerce, however, it is desirable to provide computer-processible rules for searching, comparing and validating merchandise. It is important to support of converting rules in a natural language into constraint expressions for this purpose and realizing the laborsaving, but there are many problems such as solving the fracutuations among descriptions. In this paper, we discuss technologies for locating those rules in product descriptions, categorizing them, and translating them into a formal constraint language. We take airline tickets as an example, and discuss what processes in this translation can be automated. By translating into the constraint language, we can search, compare and verify complex conditions.

Keyword e-commerce, meta-data management, knowledge processing, knowledge discovery, data mining

1. はじめに

ネットワーク社会の発展とともに，様々な経済活動がネットワークを介して行われようとしており，電子商取引は今後様々な形態のものが発展してゆくと思われる．我々は,その中でもビジネスルールと呼ばれる，売り手および買い手に付帯する条件をそれぞれ制約として表現した記述について研究を行っている．

電子商取引の普及に伴い，従来は人手で行われていた分野の商取引活動に対しても計算機による支援が求められるようになってきている．筆者らはその中でもビジネスルールの自動処理に着目し，ルール付商品，つまり商品ごとに多種多様な形式のルールが付され，その適用によって価格などの商品属性が変化するという特徴を持つ商品の電子商取引に対して，そのルール処理の支援を行う仕組みを研究してきた[4],[5],[7],[8],[10]．実際に電子商取引市場で扱われている商品の中で，このような特徴を持つルール付商品の具体例としては，航空券やホテル，レンタカー，旅行パックや PC の BTO 販売などが挙げられる．航空券では TravelXML や Open Travel Alliance Specification

などの記述言語が提案されているが，いずれも論理的なルールの記述は自然言語により行なわれ，人間による解釈を前堤としている．このような論理的表現は，将来的には商品データを作成する段階でデータ作成者によって与えられることが考えられる．

現在，電子商取引のサイトでは，ビジネスルールをチェックするプログラムをユーザに提供することが盛んに行われている．例えば，PC ベンダーのサイトの中には，ユーザが入力した構成が組み合わせ制約を満たしているのかどうかをチェックする機能があるサイトが，航空券サイトの中には座席利用や価格のルールのような基本的な制約をチェックする機能が備わっているサイトが存在する．この組みこまれたプログラムによるルールチェックには以下のような問題がある；

(1) プログラムによって扱われるルールはシンプルなものに限られている；複雑なルールの解釈は人間に任されている．

(2) ビジネスルールの検査方法は，あらかじめ用意されたプログラムで行なわれる．そのためビジネスルールの柔軟な推論，比較，照合といった機能は提供されない．

(3) プログラムはビジネスルールの検査方法を定義しているが，オープンでないため，柔軟な商品検索のための商取引サイト間でのルールの交換が行われない．

現状では，ウェブ文書のほとんどのルールデータが自然言語で記述されており，自然言語からこのような表現への変換作業が必要となる．こうした作業の省力化は，たとえば Semantic Web[1] においても重要な問題である．DAML+OIL やその後継である OWL はウェブオントロジのための記述論理に基づいた言語である．ルールは Semantic Web の七層モデルでは第四層“Rules”に対応し，一～三層（それぞれ RDF, RDF-Schema, Ontology）で表現された「知識」に対して、論理による演繹のための記述方法を与えている．RuleML[2]は Semantic Web の一環としてルールや論理式を XML タグでマークアップする方法を標準化するためのプロジェクトである．電子商取引に関しては文献[3]や IBM Common Rules などのプロジェクトからの成果を 取り入れている．

本論文で，我々はウェブ上にある商品記述からのビジネスルールの抽出について述べる．ウェブ文書からのビジネスルール抽出では[6] が書店の送料ルールを例に Web からのルール抽出を行っているが，人手で自然言語のルールにマークアップする作業を必要としている．

さらに，本論文では航空券販売を例に挙げ，筆者らが開発を行っているビジネスルール処理のフレームワークである動的制約データベース用の表現[5]に変換する作業を支援する手法を議論する．これは他の論理的表現への変換についても，共通して適用できうる部分を有する．

本論文の構成は，まず 2.節でルール付商品とその例である航空券について述べ，3.節でルール付商品のモデル化を行う．そして，4.節では本手法を適用し，動的制約表現の記述を扱うための動的制約 DB に関する説明を行う．その後，5.6.節において自然言語で記述されたルールの動的制約表現への変換する際の注意点やその準備事項について述べ，7.節では変換テンプレートを用いた変換支援手法のアルゴリズムと実行例を示す．8.節で実装の概要と結果，その考察を行い，9.節で結論を述べる．

