特　集「論理に基づく推論研究の動向」

解集合プログラミング

Answer Set Programming

坂間　千秋　和歌山大学システム工学部
Chiaki Sakama　Faculty of Systems Engineering, Wakayama University.
sakama@sys.wakayama-u.ac.jp, http://www.wakayama-u.ac.jp/~sakama

井上　克巳　国立情報学研究所
Katsumi Inoue　National Institute of Informatics.
ki@nii.ac.jp, http://research.nii.ac.jp/~ki

Keywords: logic programming, answer set semantics, knowledge representation.

1. はじめに

「論理プログラミング＝PROLOG」と捉える人が多いのではないだろうか．1970年代初頭に開発された論理型プログラミング言語PROLOGは，述語論理に基づく宣言的記述と定理証明に基づく計算手続きを併せもつ新しい人工知能言語として注目され，1980年代にはエキスパートシステム，自然言語処理，演繹データベースなどに幅広く応用された．PROLOGはチューリング完全で高い計算能力をもつ一方で，そのベースとなるホーン節論理プログラムは構文上の制約や推論能力の不足により，現実の知識表現や問題解決への適用は限定的であることも明らかになった．こうした問題を克服するために，論理プログラミングの表現能力を拡張し，推論機能を強化するための試みが1980年代後半から数多く提案された．その結果，1990年代後半頃から，論理プログラミングと制約プログラミングの概念を融合した**解集合プログラミング**（answer set programming）の概念が確立され，現在では論理プログラミングの中心的な言語の一つになっている．本解説では，知識表現言語の観点から論理プログラミングの構文論，意味論が発展してきた歴史的経緯について概観した後，近年，新しいパラダイムとして注目されている解集合プログラミングの枠組みと最近の研究動向について解説する*1．なお，本解説は紙数の制限上，関連研究をすべて網羅しているわけではない．特に論理プログラミングと非単調論理の関係についてはほとんど触れていない．また仮説推論に関する話題については，本特集に解説記事 [井上 10] があるため割愛した．本解説では読みやすさを第一に考え，諸概念の厳密な定義は必要最小限に留めることにした．本稿で未定義の概念や厳密な定義は [古川 08, Lloyd 87] などの文献を参照されたい．

2. 歴史的背景

2・1　確定論理プログラム

初期の**論理プログラミング**（logic programming）は，一階述語論理の部分クラスである**ホーン節論理**（Horn logic）に手続き的解釈を与えたものである [Kowalski 74]．ホーン節とは以下の形式をした含意式

$$A \leftarrow A_1, \ldots, A_m \qquad (1)$$

（$A, A_1, \ldots, A_m$ はアトム）で，←の左側を節（1）の**帰結部**あるいは**ヘッド**，右側を**条件部**あるいは**ボディ**と呼ぶ．ボディにおけるカンマ（ , ）は連言を表す．特に，ヘッドが空でない確定節から構成される集合を**確定論理プログラム**（definite logic program）と呼ぶ．確定論理プログラムの**宣言的意味論**は，プログラムの**最小モデル**（least model）により与えられる．[van Emden 76] は確定論理プログラムの最小モデルが，プログラム上で定義される連続関数の**最小不動点**（least fixpoint）として操作的に計算可能であることを示した．また確定論理プログラムの**手続的意味論**として導入された**SLD導出**（resolution）*2は，ヘッドが空のホーン節を証明すべきゴールとみなしてプログラムに対して**反駁**（refutation）計算を行う．確定論理プログラムの宣言的意味と手続的意味が一致することは，最小モデルに含まれる基礎アトム*3の集合とSLD導出によって証明される基礎アトムの集合が一致する事実により保証される．

確定論理プログラムの最小モデルに含まれる基礎アトムは，プログラムから演繹される真の事実を表

*1 解集合プログラミングに関しては，ほかの解説 [井上 08, 佐藤 08] もあるので参照されたい．本解説ではこれらとの重複は極力少なくなるようにし，歴史的背景と最近の話題を多く取り上げている．

