レクチャーシリーズ「知能コンピューティングとその周辺」[第１０回]

論理コンピューティング

logic computing

佐藤 健
Ken satoh
国立情報学研究所
National Institute of Informatics.
ksatoh@nii.ac.jp

デ・ヴォス マリナ
Marina De Vos
英国バース大学
University of Bath
mdv@cs.bath.ac.uk

Keywords: logic programming, negation as failure, abduction, answer set semantics, answer set programming, abductive logic programming

1. は じ め に

論理プログラミングは、表現が論理である利点とそれが効率的に実行することができるという利点があいまって発展してきた。表現が論理である利点とは以下があげられる。

- 論理は数学的表現である。→ 数学的な知見が利用できるため、論理で表現されたものの性質に関する検証がしやすい。
- あいまい性がない表現である。→ 現在の計算機ではあいまい性のある表現での実装は不可能であるし、また論理による表現を用いるとそのような暗黙の仮定が明らかになる。
- 人間が理解可能な推論体系に基づいている。→ 論理による最大の利点は、この点にあると考えられる。論理の起源は人間の「正しい推論」を説明するということである。そのため、人工知能がどのように問題解決をしているかについて説明することに、論理を用いることで人間が理解可能な形式になるという効果がある。
- 簡単な記述から複雑な機能へ展開可能である。→ 簡単なところで真であることを調べておけば、それを公理として推論規則を用いて, より複雑でかつ正しい結果（定理）を推論できる.

このような論理の汎用性は、通常の論理にも存在するが、論理プログラミングの利点は、ホーン節 (Horn clauses) という制限された表現形式を扱うことで、融合原理（resolution principle) などの効率的なメカニズムを利用することにより、通常の定理証明よりもはるかに効率的な実行システムを提供できることにある。

論理プログラミング言語は、1965 年に Robinson が提案した融合原理 (Resolution Principle) と呼ばれる定理証明手法をきっかけにしている。融合原理とは、三段論法を一般化した推論方式で、背理法によって定理の証明をする方式であり、融合原理を使うことによってさまざまな定理が効率的に証明されることがわかり、これが契機となって論理によって人工知能が作れるのではないかという期待が高まった。さらに 1974 年には Kowalski によって一階述語論理のサブセットであるホーン節論理と融合原理を用いた定理証明の過程が手続き的な意味を持つことが明らかにされ [Kowalski74]、この発見を基礎として PROLOG という言語が開発された。これが論理プログラミングの始まりと言われている。また、1982 年に始まった日本の第５世代コンピュータ計画に採用され世界に衝撃を与えた [Shapiro93]。

本稿では、論理コンピューティングのコアともいうべき論理プログラミングの基礎的な事項について述べ、次に、論理プログラミングの新たな意味論である解集合意味論について述べる。そして、その意味論をベースにした知識表現プログラミングとして、仮説論理プログラミングと解集合プログラミングについて述べる。この２つプログラミングは同じ意味論をベースにしているが、知識表現へのアプローチが全く異なっている。仮説論理プログラミングは伝統的な手続的な解釈に基づいているが、解集合プログラミングは宣言的な意味論の方に重点を置いたものであり、両者は論理ベースの知識表現言語の主要な２つのアプローチといえる。

2. 諸　　定　　義

以下、論理プログラミングで必要な諸概念の定義を行う。

2•1 構　　文　　論

論理プログラミングでは、世界の事物間の関係を表す「述語」と呼ばれる式を用いて、知識を表現する。述語は、関係を区別する為の述語名とその述語名の表す

関係に出現する事物を表す引数からなる。述語の引数として許される項 (term) は以下のように定義される。

(1) 定数 $a,b,c,...$(小文字で始まる英数字列) または数字は, 項.

(2) 変数 $X,Y,Z,...$(大文字で始まる英数字列) は, 項.

(3) n 引数関数記号 f_n(小文字で始まる英数字列) に対して, $t_1,...,t_n$ が項ならば $f_n(t_1,...,t_n)$ は項. 関数項と呼ぶこともある.

(4) 以上の条件をみたす式のみが項.

なお、変数を含まない項を基礎項 (ground term) と呼ぶ。

原子論理式またはアトムとは、n 引数述語記号 p_n(小文字で始まる英数字列) に対して, $t_1,...,t_n$ が項ならば $p_n(t_1,...,t_n)$ のことをいう。変数の含まれないアトムを基礎アトム (ground atom) と呼ぶ

ルールとは、“$H \leftarrow L_1,...,L_n$.” という形の式をいう。ここで

- H はアトムである。H をルールの結論部 (head) と呼ぶ。
- 各 L_i は、A をアトムとしたときに、A そのものか `not` A の形の式をいう。L_i が A のとき正リテラル (positive literal)、L_i が `not` A のとき負リテラル (negative literal) と呼び、$L_1,...,L_n$ をルールの本体（body) と呼ぶ。

なお、$L_1,...,L_n$ が空のときがあり、そのときのルール “$H \leftarrow$.” を事実 (fact) と呼ぶこともあり、“H.” で表す。また、H が、いつでも偽の値となる特別なシンボル $\perp$ のときはそのルールは一貫性制約 (integrity constraint) と呼ばれることもある。さらに、ルールに変数を含まない場合、基礎ルール (ground rule) と呼ぶ。

このルールの集合を**論理プログラム**と呼ぶ。なお、`not` のないプログラムを正のプログラムと呼ぶ。

2•2 手続的意味論

論理プログラミングの伝統的な定義においては、「プログラム」の意味は C 言語のようなプログラミング言語における手続的意味として与えられる。このプログラムの手続的意味は、[Kowalski74] で最初に正のプログラムに対して与えられ、さらに負リテラルを含むプログラムに拡張された。直観的には、各述語が一つの手続呼び出しに対応し、同じ述語名を持つルールの集まりがその述語の手続の定義を表している。

ここでは代表的な論理プログラミング言語である PROLOG の処理手続きを抽象化した手続的意味について述べる。この手続は、定理証明の用語を用いれば、「失敗による否定を付加された確定節に対する選択的線形融合手続」(linear resolution with selection function for definite clauses with negation as failure) であり、SLDNF 手続と略す。この名前の由来の通り、この手続は定理証明と密接な関係がある。特に、正のプログラムに対する SLDNF 手続（正確には「失敗による否定」を扱わない SLD 手続）は、プログラムを論理式と見たときの、背理法による目的論理式の証明に対応している。