2. ルール付商品

現在，旅行代理店・航空会社などが運営するサイトで電子商取引として航空券が広く販売されている．その取引形態は様々であり，Web サイトのみで商品選択と購入が可能なカタログ販売型とサイト上には簡潔な情報のみを置き，詳細については電子メールなどによって顧客と旅行代理店間で交渉を行う相対取引型が一般的である．一般に格安航空券と呼ばれるものは，航空会社からまとめて仕入れた商品に，様々な条件を付けて小売りするものであり，航空会社のものより価格は安いが様々な制約条件が付帯されているものが多い．この場合は様々な割引方式や複雑な価格構成が存在し，ルールには商品ごとのばらつきが見られる．ただし，商品ルールの自然言語記述には，表現の揺らぎはあるが，文の意味自体は曖昧性を持たないという特徴がある．実際の航空券から抜き出したルールの例を表 1 に示す．

訪問可能都市	アトランタ，ボストン，バッファロー，…，シアトル，サンフランシスコ，ソルトレイクシティ，バンクーバー，トロント
訪問可能都市数	●上記指定都市内 3 都市周遊可能です．
有効期限	1 泊 3 日～1 ヶ月以内 FIX（復路変更不可）
追加料金	週末料金：現地発金・土・日は，¥4,000UP
ストップオーバー	2 回まで無料．3 回目まで可：¥10,000UP

表 1: 航空券の商品ルールの例

こういった商品ルールを本稿で扱うような論理的表現に変換することにより，以下のような計算機による有益な応用が考えられる．

- 売り手の条件と買い手側の条件をそれぞれルールで表現し，それらの論理積の充足可能性や，含意の関係を調べることにより，互いの条件が満足されているかの検査を行う．
- 商品にどのような付帯条件や制限事項があるかを検索し，表示する．
- 買い手の要求条件を，商品ルールに代入することにより，可能な選択肢を求め，それを買い手に選択メニューとして提示する．

3. 商品ルールのモデル化

3.1. 論理的表現の要求事項

ルールの表現手法に関して望ましい性質としては次のようなものが挙げられる.

- 論理的表現には曖昧性が無く，厳密な意味が与えられることが必要である.
- 計算処理が実用的時間になるように，表現力を適切なものとする.
- 論理的表現から元の自然言語の表現が直ちに求められるようにする.
- 実際には自然言語で記述されているルールが表す概念の全てが論理的表現に変換できるとは限らないため，部分的に自然言語表現を混在させることが必要である．自然言語による表現の部分は，利用者に表示して真偽値の判断を求めるようにする.

3.2. 商品ルールの論理的表現のモデル

定義 1

1. 商品ルールは商品の各種特性を表す商品属性とその値を表す商品属性値からなる制約を表わす論理式である.
2. 1 つの商品ルールは，いくつかのルール条項に分割することができ，各商品ルールはルール条項の論理積である.
3. 各商品属性には値域集合が割り当てられている．これはたとえば日付や金額，地名等である.
4. 商品ルールあるいはルール条項の自然言語による表現をそれぞれ商品ルール文およびルール条項文と呼ぶ.
5. 商品ルールおよびルール条項を表わす論理式のクラスとして，等号論理を用いる．これは，変数と定数，または変数同志の等号を *AND*,*OR*,*NOT* のブール結合子で結合した式であり，例えば $(x="abc" \vee y \neq "def") \wedge \neg(z="ghi")$のような論理式である．各変数はいずれかの商品属性に対応付けられ，定数は商品属性の値域集合に含まれるものとする.

上記の等号論理式の制約集合に対し，制約データベース[7]の考え方により代数演算を行う質問言語として，動的制約代数[4][5] があり，商品ルールの集合的操作および論理演算に用いることができる.

4. 動的制約データベース

動的制約データベース(DCDB: Dynamic Constraint Database)は電子商取引において商品ごとに異なる論理的条件を持つような問題を扱うために開発されたデータベースモデルである．制約データベースとは，変数とそれに与えられた制約条件によってデータを保持するデータベースであり，制約を充足する変数のインスタンスすべてが制約データベースの蓄える具体的な値と見なされる．動的制約データベースとは変数のスキーマ定義を必要としないように制約データベースを拡張したものである.従来の RDB による電子商取引サイトでは，複雑な条件はプログラムとして表現され，あらかじめ規定した方法でのみしか条件の指定ができなかった．DCDB ではこのような外部プログラムで表現されていたような複雑なロジックをプログラムから分離し，商品データの一部（制約データベースの術語では制約）として表現することができる．このことにより，多様な売買条件やルールを商品ごとに柔軟に設定することができ，また質問言語である動的制約代数を用いて，照合操作や充足性の判定など制約による柔軟な検索や演算を行うことが可能となる．DCDB のテーブルの例を表 2 に示す．現在の実装では制約として和積形の等号論理式を用いることができる．これは等式あるいはその否定を論理和 OR で結んだ和項をさらに論理積 AND で結んだものである.