*2 resolutionは「融合」と訳されることもある [古川 08]．

*3 項，アトム，リテラルで変数を含まないものをそれぞれ基礎項，基礎アトム，基礎リテラルと呼ぶ．

す．これに対して，プログラムから演繹されない基礎アトムを偽と解釈する立場を**閉世界仮説**（closed world assumption: CWA）[Reiter 78] と呼ぶ．一方，SLD 導出による証明が有限ステップで失敗するような基礎アトムを偽と解釈する方法を**失敗による否定**（negation as failure: NAF）[Clark 78] と呼ぶ．NAF のもとで手続的に定義される否定事実の集合は，プログラムに対して**完備化**（completion）と呼ばれる変換手続きを施した論理式集合から演繹的に導かれる否定事実の集合と一致する [Clark 78]. ある基礎アトムが NAF の下で偽と解釈される場合は CWA の下でも偽と解釈されるが，その逆は一般にいえない [Shepherdson 84]．CWA や NAF はプログラム中で真であることが証明できない不完全な情報を偽と解釈することで，プログラムを完全化し否定情報の推論を可能にする．一方，こうした**デフォルト推論**（default reasoning）は述語論理における論理的推論とは異なる**非単調推論**（nonmonotonic reasoning）であるため，論理プログラムと述語論理の間に推論規則のうえで乖離が生まれる．

2･2　標準論理プログラム

確定論理プログラムから否定の推論を行う論理が整うと，こうして得られる否定情報をプログラム中に知識として記述し推論の過程で利用したい．実際，PROLOG にも節のボディにアトムの否定を条件として記述するための組み込み述語が用意されていたし，演繹データベースで二つの関係 R と S の間の差を計算する場合には

$$difference_{R,S}(x_1, \ldots, x_n) \leftarrow R(x_1, \ldots, x_n), not\ S(x_1, \ldots, x_n)$$

のようにボディに否定条件を記述する必要がある．一方，CWA や NAF で得られる否定はデフォルト推論の結果得られる**デフォルトの否定**（default negation）であり，述語論理で定義される否定とは意味が異なる．したがって，デフォルトの否定をプログラム中に陽に記述する場合，プログラムは述語論理のもとで定義される節集合とは異なるものになる．ここで，記述式の上でも論理プログラムと述語論理の間に乖離が生じることになる．そこで，プログラムの記述文の条件部にデフォルトの否定が出現する式

$$A \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n \qquad (2)$$

（A, A_1, …, A_n はアトム）をルールと呼び，節と区別する．ここで，*not* は NAF **オペレータ**と呼ばれ*4，(2) の形式のルールから構成される集合を**標準論理プログラム**（normal logic program）と呼ぶ．プログラムの構文の拡張により，デフォルトの否定を含むプログラムの意味論を考える必要があるが，すべての基礎アトムからなる集合を At とすると，解釈 $I \subseteq At$ の下で基礎アトム A が真でない場合，またそのときに限り I の下で $not\ A$ は真であると定義するのが自然である．そこで，(2) の形式の基礎ルール（変数を含まないルール）に対して，解釈 I が「$\{A_1, \ldots, A_m\} \subseteq I$ かつ $\{A_{m+1}, \ldots, A_n\} \cap I = \emptyset$ ならば $A \in I$」（*）という条件を満たす場合に I はこのルールを**充足する**と定義する．条件（*）は「$\{A_1, \ldots, A_m\} \not\subseteq I$ または $\{A_{m+1}, \ldots, A_n\} \cap I \neq \emptyset$ または $A \in I$」（†）と等価である．この結果，標準論理プログラム P のモデルは P を**基礎化**（ground instantiation）*5 したプログラムに含まれるすべてのルールを充足する解釈であると定義される．