SLDNF 手続の概要を図 1 に示す。この手続においては、最初に変数を含むアトムの集合（これを**ゴール**と呼ぶ）が入力として与えられ、*solve* という手続きを呼び出す。そして、プログラムを論理的に満たすようなゴールへの変数代入が *solve* から出力される。手続的にいえば、ゴールにおける各リテラルは C 言語における関数呼出しに対応し、それらを呼び出すことにより、出力されるべき変数代入が順次構築されていく。

本手続においては最汎単一化代入 (most general unifier, mgu と略す) が重要な役割をもつ。単一化代入とは、２つのアトムがあったときにその中に含まれている変数になんらかの値を代入することでそれら 2 つのアトムを全く同じ形にする操作である。そして、その代入の中で、代入結果のアトムに変数が一番多く含まれているものを最汎単一化代入と呼ぶ。最汎単一化代入は、手続的に見れば、アトムを関数呼出しとみたときの引数の受渡しに対応する。最汎単一化代入を求めるアルゴリズムを図 2 に示す。図 2 では、変数を含むアトム A_1, A_2 が与えられ、$unify(A_1,A_2)$ を呼び出す。mgu が存在すればそれが、θ に蓄積されたおり、θ を A_1,A_2 に施した結果が A_1(または A_2) に反映される。失敗すれば失敗が返される。

SLDNF 手続におけるゴールの処理の概要は以下のようである。はじめに、ゴール中の任意のリテラルを選ぶ。通常のプログラミング言語では、書かれた順番に手続呼出しを逐次に実行するが、論理プログラミングでは、その順番は任意であり、どの順番でリテラルを処理しても同じ結果が得られる。正リテラルを選んだ場合には、最汎単一化可能な結論部を持つルールを探し、その正リテラルをルールの本体に置き換える。これは、手続き的に見れば、その正リテラルの手続本体の実行を開始することに対応する。また、最汎単一化可能な結論部を持つルールが複数ある場合には、そこに分岐点を設定し、処理を続けていって、実行の失敗が判明した場合に、その分岐点に戻って再度実行を開始する。このように、実行が失敗した場合に他の実行過程をやり直す点も通常のプログラミング言語の実行と異なるところである。また、負リテラル `not` B を選んだ場合には、B から新たに実行を開始して、B の実行が失敗したときに、`not` B の実行が成功する。これは、「**失敗による否定**」(negation as failure)[clark78] と呼ばれ、演繹でいう論理的否定とは異なるが、後で述べるように、ある種の論理的意味論では論理的否定と同等の役目を果たす。知識表現では、否定の知識を扱うこともよくあるため、論理プログラミングで特別

$solve(\{G_1,...,G_n\})$ $\{G_1,...,G_n\}$:アトムの集合
ゴール $\{G_1,...,G_n\}$ 中の, 任意のリテラル G_i (ただし負リテラル `not` B を選ぶときは B は基礎アトムでなければならない) を 1 つ選ぶ.

- G_i が正リテラルのとき、
  (1) 最汎単一化代入 θ により $G_i\theta = H\theta$ となるルール $H \leftarrow L_1,...,L_m$ を選ぶ。
  (2) そのようなルールがないか、すでにそのようなルールをすべて試していれば「失敗」を返す。
  (3) そのようなルールが複数あれば分岐点を設定する。
  (4) ゴール中の G_i を $L_1,...,L_m$ に置き換える。もし、ゴール自体が空となれば代入 θ を返す。ゴールが空でなければ、新たなゴールとして $solve(\{G_1\theta,...G_{i-1}\theta,L_1\theta,...,L_m\theta,G_{i+1}\theta,...,G_n\theta\})$ を呼出し、「失敗」が返されたら直近の分岐点に戻って繰り返す。代入 δ が返されたら θ と δ を合成した代入を返す。
- G_i が負リテラル `not` B のとき、$solve(\{B\})$ を呼出し、「失敗」が返されたら「成功」を返す。

図 1 SLDNF 手続

```
global A1, A2:アトム, θ:代入 (初期状態は空)
unify(A1, A2)
begin
    if(A1 の述語名 ≠ A2 の述語名)then return(fail);
    if(A1 の引数の数 ≠ A2 の引数の数)then return(fail);
    n := A1 の引数の数;
    for i = 1 to n
    begin
        Arg1^i := (A1 の i 番目の引数); Arg2^i := (A2 の i 番目の引数);
        if unifyarg(Arg1^i,Arg2^i)=fail then return(fail);
    end
    return(θ and A1)
end

unifyarg(Arg1, Arg2)Arg1, Arg2:項
begin
    if(Arg1 と Arg2 が定数かつ (Arg1 = Arg2)) then return(success)
    if(Arg2 が変数かつ (subst(Arg2, Arg1) = success)) then return(success)
    if(Arg1 が変数かつ (subst(Arg1, Arg2) = success)) then return(success)
    if(Arg1, Arg2 が関数項)then
    begin
        if(Arg1 の関数名 ≠ Arg2 の関数名)then return(fail);
        if(Arg1 の引数の数 ≠ Arg2 の引数の数)then return(fail);
        n := Arg1 の引数の数;
        for i = 1 to n
        begin
            SubArg1^i := (Arg1 の i 番目の引数); SubArg2^i := (Arg2 の i 番目の引数);
            if unifyarg(SubArg1^i,SubArg2^i)=fail then return(fail);
        end
        return(success)
    end
    return(fail)
end

subst(V,T) V:変数, T:項
begin
    T の中に V が含まれていたら fail を返す.
    θ := θ + ⟨V := Term⟩
    A1 := A1θ; A2 := A2θ;
    return(success)
end
```

図 2 最汎単一化代入を求めるアルゴリズム

実行過程その 1
$\leftarrow add(X,Y,s(s(0)))$
$\downarrow$ unified with $add(0,Y_1,Y_1)$
$\downarrow$ with mgu $\boxed{X := 0}$,$Y := Y_1$,$\boxed{Y_1 := s(s(0))}$
□
$X = 0, Y = Y_1 = s(s(0))$ が解となる.

実行過程その 2
$\leftarrow add(X,Y,s(s(0)))$
$\downarrow$ unified with $add(s(X_1),Y_1,s(Z_1)) \leftarrow add(X_1,Y_1,Z_1)$
$\downarrow$ with mgu $\boxed{X := s(X_1)}$,$Y := Y_1$,$Z_1 := s(s(0))$
$\leftarrow add(X_1,Y_1,s(s(0)))$
$\downarrow$ unified with $add(0,Y_2,Y_2)$
$\downarrow$ with mgu $\boxed{X_1 := 0}$,$Y_1 := Y_2$,$\boxed{Y_2 := s(s(0))}$,
□
$X = s(X_1) = s(0), Y = Y_1 = Y_2 = s(s(0))$ が解となる.