ID	料金	目的地	制約
1	p_1	v_1	v_1={ロンドン，パリ}
2	p_2	v_2	v_2={ロンドン，ローマ}

表 2:制約関係テーブル

5. ルール条項文の変換処理

5.1. 変換元の航空券データ

変換元のデータとして，旅行代理店による航空券販売サイトであるアルキカタ・ドット・コム(http://www.arukikata.com/) から 3 ヶ月間 2000 ページ分の航空券データを収集した．データには商品コードや訪問可能都市，出発地，出発日と基本料金の対応表，追加料金ルール,その他のコメントなど合わせて計 17 の項目分けがなされ，日本語で記述されている．上記の日本語の記述は商品ルール文に対応することになる．商品ルール文をさらにルール条項文に分割してゆく必要があるが，意味的にまとまったものを分割するために，改行文字や箇条書きの区切りなどを手がかりとし，スクリプト処理を行った[8]．この分割では他を参照しなければ意味を把握できないルール条項文を作らないように，分割の基準を緩やかに取った．この作業により，収集した 2016 件の商品データに付随する商品ルールを分割したルール条項の数は 18482 件となった．さらに文字列の比較により重複したものを削除すると 3790 種類である.

他サイトからのデータ収集に関して，格安航空券サイト，トラベルコちゃんで上記のアルキカタ同様，1500 件のデータを収集し，その後さらに 3500 件，計 5000 件のデータを収集した．1500 件のページの場合，商品ルールを分割し，重複したものを削除すると，2712 種類のルールが収集でき，5000 件のページの場合，同様の操作を行うと，9669 種類のルールが収集できた．その他， HIS (http://www.his-j.com/tyo/air/index.htm) や JTB (http://www.jtb.co.jp/kaigai/theme/bestprice/), 格安航空券の老舗（http://www.jp-tour.com/）など，他の

格安航空券を販売しているサイトに関しても調査を行ったが，基本的にフォーマットさえ確立されているサイトであれば，データを抽出できれば本手法の適用は可能なように設計している．

5.2. 類似ルールの例

商品ルールの中には，他と同一か類似するルール条項文は多く,また日本語の表記の揺らぎがあるものや，論理的な構造は同じでも，具体的な値やその個数が異なるものがある．収集したルールのうち，週末料金に関するルール条項文の中から，このような表記の揺らぎや具体的な値，そして値の個数の違いを含む例を表3に示す．

1	*カイロ-ソウル便は現地発木・土曜日の運行です．　行き・帰りとも同日乗り継ぎが可能です．
2	*カイロ―ソウル便は現地発木・土曜日運行．往復共に同日乗り継ぎ可能．
3	●カイロ―ソウル便は現地発木・土曜日運行．往復共に同日乗り継ぎ可能．
4	*ジェッダ―シンガポール間　日・水・木曜日運行（現地発）
5	●ジェッダ―シンガポール間　日水木運行（現地発）**スケジュールは変更になる場合がございます．
6	*シンガポール―ジェッダ間　日・水・木曜日運行（同日乗り継ぎはできません．）
7	*台北-アムステルダム　月・火・木・土運行
8	*台北-ウィーン　水・木・金・日運行
9	*ウィーン-台北　月・木・金・日運行
10	*往路、ソウル―フランクフルト間は日・水・金曜日の運行です．

表3：表記の揺らぎなどを含むルール条項文

- 行頭の“*”や“●”は箇条書きの先頭記号に由来する．統一されたフォーマットが存在するわけではなく，ルール条項文の記述者によると見られる不揃いが存在する．
- ルール1,2,3は表記の違いだけで,内容としては「カイロを出発し，ソウルに到着する便はカイロ発で木曜日・土曜日に運行している」という同一のものである.ルール6と7でも同様のことが言える．

6. 商品ルールの論理的表現の抽出支援手法

6.1. 登録単語辞書

商品ルール文を解析し，論理的表現に変換するために，商品ルール文の記述に使用されている用語が登録されている登録単語辞書を用意する．登録単語のカテゴリーを表す値域集合の名前を単語型あるいは単に型と呼ぶ.例えば「都市名」や「日時」，「曜日」，「金額」，「否定語」が単語型である．登録単語辞書では，登録語から単語型を引けるようにする．また，商品ルール文は人手で書かれたものが多いため表記の揺らぎが存在する．例えば，土曜日を表す語としては，“土”，“土曜”，“土曜日”などがあり，このうちの1つを標準形として辞書に登録する．

6.2. 論理的表現の抽出支援手法の流れ

1. 商品ルール文について，登録単語辞書を検索して標準形の登録語(トークン)に置換する.この過程をトークン抽出と呼ぶ．トークン抽出により，表記の揺らぎが吸収される．
2. 航空券においては，訪問可能都市や可能な日時・曜日の列挙が見られるが，これらは同じ型のトークンの列として検出可能である．トークンの反復畳み込みとして，このようなトークンの列を検出する.これにより,トークン列の繰り返し回数や，カンマ等の句読点を無視した，同じ型のトークン列を認識する．
3. ルールのクラスタリングとして，トークンの反復畳み込みの結果が同一になったルール条項をまとめ込む．
4. 変換テンプレートの適用として，上記の結果のルール条項文にマッチする変換テンプレートを検索し，そのテンプレートに従ってルール条項文を論理的表現に変換する．

