B-04

帰納論理プログラミングによる代用のための知識の獲得

Knowledge Acquisition for Substitution by Inductive Logic Programming

古橋 翔吾†　　岡 夏樹†
Shogo Furuhashi　　Natsuki Oka

1. はじめに

人間は日常生活の中で代用を行う．ここでいう代用とは，あるものに代えて本来であれば他の用途に用いられる物を利用することをいう．具体的には，コップという飲みものを入れて飲むための物を紙の上に置くことで，紙が風で飛ぶことのないように重しとして用いるといった例が挙げられる．このような代用を人間は状況に応じて必要かを判断し，必要となれば瞬時に適切な代用物を身の回りから見つけ出し用いることができる．しかし，実際に代用を行うための知能を考えると複雑な処理が必要である．まず周囲に解決すべき問題があることを認識し，その問題解決のために代用を行う必要があるかを判断できる必要がある．次に代用を行うために必要な特徴が何かということを理解する必要がある．先に上げた重しの例であれば，転がらないよう底面が平らな物で，紙の上に載せて風で飛ばない程度の重さで，かつ，紙の上に持っていける程度の重さである必要がある．多くの場合は，紙を汚さないものであることも必要だろう．また周囲にある物の中から必要な特徴を持つ適切な物を見つけ出し使用する必要がある．使用する物がその時点で果たしている役割や将来果たすであろう役割も考慮する必要がある．今何かの重しとしての役割を果たしている物を他の物の重しに転用するのは通常は不適切である．また，急な降雨で，自分の傘の代わりに，これから使われるであろう他人の傘を拝借するのも，多くの場合，不適切である．一連の判断や処理は代用を行う上でどれも重要なものであるが，本論文では，代用に必要な物体の特徴を理解する部分に焦点を当てる．

先行研究では複数のモダリティを用いることにより，より人間の感覚に近いカテゴリを形成することが可能であることが示されている[1]．しかし，代用のために必要になるカテゴリは，このような汎用的，永続的で文脈に依存しないカテゴリではなく，アドホック・カテゴリと呼ばれる，判断主体と環境の相互作用を通じて，目標指向的，一時的に形成されるカテゴリである．

そこで本論文では，まず，エージェントに，一つ一つの物体の持つ形状・大きさ・硬さのような特徴を学習させ，つづいて，エージェントが環境内で試行錯誤的に代用を試みることを通して，特定の代用が成立するために必要な，物体が持つ特徴を学習させる（我々は，これが，アドホック・カテゴリの形成に対応すると考える）ことを試みる．

2．帰納論理プログラミング

帰納論理プログラミング(ILP)とは，一階述語論理に基づいた機械学習の一手法である．ILP は，正例(E^+)，負例(E^-)，および背景知識(β)が与えられ，それらが条件

事前不十分性：$\beta \not\models E^+$

事前無矛盾性：$\beta \cup E^- \not\models \square$

を満たすとき，

事後十分性：$\beta \cup H \models E^+$

事後無矛盾性：$\beta \cup H \cup E^- \not\models \square$

を満足する仮説Hを求めるものである[2]．つまり，ILP の目的は与えられた背景知識と正負例から論理的な規則を求めることである．本研究では，代表的な ILP の 1 つである Progol[3]を用いる．

3．代用のための知識の獲得

本節では，エージェントが積木の世界で代用を試す環境を作成し，その環境に対する入出力からエージェントが代用のための知識を獲得可能であることを示す．

3.1　積木の世界で代用を可能とする環境の仕様

最終的には，ロボットが実世界で動作する環境を用意し，そこで代用のための知識を獲得することを目標としているが，今回は手始めに，論理型プログラミング言語 Prolog を用いて作成した簡単な環境を用意した．今回環境で用いた述語とその意味は表 1 に示す通りである．

述語	意味
object(A)	記号 A は物体である
name_(A,Name)	記号 A の名称は Name である
shape(A,Shape)	記号 A の形状は Shape である
top(A,Dir,X)	記号 A が Dir の向きなら上部の状態は X
put(A,B,Dir)	Dir の向きに置いてある記号 B の上部に A を載せる
read_(A)	記号 A を読む
roll(A)	記号 A を転がす
put_a_pen_into (A)	記号 A にペンを立てる
drink(A)	記号 A を飲む