確定論理プログラムの宣言的意味は最小モデルにより定義されたが，標準論理プログラムでは一般に最小モデルが存在しない．例えば，プログラム*6

$$p \leftarrow not\ q$$

は二つの**極小モデル**（minimal model）$\{p\}$ と $\{q\}$ をもつが，いずれも最小モデルではない．そこで標準論理プログラムの宣言的意味を最小モデルの代わりに極小モデルを使って定義し，一般に複数存在する極小モデルのうち，プログラムの記述文の意味を反映するものはどれかということを考える．上の例では p は q が証明されないという条件の下で導かれている．一方，q を導くルールはプログラム中には存在しない．あるアトムを導く根拠となるルールがプログラム中に存在するとき，そのアトムは**支持されている**（supported）という．プログラムの意味を反映する極小モデルに含まれるすべてのアトムは，プログラム中で支持されていることが必要条件になる．支持されたアトムから構成されるモデルを計算するためには，ヘッドに含まれるアトムがボディに存在するデフォルトの否定に含まれるアトムよりも上位の層にあると考え，解釈における各アトムの真偽の評価は下位の層に属するものから順番に行えばよい．上の例ではまず q が証明されないので偽と解釈され，その前提の下で p が真となる．この結果，二つの極小モデルのうち $\{p\}$ のみが支持されたアトムから構成され，プログラムの意味を反映したものと考えられる．この $\{p\}$ を**完全モデル**（perfect model）[Przymusinski 88] と呼ぶ．一般にはプログラム中の述語を階層に分割し，プログラム中の任意のルール（2）について，ヘッド A に含まれる述語がデフォルトの否定中に現れるアトム A_{m+1}, …, A_n に含まれる述語よりも上位に含まれるような分割が可能であるようなプログラムを**層状プログラム**（stratified program）と呼ぶ [Apt 88]. 層状プログラムは常に唯一の完全モデルをもち，プログラムがデフォルトの否定を含まない場合は完全モデルは最小モデルに一致する．層状プログラムは各述語が自

*4 *not* は歴史的に NAF という名前で呼ばれているが，証明の有限失敗による否定という手続的な意味はなく，以下で述べるような解釈の下での真偽による宣言的な意味づけがされる．

*5 ルール（2）に出現する変数を自由変数とみなし，プログラム中のすべての自由変数に対して任意の基礎項であらゆる可能な置換えを行ったもの．

*6 プログラムはルールの集合であるが，本稿ではプログラム中のルールを列挙したものをプログラムと同一視することがある．

身のデフォルトの否定を再帰的に呼び出さないようなプログラムであり，完備化が矛盾しない特徴をもつ．

標準論理プログラムにおける証明手続きとしては，SLD導出にNAFを組み込んだ**SLDNF導出** [Lloyd 87] がある．SLDNF導出は，層状プログラムの完全モデル意味論の下で健全であるが完全ではない．例えば，プログラム

$p \leftarrow not\ q$
$q \leftarrow q$

は完全モデル $\{p\}$ をもつが，ゴール $\leftarrow p$ はSLDNF導出に成功しない[*7]．なお，プログラムの完備化が矛盾しない条件の下で，層状プログラムの構文の制約を緩和する提案もなされ，述語階層をアトムの階層に拡張した**局所層状プログラム**（locally stratified program）[Przymusinski 88] やデフォルトの否定の偶数回の再帰呼出しを許容する**呼出し無矛盾プログラム**（call-consistent program）[Sato 90] などがある．

層状プログラムはデフォルトの否定の記述を可能にし，宣言的意味および手続的意味が明確に定義されたプログラムのクラスである．一方，我々が日常もっている知識や情報の中には，必ずしも層状の否定で表現できないものもある．例えば，目覚めのためにコーヒーがなければ紅茶を飲み，紅茶がなければコーヒーを飲むという状況を以下のようなプログラム P で記述したとしよう．

$wake \leftarrow coffee$
$wake \leftarrow tea$
$tea \leftarrow not\ coffee$
$coffee \leftarrow not\ tea$