上記の手法は，自然言語という曖昧な言語を処理するという特性上，完全に自動化できるものではなく，変換に誤りが含まれることがある．そのため，各段階で人手によって変換結果の確認を行う．最終的な判断を人間が行うという意味で，本手法は抽出支援手法という位置づけになる．数千を越える商品ルールを全て人手で行うのは困難であり，本抽出支援手法による半自動化の効果は大きいと考えられる．

6.3. 抽出支援手法の実行例

6.3.1. ルールからのトークン抽出

ルールからトークン抽出を行う際，揺らぎのある表記は標準形であるトークンに変換される．表2のルール1,10に対しトークン抽出を行った結果を，表4に示す．

	(#型, 標準形,“ルール中の表記”)
1	(#都市, カイロ, “カイロ”) (#都市, ソウル, “ソウル”) (#曜日, 木曜日, “木”) (#曜日, 土曜日, “土曜日”)
10	(#往路, 往路, “往路、”) (#都市, ソウル, “ソウル”) (#都市, フランクフルト, “フランクフルト”) (#曜日, 日曜日, “日”) (#曜日, 水曜日, “水”) (#曜日, 金曜日, “金曜日”)

表4：トークン抽出を行ったルール条項文

型名の先頭には#を，標準形には下線を付し，ルール文中に現れた表記については，二重引用符でかこった．これらについて型と標準形のみで見た場合,ルール1,2,3,そして前述の通りこれと同様に，同じ意味内容を持つ

ルールは，全て一致し，表記の揺らぎを吸収できたことが分かる．

6.3.2. トークンの反復畳み込みとクラスタリング

トークンの反復畳み込みについて述べる． トークン列 $S = s_1 s_2 ... s_l$ について，各トークン s_i の型 t_i を調べ，型の部分列 $t_j, t_{j+1}, ..., t_{j+k}$ が 1 回以上の繰り返しになっている箇所が見つかれば，それを型の列の繰り返しを表わすトークン $(t_j, t_{j+1}, ..., t_{j+k})^+$ で置換する．この操作を S の先頭から順次行い，適用できなくなるまで変換する．畳み込まれたトークン $(t_j, t_{j+1}, ..., t_{j+k})^+$ を縮退トークンと呼ぶ．縮退トークン列の集合について，縮対トークン列の型が一致するものをまとめ込む操作をルールのクラスタリングと呼ぶ．

表 4 の例に対して，反復畳み込みを行った結果を表 5 に示す．ルール 1 は *a*，ルール 13 は *b* で表される列型を持つ縮退トークン列に畳み込まれる．この他にも，表 3 で示した例のいくつかがこれら *a, b* いずれかの列型を持つ縮退トークン列に畳み込まれる．

	列型
a	“カイロ”[1] “ソウル”[2] $(_1$曜日[3]$)_1{}^+$
b	“往路”[1]; “ソウル”[2] “フランクフルト”[3] $(_1$曜日[4]$)_1{}^+$

表 5：まとめ込まれたトークン列の型

7. 変換テンプレートの適用

7.1. 変換テンプレートの概要

得られた縮退トークン列の集合に対し，縮退トークン列ごとに論理的表現への変換を与える変換テンプレートを用意する．変換テンプレートでは，縮退トークンの型および定数に応じて，変数および定数を割り当て，さらに縮退トークンの括弧の入れ子構造に対応して，論理演算を割り当ててゆく情報を持たせる．

縮退トークン列 τ の各部分列に変換テンプレートの対応する変換を適用するために，τ の開き括弧と閉じ括弧に同じ番号からなる識別子を与え，また τ に f 含まれる各トークン型にも先頭から番号の識別子を与える．表 5 には，型に番号付けを行っており，トークン型の番号は上付で,括弧の番号は下付で示されている．

次に論理的表現への変換する際の“～は不可”といった否定を含む表現について考える．すべての否定演算子“¬”はド・モルガンの法則によって，$v = "c"$ または $v \neq "c"$ といったリテラルに押し下げることができる．これにより，商品ルール文の大部分は，変換結果として AND($\wedge$)と OR($\vee$)，NOR($\oplus$)，または Sequence(σ)で表記できる．そして，括弧に対応する部分列に対する変換規則について考える．各関数の計算は文字列操作などであり，変換スクリプトとして定義される．変換関数の定義において，“.”は文字列連接の演算子であり，$std(c)$ は c に対するトークンの標準形を与える．例えば，型が都市であるトークン列 $c_1, c_2, ..., c_k$ は $(_i w_j)_i^+$ と表現される．$\tau((.)_i) = \vee$，$\tau(w_j) = city1$ とすると，ルール文は次のように変換される．