図 3 SLDNF 実行例

に導入されたメカニズムである。

SLDNF 手続きを使った実行例を示す。以下のプログラムを考えてみよう。

$add(0,Y,Y).$ (1)
$add(s(X),Y,s(Z)) \leftarrow add(X,Y,Z).$ (2)

ここで、$s(X)$ は $X+1$ を表す。このプログラムは、第 1 引数と第 2 引数の和が第 3 引数の結果となる関係を表している。このプログラムは単に、足し算の計算をするだけではなく、第 3 引数に値を入れたときに、それを満たす第 1 引数と第 2 引数の組を求めることができる。たとえば、図 3 は、$add(X,Y,s(s(s(0))))$ なる（X+Y=3 となる）X,Y を見つける実行過程を示している (4 つの実行過程があるが 2 つのみ示す。)。図 3 では、空ゴールを □ で表し、空ゴールが導かれた場合に、実行が終了する。図 3 中、$\downarrow$ の右側にゴールと結論が最汎単一化可能なルール及びその時の mgu を示す。

2•3 論理的意味論

論理プログラムには、その名の通り論理的な意味がある。ここでの論理的な意味とは各ルールの $\leftarrow$ を論理的含意記号とみなすことで、論理プログラムを論理式の集合として見たときに、その論理式の集合に対して真となる論理的解釈（論理式のモデルともいう）のうち、論理プログラミングの意味としてふさわしいものに限定されたモデル (intended model) のことをいう。

このモデルを定義するために以下の準備を行う。論理プログラム P に対して, P 内の定数 (なければ任意の定数 c を 1 つ取る) および関数記号から作られる変数を含まない項全体の集合を $H(P)$ と表し, エルブラン領域 (Herbrand universe) と呼ぶ.

〔例 1〕 以下の 3 つのルールからなるプログラム P を考える。

$p(a,f(X)) \leftarrow q(b,g(Y,X)),r(Y).$
$q(b,g(c,b)).$
$r(c).$

すると、

$H(P) = \{a,b,c,f(a),f(b),f(c),g(a,a),g(a,b),...$
$g(c,c),f(f(a)),...,f(g(a,a)),...,$
$f(g(a,f(c))),...\}$ となる.

ルール R 中の変数をすべてエルブラン集合の要素（の組み合わせ）で置き換えたルールの集合を $ground(R)$ で表し**ルールの命題化**と呼び、プログラム中のルールをすべて命題化して得られるプログラムを $ground(P)$ で表す (例 3 参照)。

プログラム P に対して, P 中に含まれる述語から作られる基礎アトムの集合 $B(P)$ を

$B(P) =$
$\{p(t_1,...,t_n) | p$ は P における述語,
各 t_i は $H(P)$ の要素 $\}$

と定義し, **エルブラン基底** (Herbrand base) と呼ぶ.

〔例 2〕 例 1 のプログラム P を考えると、$B(P) =$
$\{p(a,a),p(a,b),p(a,c),p(a,f(a)),p(a,f(b))...,$
$q(a,a),q(a,b),..,q(b,g(c,b)),...,$
$r(c),r(a),r(f(a))....\}$ となる.

そして, B(P) の任意の部分集合 I を**エルブラン解釈**と呼ぶ。エルブラン解釈の直観的な意味は、I に含まれる基礎アトムは真であり、それ以外の基礎アトムは偽であるというものである。さらに、以下の条件を満たすエルブラン解釈を**エルブランモデル**と呼ぶ。

論理プログラム P に対するエルブラン解釈が**エルブランモデル** M とは、以下の条件を満たす解釈である。

- $\perp \notin M$
- $ground(P)$ の要素である基礎ルール $H^g \leftarrow L_1^g,...,L_n^g$ の各 L_i^g に対して、以下が満たされるとき $H^g \in M$
  - L_i^g が基礎正リテラル B のとき、$B \in M$
  - L_i^g が基礎負リテラル `not` B のとき、$B \notin M$

2 番目の条件は、M が、ルールの本体を満たす場合には、必ず結論部が M に含まれることをあらわしている。以降エルブランモデルを単にモデルと呼ぶ。

〔例 3〕 例 1 のプログラム P を考えると、$ground(P)$ は以下のようになる。

$p(a,f(a)) \leftarrow q(b,g(a,a)),r(a).$
$p(a,f(a)) \leftarrow q(b,g(b,a)),r(b). \ldots$
$p(a,f(b)) \leftarrow q(b,g(c,b)),r(c). \ldots$
$p(a,f(g(a,b))) \leftarrow q(b,g(f(a),g(a,b))),r(f(a)). \ldots$
$q(b,g(c,b)).$
$r(c).$

これに対するエルブラン解釈を考えてみると、

$I_1 = \{q(b,g(c,b)),r(c),p(a,f(b))\}$

や

$I_2 = \{q(b,g(c,b)),r(a),r(c),p(a,f(b))\}$

はモデルとなるが、

$I_3 = \{q(b,g(c,b)),r(c)\}$

は、$p(a, f(b))$ が含まれていないのでモデルとならない。プログラム P のモデル M が**極小モデル** (minimal model) であるとは、M の真部分集合でモデルであることがないときをいう。極小モデルが唯一のときには**最小モデル** (minimum model) と呼ぶ。プログラムが正のプログラムの場合には、極小モデルは最小モデルとなり、これを正のプログラムの意味として定義する。つまり、正のプログラムの意味は、モデルとして、最低限真でなければならないアトムのみが真であり、それ以外は偽であると考えることである。これにより正のプログラムのモデルと SLDNF で導かれる基礎項の集合は一致することになる。

しかし、ルール本体に負リテラルを含むルールがプログラム中にあるときには、極小モデルは唯一とならない。したがって、どの極小モデルが正しい意味なのかという問いが生じ、これまでいろいろな提案がなされてきた。すなわち、論理プログラミングの発展の一つはこの負リテラルを含むプログラムの意味をどう捉えるか、ということにある。ここでは意味論の発展については紙面の都合上割愛し[*1]、現在の到達点の一つである解集合意味論 (answer set semantics)（または安定モデル意味論 (stable model semantics) と呼ばれることもある）に基づいたプログラミングについて述べる[*2]