表 1 環境に使用した述語とその意味

また，環境中には記号 a から記号 h で示す 8 つの物体があり，それぞれ表 2 に示す属性を持っている．

記号 object	名前 name	形状 shape	縦置き時の上部の状態 top(vertical)	横置き時の上部の状態 top(horizontal)
a	block	cuboid	flat	flat
b	block	cuboid	flat	flat
c	book	cuboid	flat	flat
d	ball	sphere	unflat	unflat
e	pen_ stand	triangular _pole	flat	unflat
f	can	cylinder	flat	unflat
g	dice	cuboid	flat	flat
h	medicine bottle	cylinder	flat	unflat

表 2 物体と物体の持つ属性の対応表

†京都工芸繊維大学，Kyoto Institute of Technology

表 1 の述語 read_(A)は A=c のとき，roll(A)は A=d, g のとき，put_a_pen_into(A)は A=e のとき，drink(A)は A=f, h のとき，それぞれ true となりそれ以外の記号が入ると false を返す．これは各記号が指す物体に対して本来行う動作に対応させた仕様である．また，put(A,B,Dir)は top(B,Dir,flat)が true であれば true を返す仕様である．これは下にある物体 B の上部が平らであれば上に A を積むことができるという意味である．A, B が block(積木)であれば，もちろん積むことができるが，積木でなくても B の上部が平らであれば積むことができる仕様とすることにより，積木の代用が可能である環境とした．

3.2 学習エージェントの仕様

エージェントには論理型プログラミング言語 Prolog と帰納論理プログラミングシステム Progol を用いた．これによりエージェントの持つ背景知識と正負の事例を基に帰納的に推論することが可能になる．

エージェントは環境に対して物体の属性を問い合わせることができる．例えば，name_(a,X)と名前を問い合わせると，X=block のように応答が返ってくる．形状の問い合わせ shape(b,X)や上部の状態の問い合わせ top(a,vertical,X)等も同様である．環境に対するこれらの問い合わせとそれへの応答を通じて，エージェントの内部には name_(a,block), shape(b,cuboid)といった物体が持つ属性が記述されていき，結果的に環境内部の物体と物体の持つ属性情報と同じものがエージェント内部に記述されていく．これをエージェントの背景知識とする．

また，動作を表す述語 put, read_, roll, put_a_pen_into, drink については，それらを用いた環境への問い合わせ（例：put(a,b,vertical)への応答が true であった場合は正例，false であった場合は負例としてエージェント内部に記述していく．以上の背景知識，正例，負例により，ある動作に必要な属性が何かを帰納論理プログラミングによって導き出す．

3.3 動作確認

まず積木以外である物体 c(本)での put 述語を用いた問い合わせに対して次のように true が返ってきたとする．

$$\mathrm{put}(a, c, \mathrm{horizontal}) \rightarrow \mathrm{true}$$

次に積木同士を組み合わせた次の問い合わせに対して，以下のように応答が返ってきたとする．

$$\mathrm{put}(a, b, \mathrm{vertical}) \rightarrow \mathrm{true}$$
$$\mathrm{put}(b, a, \mathrm{vertical}) \rightarrow \mathrm{true}$$
$$\mathrm{put}(a, b, \mathrm{horizontal}) \rightarrow \mathrm{true}$$
$$\mathrm{put}(b, a, \mathrm{horizontal}) \rightarrow \mathrm{true}$$

そして次のように，ボールの上に積木を置く問い合わせに対して false が返ってきたとする．

$$\mathrm{put}(a, d, \mathrm{vertical}) \rightarrow \mathrm{false}$$

ここでエージェント内の事例の記述は以下のようになっている．

正例： $\mathrm{put}(a, b, \mathrm{vertical})$. $\mathrm{put}(b, a, \mathrm{vertical})$.
$\mathrm{put}(a, b, \mathrm{horizontal})$. $\mathrm{put}(b, a, \mathrm{horizontal})$.
$\mathrm{put}(a, c, \mathrm{horizontal})$.