$(city1 = std(c_1) \vee city1 = std(c_2) \vee \cdots \vee city1 = std(c_k))$

また，論理演算子として $\tau((.)_i)$ に NOR($\oplus$)を用いると，このルールは次のように表せる．

$(city1 \neq std(c_1) \wedge city1 \neq std(c_2) \wedge \cdots \wedge city1 \neq std(c_k))$

動的制約代数はこのような変数と定数による等式とその否定を OR で結んだ和項,あるいはその否定を AND で結んだ積項を扱えることが特徴である．また，トークン列が NAND である場合は考える必要はない．なぜならば，これは $k \geq 2$ である場合，$(city1 \neq std(c_1) \vee city1 \neq std(c_2) \vee \cdots \vee city1 \neq std(c_k))$ という式はすべて真になるからである．同様にして，1 変数の AND のトークン列として $(city1 = std(c_1) \wedge city1 = std(c_2) \wedge \cdots \wedge city1 = std(c_k))$ を考えても $k \geq 2$ のときすべて偽となる．ゆえに，AND は異なる変数における選言や等号を結合するという目的で使われるのが適している．さらに，すべてのトークン列が AND, OR や NOR という論理結合子で変換されるわけではないということに注意しなければならない．これは，トークン列が論理結合ではなく序列を表す場合が例として挙げられる.こういった場合には SEQUENCE を用い，変換を臨機応変に行う．

変換形式としては以下の 2 つの変換形式を考えた．

意味型変換テンプレート(Semantic Transformation)

意味型テンプレートでは,テンプレート τ_s がフォーマット $< t_1, t_2, ..., t_k >$ を備えている（各々の t_i は文字列もしくは畳み込まれたトークン列）．T が正確に文字列 $t_1, t_2, ..., t_k$ とマッチすれば，そのテンプレートが畳み込まれたトークンに適用される．テンプレート τ_s は畳み込まれたトークン列 t_i に論理演算子である(AND，OR，NOR)や SEQUENCE，または変数名を割り当てる機能を有している．

包括型変換テンプレート(Generic Transformation)

包括型変換テンプレート τ_{gen} は，登場したすべての文字列に変数と定数を自動的に割り当てるというものである．変換の手順は以下の通りである．

もし，畳み込まれたトークン列が，論理演算子を指し示す特定の語句を含んでいるのであれば，論理演算子に従ってトークン列の変換を行う．典型的な例として，「$c_1, c_2, ..., c_k$ は除く」といった表現があれば NOR に変換するという規則が挙げられる．

もし、トークン列に一致する語句がなければ、その列は SEQUENCE と見なされ，和積形の等号論理式に変換される．各々の等号は異なった変数名を持つ．例えば，列 $c_1, c_2, ..., c_k$ が id i と，語句の型 wt を持つことを考える．すると，変換は以下のようにして行われる：

$i\_wt\_1 = std(c_1) \wedge i\_wt\_2 = std(c_2) \wedge \cdots \wedge i\_wt\_k = std(c_k)$

包括型変換テンプレートの考え方は，各々のルール文は等号 $i\_wt\_i = std(t)$ の連言である 1 つの“タプル”に変換されるというものである．t は語句の型 wt に対して登録されたトークンである．論理演算子（AND，OR，NOR）が文脈から明らかなトークン列に関しては，そのまま変換が行われる．

意味型変換テンプレートは様々な文字列パターンに対し，正確に変換を行うことができるが，テンプレートを一つ一つ用意しなければならず，コストがかかるという欠点がある．一方，包括型変換テンプレートでは各々の文字列の意味に応じたテンプレートを適用する必要がなく,コストが少ないという長所がある反面，逆に，ルール文のすべての意味を論理式に反映できないという短所もある．結局，理想的な変換テンプレー

トとは，各々のルールに対して，必要に応じてユーザと対話しながら精度を向上できるものであると考えられる．

次に，変換テンプレートの定義を行う．列型 τ を持つ縮退トークン列集合 C_τ への変換テンプレート MC_τ とは，以下の規則に従う，τ の各構成要素への，変換関数 f，論理演算子 op，空文字 ε のいずれかの割り当てである．構成要素 a に A を割り当てることを $MC_\tau : a \mapsto A$ と書く．

7.2. 変換の際の変数設定方法

変換テンプレートにおける変換の際，意味型変換テンプレートにおいては，変数の意味を正確に反映できるように変数の付け方に関しても，意味によって異なった変数を付けるように考慮した．例えば，都市を表す“City”という変数でも，出発都市と到着都市では意味が異なってくるため，出発都市は“City_Dep”，到着都市は“City_Arr”として表現を行った．またこの変数の設定方法に関しては，さらなる自動化の手法が考えられる．この手法としては，①現在上記の変数設定は手動で行っているが，これを自動化する，②包括型の変数設定に関して，現在は変数＋数字という形でのみ行っているが，これを意味が同じか，異なる用法かを判別する，といったものである．①に関しては，予め主となる変数である型名とその意味を表す変数(上記で表すと DEP，BEG，PER など)を用意しておき，文章解析により自動的に変数を割り当てていく方法である．この方法が実現できれば②に関する拡張も可能になると考えられる．文章解析を行うことによって，意味が同じか,異なる用法かを判別し,それによって City_1_A など変数＋数字＋変数という形で表す．最後の変数が一致していれば同じ意味，一致していなければ異なる意味だと判別できるようになる．