このプログラムは，*tea* と *coffee* がそれぞれ自身のデフォルトの否定を再帰的に呼び出しているため層状プログラムではない．このような層状でないプログラムの宣言的意味を定義するために導入されたのが，**安定モデル意味論**（stable model semantics）[Gelfond 88] である．いま，解釈 I が与えられたとき，基礎化された標準論理プログラム P に含まれる任意の基礎ルール（2）に対して，(i) $\{A_{m+1}, \ldots, A_n\} \cap I \neq \emptyset$ であるようなルールは前頁の定義（†）から I により充足されるので P から消去し，(ii) 次に残ったルールに含まれるデフォルトの否定については $\{A_{m+1}, \ldots, A_n\} \cap I = \emptyset$ が成り立つことから，$not\ A_{m+1}, \ldots, not\ A_n$ は I により満たされるのでこれらの否定条件をルールから消去する．この結果，得られたプログラムを P^I で表すと，P^I はデフォルトの否定を含まないホーン節からなる論理プログラムと同一視できる．ここで，I が P^I の最小モデルと一致するとき，I を P の**安定モデル**と呼ぶ．安定モデルの直感的な意味は，ある解釈 I に対してプログラム P からデフォルトの否定条件が満たされるようなルールの集合 P^I を構成し，ここで P^I から導かれる事実の集合が I に一致するとき，I は P の安定なモデルであると考える．上の例では $I_1 = \{wake, tea\}$ としたとき，P^{I_1} は

$wake \leftarrow coffee$
$wake \leftarrow tea$
$tea \leftarrow$

となる．ここで，I_1 は P^{I_1} の最小モデルになっているので，I_1 は P の安定モデルである．同様に $I_2 = \{wake, coffee\}$ も P の安定モデルであることがわかる．安定モデルは極小モデルであるが，その逆はいえない．また層状プログラムにおいては安定モデルは完全モデルと一致する．安定モデルは完全モデルを自然に拡張し，宣言的意味が定義されるプログラムのクラスを拡張した．一方，層状でない標準論理プログラムは複数の安定モデルをもつことがあり，また一つももたないこともある．この事実は論理プログラムの意味論に**非決定性**を導入するばかりでなく，SLD導出以来使われてきたゴール駆動型の計算手続きに限界をもたらした．例えば，以下のプログラム

$p \leftarrow not\ q$
$r \leftarrow not\ r$

においてゴール $\leftarrow p$ はSLDNF導出に成功するが，このプログラムは安定モデルをもたない．すなわち，ゴールが成功してもそのゴールに含まれるアトムが真となる安定モデルの存在が保証されないため健全性がいえない．確定論理プログラムや層状プログラムでは，ゴール $\leftarrow G$ がプログラムの一部のルールを使って反駁されれば，G はプログラム全体の帰結であることが保証されるが，標準論理プログラムにおける安定モデル意味論の下ではこの**局所性の原則**（locality principle）が成り立たないのである．

標準論理プログラムの意味論として，完全モデルを拡張した別のアプローチとして**整礎モデル意味論**（well-founded model semantics）[van Gelder 91] がある．詳しい定義は省略するが[*8]，整礎モデル意味論は**3値論理**のもとで定義され，すべての標準論理プログラムは唯一の**整礎モデル**をもつことが保証される．また，安定モデル意味論と異なり局所性の原則を保持するため，従来方式のゴール駆動型証明手続きが有効であるというメリットがある．一方，安定モデル意味論のもとで真または偽と解釈される基礎アトムが整礎モデルのもとでは未定義（unknown）とされることがある．例えば，上のコーヒーと紅茶のプログラムではどちらを飲んでも目覚めることが安定モデルのもとではいえるが，整礎モデルの下では *wake, coffee, tea* はすべてunknown値を取る．このように整礎モデル意味論の下では本来，真偽が決まるはずの基礎アトムがunknown値を取ることがあるため，安定モデルに比べて**慎重**（skeptical）であるといわれる（逆に安定モデルは**安易**（credulous）であるといわれる）．このように安定モデル意味論，整礎モデル意味論ともに一長一短があり，1990年代はこれらの問題点を解決するための意味論が数

*7 完全モデル意味論の下で健全かつ完全な証明手続きも提案されている [Przymusinski 89].