負例： $:-\mathrm{put}(a, d, \mathrm{vertical})$.

一つの負例が出てきたところで一度この事例だけで帰納論理プログラミングシステムを用いる．

帰納論理プログラミングは記述最小原理により事例の一般化のために得られた仮説のデータ記述長を最小にする．また正例の中にのみ見られ，負例の中には見られないパターンを見つけ出すことから以下の仮定が導き出される．

$$\mathrm{put}(A, B, C) :- \mathrm{shape}(B, \mathrm{cuboid}).$$

しかしこれは次に以下の問い合わせが成功し正例に加えられることで他の仮定に書き換えられる．

$$\mathrm{put}(b, e, \mathrm{vertical}) \rightarrow \mathrm{true}$$

これは cuboid でないものを用いたことで仮説の条件とは異なる．ここで改めて帰納推論を行うと以下の仮説が導き出される．

$$\mathrm{put}(A, B, C) :- \mathrm{top}(B, C, \mathrm{flat}).$$

新たに追加された正例により，B が C の向きのとき上部が flat(平ら)であれば積木の代用が可能であるということが学習される．これは環境で設定した積木の代用が可能である条件と一致していることから，経験として仮想空間上で物体を使用するという問い合わせから特徴を学習し，その特徴を用いて帰納論理プログラミングから導き出すことができた．

4. 考察

環境への問い合わせにより得た背景知識と正例・負例を用いて，帰納論理プログラミングで学習を行った結果得られる積木の代用のための条件と，環境で設定していた積木の代用の条件とが一致したことから，代用のための知識の獲得ができたといえる．このことから，簡単な環境では代用のための知識の獲得が行なえることを示せた．しかし，今回扱った属性の数はかなり少数であったため，属性が増えた場合に帰納論理プログラミングにより導き出される仮定に変化が生じる可能性がある．また属性の増加につれて必要となる事例の数も多くする必要があると考えられる．そして物体自体の数や，今回は環境に実装していなかった重さや硬さといった物体の持つ属性の数が増加することにより，帰納論理による推論にかかる時間は増加していくため，エージェント内部に記述している背景知識や正例・負例の表現の方法や効率的な処理の方法ついても検討することが必要である．

また今回は極めて単純な環境を論理プログラムで記述し，上部が平らな物の上であれば他の物を積めるという設定をしていたが，実世界環境では平らでなくとも凹みのある部分に積むことや不安定でもバランスをとって積むということも可能である．一方で上部が平らでも傾きや大きさの違い，重心のずれといった要因で必ずしも積めないことがある．このような積めない事例を負例として扱ってしまうと現在の帰納推論では積めない物の条件に入ってしまうため，学習することを想定した論理的な代用の条件が学習できないことになってしまう．そのために例外の事例として扱う条件を設定することも必要であると考える．

5. おわりに

本論文では，限定された環境において積木の代用を行うための知識の獲得を帰納論理プログラミングを用いることで可能にし，論理的表現という枠組みの中で代用ができる可能性を示せた．今回は積木の単純な代用のみを扱ったが将来，さまざまな代用を行うことを目指すとより多くの属性について背景知識として持たせる必要がある．そうなると帰納論理プログラミングにより扱うデータの量が膨大になり，推論の時間もそれに応じて大きく増加してしまう．

これを解決するため，焦点を絞った学習・推論を可能とする方式を今後検討する計画である．

また，実世界環境での実装を将来的に考えると物体の認識や物体の持つ重さや形状，硬さといった属性の表現方法・学習方法について検討する必要がある．この点についてはディープニューラルネットワークを用いて学習を行うことを試みる計画である．

◇参考文献◇

[1] アッタミミ ムハンマド，ファドリル ムハンマド，阿部 香澄，中村 友昭，船越 孝太郎，長井 隆行：多層マルチモーダル LDA を用いた人の動きと物体の統合概念の形成，日本ロボット学会誌，Vol.32，No.8，pp.753-764，2014

[2] 古川 康一，尾崎 知伸，梅野 研：帰納論理プログラミング，共立出版，2001

[3] Muggleton，S. : Inverse Entailment and Progol. New Generation Computing, Vo.13, pp.245-286, 1995