7.3. 変換テンプレートを用いた変換例

7.3.1. 包括型変換テンプレートによる変換例

包括型変換テンプレートを適用した結果を表 6 に示す．表 6 におけるルールは 4 節で紹介した動的制約データベースで扱える動的制約表現の形式で記述している．ルールの ID や用いている型のタイプ，元の自然言語文とそれを動的制約表現に変換した形式がタグ<cond></cond>の中身に記述されている． 包括型変換テンプレートにおいては，“登場した文字列の型名＋登場順”で変数名を構成している．この例の場合，曜日や日時が連続しているので choice を用いて実現し，その他は eq を用いた和積形の形で表現している．

```
<tuple>
<value num="0" atr="rule_id">1304</value>
<value num="1" atr="cluster_id">[( ( City ) ( DOW ) )]</value>
<value num="2" atr="hash_key">CITY DOW</value>
<value num="3" atr="sentence">1304*往路・復路共に、ソウル―フランクフルト間は日・月・水・金曜日、ソウル-ロンドン間は火・木・土曜日の運行です。</value>
<cond>
        choice(DOW_1, "日", "月", "水", "金");
        choice(DOW_2, "火", "木", "土");
        eq(CITY_3, "ソウル");
        eq(CITY_4, "フランクフルト");
        eq(CITY_5, "ソウル");
        eq(CITY_6, "ロンドン")
</cond>
</tuple>
```

表 6：包括型変換テンプレートの適用例

7.3.2. 意味型変換テンプレートによる変換例

本節では，意味型変換テンプレート(Semantic Transformation)を用いた変換について述べる．クラスタリングされたデータを入力とし，動的制約表現を出力とするシステム上でテンプレートを適用する方法としては， 入力されたデータをインスタンスとし、その型でマッチングを行い，該当するテンプレートを適用するという形をとっているが，その中でも以下の 2 通りの方法が考えられる．

(1) インスタンスの畳み込みを行なった後，マッチング
  (ア) クラスタリングされた XML データを①インスタンス形式と②型形式に変換．括弧の箇所にタグ<sequence>を適用．
  (イ) 別にセットで用意してある、(②型＋③変換テンプレート）の組の型とマッチングを行う．
  (ウ) ②の型が一致したときに③のルールに従って①インスタンスの適当な場所に演算子を入れ、論理式を作成．最後に値を代入し、完成させる．
(2) テンプレートの正規表現にしたがって，インスタンスを読み込んでそのインスタンスが受理されるか(正規表現にマッチするか）を判定する．

型の例：

```
<type>( City ) ( Date ( DOW ) )</type>
<sequence id="1">
<token type="City”/>
</sequence><sequence id=“2”>
・・・
```

テンプレートの例：

```
<sequence id="1" type="City" op="OR"></sequence>
<op="AND">
<sequence id="2" type="Date (DOW)" op="OR">
・・・
```

(1)の方法による実装においては，アルゴリズムの入力を XML データとし，DOM (Document object model) という木構造に変換しておいて，木構造への操作として実装する Perl のモジュール XML::DOM を用いて実装，もしくは XML::SAX による行抽出＋Perl byacc によるいう方法が考えられるが，トークンの繰り返し処理などがかなり複雑化する．さらに，今回の例である航空券の例で言えば正規表現（正規文法）しか用いておらず，正規表現でトークンの繰り返しを表現でき，マッチングも行えるため，(2)の方法を採用することにした．トークン列から型名とキーワードを抽出した列を，Perl の正規表現で書いておいたテンプレートにマッチするかを判定し，マッチした部分にどの論理演算を割り当てるかを指定するというものである．よってここで次に，クラスタリング化されたデータを

動的制約表現に変換するアルゴリズムを前述の(2)の方法を用いて実装する手順を説明する.

(ⅰ) インスタンスの生成

まず，入力されたデータから，意味のあるトークンに関しては決まった形に変換する.

[((City) (DOW))]
1304　*往路・復路共に、ソウル―フランクフルト間は日・月・水・金曜日、ソウル-ロンドン間は火・木・土曜日の運行です.
[((City) (DOW))]
1304　*往路・復路共に、<CITY, ソウル>―<CITY, フランクフルト>間は<DOW, 日>・<DOW, 月>・<DOW, 水>・<DOW, 金>曜日、<CITY, ソウル>-<CITY, ロンドン>間は<DOW, 火>・<DOW, 木>・<DOW, 土>曜日の運行です.