2•4 解集合意味論

解集合意味論は、*Gelfond-Lifschitz* 変換[gellif88, gellif91] と呼ばれる以下の変換を用いて負リテラルを含むプログラムを正のプログラムに変換し、その上での最小モデルを考えるものである。解集合は、この変換されたプログラムの最小モデルが、最初に与えられた解釈と一致した場合に得られる。

【定義 1】 P を基礎ルールの集合とする。基礎アトムの集合 S による P の *Gelfond-Lifschitz* 変換 P^S とは、以下の操作によって得られるプログラム P^S である。それは、P に含まれるルール $H \leftarrow L_1, ..., L_n$ の本体にあるリテラルを、負リテラル $\texttt{not}\ B_1, ..., \texttt{not}\ B_j$ と正リテラル $A_1, ..., A_k$ に分けたとき、すべての B_i が $B_i \notin S$ のときかつそのときのみ $H \leftarrow A_1, ..., A_k \in P^S$ になるプログラムである。

この P^S の直感的な意味は、S に含まれない命題を偽としたときにプログラム P 内で有効なルールの集合を集めたものである。

*1 興味のある読者は [minker93, apt94] を参照されたい。

*2 正確には、安定モデル意味論は、通常の論理プログラミングに対する意味論であり、解集合意味論は、論理プログラミングにもう一つの論理的否定 (explicit negation) を導入した拡張論理プログラミング (extended logic programming) に対する意味論である。しかし、本稿では論理的否定を扱わず、通常の論理プログラミングでは両者は一致するため、あとで述べる解集合プログラミングとの整合性のため解集合意味論と呼ぶことにする。

$guilty \leftarrow evidence.$ (1)
$evidence \leftarrow trusted\_witness.$ (2)
$trusted\_witness \leftarrow \texttt{not}\ lying, witness.$ (3)
$witness.$ (4)
$believe \leftarrow \texttt{not}\ disbelieve.$ (5)
$disbelieve \leftarrow \texttt{not}\ believe.$ (6)
$lying \leftarrow disbelieve.$ (7)

図 4 判事の判断に関する論理プログラム

$guilty \leftarrow evidence.$ (1)
$evidence \leftarrow trusted\_witness.$ (2)
$trusted\_witness \leftarrow witness.$ (3)'
$witness.$ (4)
$believe.$ (5)'
$lying \leftarrow disbelieve.$ (7)

図 5 P^{M_1} のルール集合

【定義 2】 P をプログラムとする。エルブラン解釈 M が P の解集合であるとは、M が $(ground(P))^M$ の最小モデルであることである。

上記の定義の意味は、P から導かれると仮定したアトムの集合 M と P から M 自身が過不足なく導かれる、ということである。すなわち、まず解集合ではないかというアトムの集合 M を考えて、M と P とが矛盾しない場合に、M を P の解集合であると考えるわけである。また、このような定義を行うと、アトム A が M で真であるときには P の中に必ず A の根拠となるルールが存在することが保証されており、その意味でも望ましい定義となっている。

ただし、この定義からは、プログラムによっては解集合が複数存在することもある。しかし、このような解集合の非決定性は、以下の例が示すようにプログラムの意味が本質的に非決定的な場合にそれを直接表現できるため、解集合意味論の利点と見ることもできる。

〔例 4〕 判事が証人によって与えられた証拠を元に被告人が有罪かどうかを決める状況を考えてみよう。この場合には、判事は証人を信用するかどうかを決定しなければならない。この決定プロセスを反映したプログラム P を図 4 に示す。この解集合を求めてみよう。たとえば $\{guilty, evidence, trusted\_witness, witness, believe\}$ という集合 M_1 を考えてみる。すると、P^{M_1} は図 5 に示すルールの集合になる。

P のルール (6) では、$\texttt{not}\ believe$ が本体にあるが、M_1 では $believe$ は真であるため、ルール (6) 自体は条件部が真とならないので削除される。ルール (3) と (5) には、それぞれ $\texttt{not}\ lying$, $\texttt{not}\ disbelieve$ が本体に含まれているが、M_1 では $lying, disbelieve$ は偽であるため、それらは各ルールの本体から取り除かれる。そして、P^{M_1} の最小モデルを求めると M_1 と一致するので M_1 が解集合となる。一方、$\{witness, lying, disbelieve\}$ という集合 M_2 を考えて P^{M_2} の最小モデルを求めると M_2 に一致することがわかる。したがって M_2 も解集

$fly(X) \leftarrow bird(X), normal\_bird^*(X).$
$bird(X) \leftarrow penguin(X).$
$bird(X) \leftarrow canary(X).$
$\bot \leftarrow fly(X), penguin(X).$
$penguin(t).$
$canary(a).$

図 6 デフォルト推論の例

合となる。したがって、このプログラムからは 2 つの解集合が得られるが、これは判事が証人を信じるかどうかに依存した解が 2 つあることと対応しているので、妥当な結果と考えられる。

以降ではこの解集合意味論をベースにして発展した 2 つの研究（仮説論理プログラミングと解集合プログラミング）について述べる。

3. 仮説論理プログラミング（Abductive Logic Programming）

3•1 概　　要

仮説推論はアブダクション (abduction) とも呼ばれ、演繹推論、帰納推論とともに重要な人間の推論の一つである[*3]。仮説推論とは、現在正しいとわかっている論理式を T とし、今、観察された事実を O としたとき、T のみでは O を説明できないときに、仮説 E を加えることで O を説明しようとする推論である。この仮説推論を論理プログラミングにおいて可能にしたものが仮説論理プログラミング（Abductive Logic Programming, ALP と略す）[kakas92] である。ALP においては、仮説述語を用意し、その仮説述語を使った基礎アトムを仮説集合として、与えられたゴールを証明する。論理プログラムを P, 仮説述語の集合を A とするとき、$\langle P, A\rangle$ のペアを仮説フレームワーク (abductive framework) と呼ぶ。この仮説フレームワークに対する意味論は、一般安定モデル (generalized stable model) と呼ばれる [kakas90b]。A から作られる基礎アトムの集合の部分集合を Δ とするとき、$M(\Delta)$ が $\langle P, A\rangle$ の一般安定モデルであるとは、$M(\Delta)$ が $P \cup \Delta$ の解集合のときをいう。そして、ALP の手続きでは、変数を含むゴール G を入力として与え、以下のような変数への基礎項の代入 θ および仮説 Δ を出力する。その代入とは、ある一般安定モデル $M(\Delta)$ が存在して、$G\theta \subseteq M(\Delta)$ になる代入のことである。

プログラム例を図 6 に示す。このプログラムは、デフォルト推論の例である。本プログラムでは、仮説述語は $normal\_bird^*$ であり、「通常の鳥である」という仮説を表す。このときの $fly(X)$ について質問をした場合に、$X = a$ 並びに $\Delta = \{normal\_bird^*(a)\}$ が得られる。この質問に対して $X = t$ は得られない。なぜならば、そうなると $fly(t)$ と $penguin(t)$ が導かれてしまい、４番目の一貫性制約により矛盾が導かれてしまうからである。

なお、この一般安定モデルは通常の解集合モデルに変換できることが知られている [satoh91]。具体的には仮説述語 p^* が仮説フレームワークに含まれていたときに、それに対して新しい 2 つの述語 p および p' を導入し、プログラム中の p^* を新しい p に置き換え、さらに

$p(X_1, ..., X_n) \leftarrow \texttt{not}\ \ p'(X_1, ..., X_n).$
$p'(X_1, ..., X_n) \leftarrow \texttt{not}\ \ p(X_1, ..., X_n).$

をプログラムに付け加えることで、p^* が入る一般安定モデルと入らない一般安定モデルをシミュレートすることができる[*4]。

3•2 ALP の実行手続

仮説論理プログラムを実行する手続は、Kakas らによって最初に提案された [kakas90a]。この手続は SLDNF 手続を基本としているが、仮説述語の扱いが追加されている。SLDNF 手続では処理の途中で変数代入の蓄積を行っているが、ALP では、さらに仮説の蓄積も行う。そして、ゴール中の仮説述語のリテラル（仮説リテラルと呼ぶ）が選ばれた場合には、もしそれが既に蓄積されていればすぐに成功し、仮説リテラルが蓄積されていなければそのリテラルを蓄積する。それとともに、それが含まれている一貫性制約の無矛盾性のチェックを行う。このため、この手続においては、一貫性制約には必ず 1 つの仮説が含まれているという制限が含まれていた。さらに、より深刻な問題として、一般安定モデルでは真とならないゴールが成功してしまうという健全性の問題があった。なお、これらの問題は [satoh92] において解決されている。

その後、さまざまな処理系が提案されている。その中には、制約論理プログラミングという一般的な制約を扱える論理プログラミングと組み合わせた ACLP システム [kakas00]、仮説論理プログラムの実行を論理的な同値関係の置き換えで行う CIFF[endriss04]、[kakas90a] における冗長な処理を除去した効率的な処理系 ProLogICA[ray06] 等がある。

3•3 ALP の 応 用

ALP の応用としては、仮説を知識表現におけるどの対象とするかによってさまざまなものがある。スケジューリングやプラニングにおいては、オペレーションを仮説として捉え、ある制約を満たす解をその仮説の組み合わせで求めるという方法が考えられる。そのような応用として航空会社の乗務員のスケジューリングに応用した例がある [denecker02]。また、一般的な

---

*3 アブダクションの詳しい説明については [inoue92] を参照されたい。

*4 同様の技法が、拡張論理プログラミングでの仮説推論 [inoue91, pereira91] で提案されている。

プラニングへの応用の研究もある [shanahan00]。さらに、非常に興味深い ALP の応用として音楽への応用がある [kobayashi08]。[kobayashi08] では、演奏法を仮説として捉え、演奏者の肉体の制約を一貫性制約で記述したときにスムーズな演奏法を ALP によって求めている。

筆者の一人も ALP のさまざまな応用を提案してきた。仮説推論を用いて、法的推論において被告、原告の立場に応じて判例の仮説的な類似性を動的に変えるシステム [satoh97]、ソフトウェア工学において仮説を仕様変更に伴うプログラム中の修正場所に対応させることで、仕様変更が生じたときに、どの場所を修正すればよいかを発見するシステム [satoh00]、また、マルチエージェントシステムへの応用として、エージェントの予測を仮説推論で行い、その予測が後で間違いであることがわかったときに、動的にエージェントの行動を修正していく投機的計算 (speculative computation) の枠組みを提案している [satoh03]。

3•4 ALP の限界と今後の動向

以上見てきたように ALP は、伝統的な手続き的意味を持つ論理プログラミングを継承したプログラミング手法である。したがって、論理プログラミングでの限界と ALP の限界は重なるものが多い。たとえば、論理プログラミングでは、実行が分岐して失敗した場合には分岐点に戻って実行過程をやり直すが、失敗までに実行した結果はすべて失われてしまう。つまり失敗から学ぶことはできない機構になっている。ALP でも同じ機構となっているので、ある分岐においてある仮説を使うと失敗することがわかっても、その仮説を別な分岐で使わないようにするような機構が用意されていない。このような制御構造の洗練化が今後の課題の一つといえる。また、現在 ALP の特徴を生かすような実用的な応用が提案されていないことも問題と思われるので、より高次の知識処理の応用を探すことも課題である。

4. 解集合プログラミング（Answer Set Programming）

4•1 概　　要

解集合プログラミング[baralbook](Answer Set Programming, ASP と略す。) においては、論理プログラムは、問題の解が満たすべき条件を記述するのに使われる。プログラムの解集合は、解集合意味論によって定義され、それが問題の解となる。これは従来の手続意味論を持つ論理プログラミングと全く異なることに注意されたい。従来の論理プログラミングでは、質問応答処理によりゴール中の変数代入により解を表現していたが、ASP においては、モデル自体が解となる。したがって、ASP にはC言語のようなプログラミング言語としての手続的な意味はないことに注意されたい。そういう意味では、解集合プログラミングでの「プログラミング」は、プログラミング言語における「プログラミング」の意味ではなく、線形計画法 (linear programming) や整数計画法 (integer programming) における “programming” の意味と同じである。

$$
\begin{aligned}
&position(1..9). && (1)\\
&value(1..9). && (2)\\
&1\ \{state(X,Y,N) : value(N)\}\ 1 \leftarrow \\
&\quad position(X), position(Y). && (3)\\
&\bot \leftarrow state(XA,Y,N), state(XB,Y,N),\\
&\quad position(XA), position(XB),\\
&\quad position(Y), value(N), XA \neq XB. && (4)\\
&\bot \leftarrow state(X,YA,N), state(X,YB,N),\\
&\quad position(X),\\
&\quad position(YA), position(YB),\\
&\quad value(N), YA \neq YB. && (5)\\
&subRegion(X,1) \leftarrow 1 \leq X \leq 3. && (6)\\
&subRegion(X,2) \leftarrow 4 \leq X \leq 6. && (7)\\
&subRegion(X,3) \leftarrow 7 \leq X \leq 9. && (8)\\
&sameSubSquare(XA,YA,XB,YB) \leftarrow \\
&\quad subRegion(XA,XR), subRegion(XB,XR),\\
&\quad subRegion(YA,YR), subRegion(YB,YR). && (9)\\
&\bot \leftarrow state(XA,YA,N), state(XB,YB,N),\\
&\quad sameSubSquare(XA,YA,XB,YB),\\
&\quad position(XA), position(XB),\\
&\quad position(YA), position(YB),\\
&\quad value(N), XA \neq XB, YA \neq YB. && (10)
\end{aligned}
$$