(ⅱ) 正規表現でのマッチング

変換されたデータに対して，正規表現でマッチングを行う.

```
if ($template =~ /<CITY, (¥S+)>(¥-|―|¥/)<CITY, (¥S+)>(　| 間 は |¥-)<DOW, (¥S+)>(( ・ <DOW, (¥S+)>)+)(運行　+|、|曜日、|)<CITY, (¥S+)>(¥-|―|¥/)<CITY, (¥S+)>(　| 間 は |¥-)<DOW, (¥S+)>(( ・ <DOW, (¥S+)>)+)/) {
    連続した DOW に関しては論理演算子 OR
    CITY と(DOW)に関しては論理演算子 AND
  を割り当てる
  }
```

これらの変換アルゴリズムを用いて得られる結果を表 7 に示す.

```
<tuple>
<value num="0" atr="rule_id">1304</value>
<value num="1" atr="cluster_id">[( ( City ) ( DOW ) )]</value>
<value num="2" atr="hash_key">CITY DOW</value>
<value num="3" atr="sentence">1304　*往路・復路共に、ソウル―フランクフルト間は日・月・水・金曜日、ソウル-ロンドン間は火・木・土曜日の運行です.</value>
<cond>
  eq(CITY_DEP, "ソウル");
  eq(CITY_ARR, "フランクフルト");
  choice(DOW, "日", "月", "水", "金");
  eq(CITY_DEP, "ソウル");
  eq(CITY_ARR, "ロンドン");
  choice(DOW, "火", "木", "土")
</cond>
</tuple>
```

表 7：意味型変換テンプレートの適用例

これらの例を見てもわかるように,意味型では,個々のルールの意味を解釈して論理式化しているため，包括型と意味型を比べたときに変数名や AND，OR の処理面で，意味型のほうがより緻密に論理式として表せることがわかる.

8. 実装と評価

8.1. 結果

5.節，6.節で述べた手法に基づいて，辞書を元にルール条項文をトークン抽出し，反復畳み込みとクラスタリングを行い，その後，変換テンプレートを適用するというプログラムを Perl で実装した．実験に用いている環境であるが，CPU：Pentium4・3.20GHz，メモリ 768MB のマシンで OS は WindowsXP, CYGWIN 上でプログラムの実行を行っている.

	Parsing	Reducing	Grouping	Total
計算時間	4808	4	4	4816

表 8：クラスタリングにかかった計算時間（単位：秒）（アルキカタ・ドット・コム）

	Generic	Semantic
全体の計算時間	191	234
ルール 1 つあたりに要した計算時間	0.050	0.090

表 9：変換テンプレートの適用に要した計算時間（単位：秒）（アルキカタ・ドット・コム）

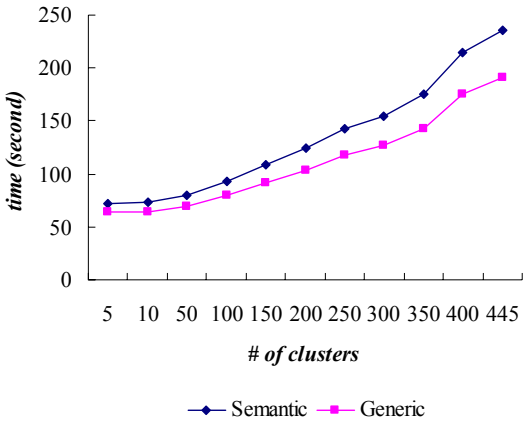

図 1：意味型＆包括型変換テンプレートの実行時間（アルキカタ・ドット・コム）

変換テンプレートに関しては，前述の包括型と意味型の 2 通りの手法を適用した．アルキカタの処理に関しては,前述の航空券データより抽出した 3790 件のルール条項文について,クラスタリングの結果 295 種(括弧付けのみが異なる物を含めると 445 種）のパターンに類別することができた．包括型変換テンプレートを適用することで，これらをすべて処理することができた．意味型変換テンプレートでは，3790 件のルール中 2602 件まで処理することを可能にした．同様に，トラベルコちゃんの処理に関しては 1500 件のページ収集を行った場合には 2712 件のルールからクラスタリングの結果，121 種（括弧付けのみが異なる物を含めると 174 種）のパターンに類別することができ，こちらも包括型変換テンプレートにおいては全て処理することができた.意味型変換テンプレートでは 2710 件のルール中 1911 件の処理を可能とした．双方とも，処理できなかった残りのルールに関しては，5 個以内のルールに関して逐一テンプレートを用意しなければならなかったため，時間がかかるという理由で残りの処理に関して