図 7 数独の解集合プログラム

この節では、ASP の概要を述べる。まず、ASP のプログラム例を示し、複雑な問題が簡単にモデル化できることを示す。次に、ASP での処理アルゴリズムについて述べ、最後に ASP の応用について述べる。なお、より詳しいことが知りたい読者は [baralbook, gellif88, iwayama01, inoue08] を参照されたい。

ASP の表現力を示すために数独パズルのプログラム例を示す。数独パズルとは、パズルの一種で 3×3 の升目が 3×3 に並んでいる計 81 個の升目に対して 1 から 9 の数字を入れるが、以下の制約を満たす配置を求める問題である。

- 3×3 の升目には、1 から 9 の数字をすべて入れる。
- 縦一列、横一列には同じ数字を入れてはいけない。

図 7 に数独パズルの解集合プログラミングによる問題表現を示す。

最初の 2 つの事実 (1)(2) は座標位置と値の可能な値を表す。$position(1..9).$ は、$position(X)$ の X の値として取れる値が 1 から 9 であることを表す。これは升目の座標位置の最大値が 9 であることに対応している。また、$value(1...9).$ は、各升目に入る値の範囲が 1 から 9 であることを表す。これらは ASP のルールに変換することができる。

ルール (3) は、9×9 の各升目の値は一意であることを表す。このルールの中の $state(X,Y,N)$ で升目 (X,Y)

の位置にある数字が N であることを表す。ルール (3) の “1 $\{state(X,Y,N) : value(N)\}$ 1” は、ASP における特別の記法である。一般に、$L\{A(X_1,...,X_n,Y_1,..,Y_m) : B(Y_1,...,Y_m)\}M$ は、$A(X_1,...,X_n,Y_1,..,Y_m)$ の $X_1,...,X_n$ の値の組が決まったときに $Y_1,..,Y_m$ の組は $B(Y_1,...,Y_m)$ を満たし、さらに $Y_1,...,Y_m$ として取りうる値の組み合わせの個数は L 以上 M 以下であることを表す。したがって、1 $\{state(X,Y,N) : value(N)\}$ 1 は $state(X,Y,N)$ の X,Y の値が決まれば（つまり位置が決まれば)、N の値は $value(N)$ の取りうる値の範囲内（つまり 1 から 9）であって一意に定まる、ということを意味する。この記法も通常の ASP のルールへ変換することができる。

一貫性制約 (4),(5) は、同じ行や同じ列には同じ値が現れないことを表す。たとえば (4) では、2 つの升目の位置が (XA,Y)、(XB,Y) に入る数字が同じ数字 N であったときに XA,XB が違う値であったら矛盾であることを表す。

次のルール (6)〜(9) は、3 × 3 の升目の同じ正方形に入る位置の組み合わせを生成するために用いられる。(6)〜(9) で用いられている $subRegion(X,M)$ は、一列または一行の 9 つの升目の位置 X を 3 つの領域に分けて領域番号 M をつけるために用いられる。たとえば X が 2 のとき（つまり列でいえば 2 番目の升目、行でいえば左から 2 番目の升目のとき）は、その升目は第 1 領域（つまり、列でいえば一番上の領域、行でいえば一番左の領域）に属することを表す。また $sameSubSquare$ は 9 × 9 の升目を 9 つの 3 × 3 の升目の正方形に分けたときに 2 つの升目の位置が同じ正方形に入っていることを表す。

最後の一貫性制約 (10) は 3 × 3 の正方形の中には同じ値が現れないことを表す。もし 2 つの升目の位置が (XA,YA)、(XB,YB) であって、それらが同じ 3 × 3 の升目の正方形に入っていて、その 2 つの位置に入る数字が同じ数字 N のときには、その升目の位置は一致しなければ矛盾であることを表す。

このプログラムを実行すると数独パズルのすべての解を得ることができる。もし升目の中にあらかじめ値の入っている数独パズルを解くときには、値 N が入っている地点 (X,Y) ごとに、$state(X,Y,N)$ をプログラムに追加すればよい。

数独パズルのような NP 完全問題は、このように ASP で直観的に表すことができる。より多くの ASP のベンチマークが Asparagus System として集められているので興味のある読者は参照されたい [asparagus]。

4•2 ASP ソルバー

解集合プログラミングにおける解集合を求めるアルゴリズムやシステムのことを総称して *ASP* **ソルバー**と呼ぶ。有名なシステムとしては、DLV[eilemapfsc98], SMODELS[nisi97], SMODELS-IE[bdvs07] がある。また、ASP プログラムを論理式の充足問題に変換して解く CMODELS[satasp] や、その発展形である CLASP [gekanesc07a]、分散環境で実行される PLATYPUS [grjamesctht i05a] がある。これらのソルバーの比較については 2007 年に Schaub と Truszczyński の開催した [gelinanesctr07a] を参照されたい。

さて、ASP プログラムが与えられたとき、解集合は以下のように計算される。

(1) プログラム中の変数をすべて定数で置き換えて、変数のないプログラムを作る。
(2) 以下で述べる技法を使って、変数のないアトムすべての真偽値を決定する。

アトムの真偽値を決定する方法としては大きく分けて以下の 2 つの技法（分岐伝播法 *(branch-and-propagation)*、節学習法 *(clause learning)*）が使われている。

分岐伝播法は、[eilemapfsc98, nisi97, bdvs07, grjamesctht i05a] で用いられている技法で、解集合を部分的に作り上げて行く方法である。最初にプログラム上明らかに真であるアトムと偽であるアトムの集合を見つけ、その集合とルールから導かれるアトムの集合に拡張する（伝播フェーズ）。そのようなアトムがなくなった場合には、真偽が不明な（通常は解集合を決定付ける重要な）アトムを一つ選び、真の場合と偽の場合に分岐し（分岐フェーズ)、再び伝播フェーズに入る。そしてすべてのアトムの真偽値が決定されたときに終了する。