テンプレートの作成を中断することにした．

表 8 はクラスタリングにかかった計算時間，表 9，図 1 は各々の変換テンプレートの適用にかかった計算時間を示している．

8.2. 評価

クラスタリングまでの処理に関して，単語の型として曜日，価格，日付，都市名など 11 種類の辞書データを用意し，それぞれに属する語をルール文から抜き出し登録することで辞書を作成した．トークン抽出の精度は辞書の充実度に依存しており，辞書が完全でない段階では，変換誤りが見られたが，辞書に用語を追加することによりトークン抽出およびクラスタリングの精度の向上が見られた．具体的には，以前，クラスタリングによって類別された種類が 500 近くあったものを 450 以下に減らすことが可能になった．トークン抽出に関しては，現在，ほぼ確実な抽出が行える．クラスタリングは Parsing，Reducing，Grouping という 3 要素からなっているが，ほとんどが Parsing の時間に費やされている．実際，処理するルール数に比例して Parsing の時間も増えていっているが，実用の際には，何度も実行する類のものではなく，本システムに付加されているキャッシュを用いてこの問題を解決することができる．

変換テンプレートの処理では，前章で述べたように意味型変換テンプレートの方が包括型変換テンプレートと比べて，変数の明確さや論理演算子の明確化によって，より緻密にルールを論理式化できる反面，表 12 が示すように，計算時間を始め，最初にテンプレートを明確に記述しなければならないのでその時間がかかるという欠点があることが示された．つまり，緻密性に関しては，テンプレートの設定次第でコントロール可能であるが，テンプレートの記述時間や実行時間とのトレードオフな関係であるといえる．

変換の際に，意味型変換テンプレートの作成が難しいケースも存在した．これは，①最初のページに打ち込みミスが存在する場合，②全角数字が使用されている場合，③元の自然言語が複雑な場合などである．①に関しては，テンプレート作成の段階で，そういった間違いに対応した正規表現を記述することで対応し，②に関してはデータベースに格納する段階で，予め全角の数字は半角に変換しておくことで対応した．③の変換に関しては今後の課題である．また，結果が示すように包括型変換テンプレートでは全ルールの変換を可能とし，意味型変換テンプレートでは[23]，[24]双方のサイトにおいて，ルール全体の 70%以上の変換を可能とした．逆に，言い換えれば，全てのルールを処理することまではできなかった．これは，前述のように残りのルールに関しては，1 個，ないしは 2 個，3 個といった少数のルールに対して，テンプレートを逐一用意しなければならず，効率としては良くないからである．これを防ぐためには，残りのルールに関しては包括型変換テンプレートを適用することや，ルールに関して逐一論理演算を割り当てる，対話型のシステムを構築するということも考えられる．このことからわかるように，現在のシステムでは変換した結果の誤りを人間が修正する形式を取っているが，登録単語辞書や変換テンプレートの修正過程で適宜入力して精度向上を図れるような対話型システムも今後の課題である．

9. まとめ

本論文では，Web 上に自然言語で記述されたルール付商品の商品ルールを論理的表現に変換するために，トークン抽出や変換テンプレートを用いた変換支援手法を提案し，それを実装することで評価することに成功した．今後は，さらなる自動化のための手法の改良や計算時間の短縮，さらには前述した対話型システムの実現を目指す予定である．

文　献

[1] Berners-Lee, T.: The Semantic Web-LCS seminar, http://www.w3.org/2002/Talks/09-lcs-sweb-tbl/

[2] Boley, H.: The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL transformations, INAP2001, Tokyo, Japan, in October 2001(2001).

[3] Grosof, B. N. , Labrou, Y. and Chan, H. Y.: A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML, E-COMMERCE 99, Denver, Colorado(1999).

[4] Iwaihara, M.: Supporting Dynamic Constraints for Commerce Negotiations, 2nd Int. Workshop in Advanced Issues of E-commerce and Web-Information Systems (WECWIS), IEEE Press, pp.12-20, June (2000).

[5] Iwaihara, M.: Matching and Deriving Dynamic Constraints for E-Commerce Negotiations, Workshop on Technologies for EServices, (informal proceedings), Cairo, Sep. (2000).

[6] Kang, Juyoung and Lee, Jae Kyu: Extraction of Structured Rules from Web Pages and Maintenance of Mutual Consistency: XRML Approach, Proc. of RuleML 2003, Springer-Verlag, LNCS2876, pp. 150-163, (2003).

[7] Kozawa, M. , Iwaihara, M. and Kambayashi, Y.: Constraint Search for Comparing Multiple- Incentive Merchandises, Proc. of EC-Web 2002, Springer, LNCS2455, pp.152-161, Sep. (2002)

[8] 志賀隆之，岩井原瑞穂，小澤正幸: 電子商取引における商品ルールの動的制約表現への変換手法の実装，夏のデータベースワークショップ (DBWS2004), Jul. 2004.

[9] Kuper, G. , Libkin L. and Paredaens, J. (Eds.): Constraint Databases, Springer-Verlog, (2000).

[10] Iwaihara. M, Shiga. T and Kozawa. M, "Extracting Business Rules from Web Product Descriptions," Proc. 5th Int. Conf. on Web Information Systems Engineering (WISE2004), LNCS3306, pp. 135-146, Nov. 2004.