節学習法においては、節の完備化 [clark78] を用いて ASP プログラムを節形式に変換し、それを充足する真偽値割り当てを求める。その過程で、矛盾が検出された場合、その矛盾の原因となる節を発見してそれを加えることで探索空間を狭めていき、効率的に解集合を求める。

4•3 ASP の応用

ASP ソルバーの効率化および ASP の利点、欠点が明らかになることにより、ASP の研究者は、このプログラミング手法の適切な応用を発見している。以前は、知識表現や推論での応用が主流であったが、現在は、制約充足のような問題に対しても応用できることが分かってきた。

一番良く知られている応用分野としては、プラニングと故障診断があげられる [eiter02, lif00, nobagewaba01]。この中には、スペースシャトルの推進システムのモデリングおよび再スケジューリングプログラムのような産業応用プログラム [nobagewaba01] もある。

他の応用としては、製品構成問題への応用がある [soininen99]。この応用では、製品構成要素とその間の制約が与えられたときに、すべての仕様を満たす製品

構成を計算する。

また、マルチエージェントシステムへ応用もある [baral00, dvver04, cdvp06c, sain08]。ASP は一つのエージェントをモデル化したりシステム全体の振る舞いを検証するような問題において役に立つことが示されている。

セマンティックウェブや Web 関連技術は現時点で ASP でも大変盛んな研究領域である。[polleres05, RuffoloLMSZ05] はこの領域において ASP が適用可能であることを示している。

4•4 ASP の限界と今後の動向

上で述べた様に ASP は、線形プログラミングや整数プログラミングと同じような目的で開発されてきているため、それらの制約ソルバーと同じように問題解決エンジンとして使われていくことになるであろう。このためには、少なくとも線形プログラミングや整数プログラミングと同じかそれ以上の性能を出すことが望ましい。そのために ASP のソルバーに関する研究はさらに深化していくと思われる。また、表現能力を高める為に、解の選好順序や最適化をも ASP で表現できるような手法が研究されていくと考えられる。

しかし、ASP は、そのような効率的な解探索を目指すため、伝統的な論理プログラミング言語の表現力を犠牲にしている。一番大きな犠牲は、関数項を扱えないことである。これは、変数の数を有限にするために必要な制限であるが、柔軟な知識表現ができないという問題がある。このため、構造的なデータをいかにして表現していくかが、今度の大きな課題と考えられる。

5. お わ り に

以上、論理コンピューティングの代表的技術である論理プログラミングの基礎的な理論と、最近の研究動向である解集合意味論に基づいた仮説論理プログラミングや解集合プログラミングについて概説を行った。最後に論理コンピューティングにおける今後の課題について述べる。論理プログラミングにおいては、すでに知識を表現する為に、述語や関数などの言語要素があらかじめ用意され、さらにそれらがルールの形で表現されていることが前提である。すると、未知の問題に対してどのような述語を用意すればいいか、どのようにルールを作ればいいかということに関して、今まで述べた論理プログラミング自体は答えてくれない。この問題を解決することを目指して、帰納論理プログラミングや確率論理プログラミングが提案されている。さらに最近では、それらの分野について顕著な発展があるが、それらについては、紙面の都合上紹介することができなかった。[Yamamoto06, sato07] を参照されたい。

謝辞：本稿に関して貴重なご意見をいただいたＮＩＩの市瀬龍太郎先生ならびに細部博史先生、京都大学の西田豊明先生に感謝いたします。

◇ 参 考 文 献 ◇

[apt94] K.R. Apt, R. Bol. Logic programming and negation: a survey. J. Logic Programming, Vol. 19 and 20, pp. 9–71 1994.

[asparagus] The asparagus benchmark system. http://asparagus.cs.uni-potsdam.de/.

[baralbook] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[baral00] Chitta Baral and Michael Gelfond. Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*, pp. 257–279. Kluwer Academic publishers, 2000.

[bdvs07] Martin Brain, Marina De Vos, and Ken Satoh. SMODELS-IE: Improving the cache utilisation of smodels. In *Proceedings of the 4th Workshop on Answer Set Programming: Advances in Theory and Implementation*, pp. 309–313, Porto, Portugal, September 2007.

[clark78] Keith Clark. Negation as Failure Logic and Database, pp. 293–322, 1978.

[cdvp06c] Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and analysing agent-based social institutions using answer set programming. LNCS 3913, pp. 99–113, 2006.

[dvver04] Marina De Vos and Dirk Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artifical Intelligence*, Vol. 42, No. 1–3, pp. 103–139, September 2004. Special Issue on Computational Logic in Multi-Agent Systems.

[denecker02] M. Denecker, A. C. Kakas Abduction in Logic Programming, Computational Logic: Logic Programming and Beyond, LNCS 2407, pp. 99–134, 2002.

[eiter02] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. The DLV^k planning system. *Proc. of JELIA 2002*, LNAI 2424, pp. 541–544, 2002.

[eilemapfsc98] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pp. 406–417. Morgan Kaufmann, San Francisco, California, 1998.

[endriss04] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF Proof Procedure for Abductive Logic Programming with Constraints. *Proc. of (JELIA-2004)*, pp. 31–43, 2004.

[gekanesc07a] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 386–392. AAAI Press/The MIT Press, 2007. Available at http://www.ijcai.org/papers07/contents.php.

[gelinanesctr07a] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, and M. Truszczyński. The first answer set programming system competition. In Baral, et al. [lpnmr07], pp. 3–17.

[gellif88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pp. 1070–1080, Seattle, Washington, August 1988. The MIT Press.

[gellif91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, Vol. 9, No. 3-4, pp. 365–386, 1991.

[satasp] Enrico Giunchiglia, Yuliya Lierler, and Marco

Maratea. SAT-Based Answer Set Programming. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-04)*, pp. 61–66, 2004.

[grjamescthti05a] J. Gressmann, T. Janhunen, R. Mercer, T. Schaub, S. Thiele, and R. Tichy. Platypus: A Platform for Distributed Answer Set Solving. In *Proc. of LPNMR'05*, pp. 227–239, 2005.

[inoue91] K. Inoue. Extended Logic Programs with Default Assumptions. *Proc. of ICLP'91*, pp. 490–504, 1991.

[inoue92] 井上 克巳. アブダクションの原理. 人工知能学会誌, Vol. 7, No.1, pp. 48–59, 1992.

[inoue08] 井上 克巳、坂間千秋. 論理プログラミングから解集合プログラミングへ. コンピュータソフトウェア, Vol.25, No.3, 2008.

[iwayama01] 岩山 登、佐藤 健. 論理プログラミングの解集合意味論に関する証明系. 人工知能学会誌, Vol. 16, No.5, pp. 655–660, 2001.

[kakas90a] A.C. Kakas, P. Mancarella, P. Database Updates through Abduction. *Proc. of VLDB'90*, pp. 650–661, 1990.

[kakas90b] A.C. Kakas, P. Mancarella, P. Generalized Stable Models: A Semantics for Abduction. *Proc. of ECAI'90*, pp. 385–391, 1990.

[kakas92] A. C. Kakas, R. Kowalski, F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, Vol.2 No.6, pp. 719–770, 1992.

[kakas00] A. C. Kakas, A. Michael, C. Mourlas. ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, Vol. 44, Nos. 1-3, pp. 129–177, 2000.

[kobayashi08] I. Kobayashi, K. Furukawa. Modeling Physical Skill Discovery and Diagnosis by Abduction. 人工知能学会論文誌, Vol. 23, pp 127–140, 2008.

[Kowalski74] R. Kowalski. Predicate Logic as a Programming Language. Proc. of IFIP, pp. 556–574, 1974.

[lif00] Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, Vol. 138, No. 1-2, pp. 39–54, 2002.

[lpnmr05] *Proc. of 8th International Conference on Logic Programming and Nonmonotonic Reasoning(LPNMR'05)*, LNAI 3662, 2005.

[lpnmr07] *Proc. of 9th International Conference on Logic Programming and Nonmonotonic Reasoning(LPNMR'07)*, LNAI 4483, 2007.

[minker93] J. Minker. An overview of nonmonotonic reasoning and logic programming. *Journal of Logic Programming*, Vol.17, Nos. 1-4, pp. 95-126, 1993.

[nisi97] I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. LNAI 1265, pp. 420–429, 1997.

[nobagewaba01] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. A A-Prolog Decision Support System for the Space Shuttle. In *Answer Set Programming: Towards Efficient and Scalable Knowledge Represenation and Reasoning*. American Association for Artificial Intelligence Press, Stanford (Palo Alto), California, US, March 2001.

[pereira91] L. M. Pereira, J. N. Aparicio, J. J. Alferes. Nonmonotonic Reasoning with Well Founded Semantics. *Proc. of ICLP'91*, pp. 475–489, 1991.

[polleres05] Axel Polleres. Semantic web languages and semantic web services as application areas for answer set programming. In Gerhard Brewka, Ilkka Niemelä, Torsten Schaub, and Miroslaw Truszczynski, editors, *Nonmonotonic Reasoning, Answer Set Programming and Constraints*, Vol. 05171 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[ray06] O. Ray and A. Kakas. ProLogICA: a practical system for Abductive Logic Programming. *Proc. 11th Int. Workshop on Non-monotonic Reasoning*, pp. 304–312, 2006.

[Robinson65] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. Journal of the ACM, Vol. 12. No.1, pp. 23–41, 1965.

[RuffoloLMSZ05] Massimo Ruffolo, Nicola Leone, Marco Manna, Domenico Saccà, and Amedeo Zavatto. Exploiting asp for semantic information extraction. In Marina De Vos and Alessandro Provetti, editors, *Answer Set Programming*, Vol. 142 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[sain08] Chiaki Sakama and Katsumi Inoue. Coordination in answer set programming. *ACM Transactions on Computational Logic*, Vol. 9, No. 2, 2008.

[sato07] 佐藤 泰介, 亀谷 由隆. グラフィカルモデルにおける論理的アプローチ. 人工知能学会誌, Vol.22, No.3, pp. 306–319, 2007.

[satoh91] K. Satoh, N. Iwayama, N. Computing Abduction by Using TMS. *Proc. of ICLP'91*, pp. 505–518, 1991.

[satoh92] K. Satoh, N. Iwayama. A Query Evaluation Method for Abductive Logic Programming. *Proc. of JICSLP'92*, pp. 671–685, 1992.

[satoh97] 佐藤 健. 事例ベース推論における動的類似性の仮説論理プログラミングによる実現. 人工知能学会誌 Vol. 12, No.6, pp. 901–910, 1997.

[satoh00] 佐藤 健. 仮説論理プログラミングによる極小変更仕様の計算およびその応用. コンピュータソフトウェア別冊,「ソフトウェア発展」, 日本ソフトウェア科学会編, pp. 109–121, 2000.

[satoh03] 佐藤 健, 井上 克己, 岩沼 宏治, 坂間 千秋. エージェント間通信におけるアブダクションによる投機的計算. コンピュータソフトウェア, Vol. 20 No.1, pp. 27–35, 2003.

[shanahan00] M. Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, Vol. 44, Nos. 1-3, pp. 207–240, 2000.

[Shapiro93] E. Y. Shapiro, D. H. D. Warren, K. Fuchi, R. A. Kowalski, K. Furukawa, K, Ueda, K. M. Kharn, T. Chikayama, and E. Tick. The Fifth Generation Project: Personal Perspective, C.A.C.M. Vol. 36 No. 3, pp. 46–103, 1993.

[soininen99] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL '99)*, LNCS, San Antonio, Texas, 1999. Springer.

[Yamamoto06] 山本章博. 帰納論理プログラミングの基礎理論とその展開. コンピュータソフトウェア, 23(2), pp. 29–44, 2006.

2008 年 7 月 1 日 受理

著 者 紹 介

De Vos, Marina(非会員)

Marina De Vos joined the University of Bath, UK, in 2002, after completing her Ph.D. at the Vrije University of Brussel, Belgium. Currently, her main research area is Answer Set Programming (ASP). Within that area, she is interested in the theoretical formalisations, applications and software engineering aspects. Application areas include: game theory, multi-agent systems, virtual institutions, superoptimisation and music synthesis. She is an international author and appears regularly on programme committees in the field. From 2002 to 2006, she was vice coordinator of the European Working Group on Answer Set Progamming.

佐藤 健(正会員)

1981 年 東京大学理学部情報科学科卒業. 同年 (株) 富士通研究所入社. 1984 年 英国インペリアル大学客員研究員. 1987 年〜92 年 (財) 新世代コンピュータ技術開発機構出向. 1995 年 北海道大学工学部助教授. 2001 年より国立情報学研究所教授. 人工知能の基礎研究に従事. 博士 (理学)