*8 [勝野 90] に詳しい定義が解説されている．

多く提案されたが，その多くが構文的に特殊なプログラムに直感的な意味を与えるために悪戯に理論を複雑化したもので，今日ではほとんど使われていない．

2･3　標準選言論理プログラム

確定論理プログラムにおける記述文は，帰結部が単一のアトムからなる確定節でなければならないが，この事実は一般に不確定な情報を記述する上での制約になる．そこで，ルールのヘッド部にアトムの選言を含むような非ホーン節からなる**選言論理プログラム**（disjunctive logic program）の枠組みが導入された [Minker 82]．選言論理プログラムは以下の形式の節

$$A_1 \vee \cdots \vee A_l \leftarrow A_{l+1}, \ldots, A_m \qquad (3)$$

（$A_1, \ldots, A_m$ はアトム）からなる集合である．このとき，確定論理プログラムは節（3）において $l=1$ であるような節のみから構成される特殊な場合であると考えられる．選言論理プログラムは一般に最小モデルはもたないため，プログラムがもつ複数の極小モデルによって宣言的意味が定義される．例えば

$$p \vee q \leftarrow$$

という選言節からなるプログラムは二つの極小モデル $\{p\}$ と $\{q\}$ をもつ．これらの極小モデルは同等であり，プログラムの二つの可能な解釈を表現している．一方，選言論理プログラムに CWA を適用すると，CWA から得られる否定情報とプログラムが矛盾することがある．上のプログラムでは p と q はいずれもプログラムから証明されないため CWA により $\neg p$ と $\neg q$ が推論されるが，これらの否定事実はプログラムの選言節と矛盾する．こうした問題を解決するために，[Minker 82] は CWA を拡張した**一般閉世界仮説**（generalized CWA: GCWA）を導入した．GCWA は基礎アトム A が真となるような極小モデルが存在しない場合に限り，A を偽と解釈するものである．上の例では p または q を含む極小モデルが存在するため，GCWA のもとではこれらの否定は推論されない．一方，GCWA による否定推論の問題として，例えば以下のプログラム[*9]

$$land\_animal \vee aquatic\_animal \leftarrow$$
$$amphibian \leftarrow land\_animal, aquatic\_animal$$

は二つの極小モデル $\{ land\_animal \}$ と $\{ aquatic\_animal \}$ をもつが，*amphibian* を含む極小モデルが存在しないため，GCWA のもとでは $\neg$ *amphibian* が推論される．このように極小モデルのもとで定義された GCWA は選言を**排他的**（exclusive）に解釈し，選言に含まれる複数のアトムが同時に真となる**包含的**（inclusive）な解釈が排除されることがある．古典論理における選言は本来，包含的な解釈も含むため，この意味で GCWA による推論結果は強すぎるといえる．そこで GCWA を弱めて選言の包含的な解釈のもとで否定推論を行う方式として WGCWA（weak GCWA）[Rajasekar 89] または DDR（disjunctive database rule）[Ross 88] が提案された．WGCWA ではプログラムから導出される，基礎アトムからなる選言の集合に含まれないような基礎アトムの否定を推論する．上の例では選言 *amphibian*$\vee$*land_animal* $\vee$ *aquatic_animal* がプログラムから導出されるため，ここに含まれる *amphibian* の否定は推論されない．

WGCWA や DDR は GCWA を緩和した否定推論を実現するが，選言の包含的解釈に基づくプログラムのモデル論は提供しない．また，プログラムが排他的な選言と包含的な選言を同時に含む場合は GCWA も WGCWA（DDR）も対処できない．そこで排他的な選言と包含的な選言を区別し，2 種類の異なる選言を含むプログラムの宣言的意味論として**可能モデル意味論**（possible model semantics）[Sakama 90] が導入された．上の例ではプログラムは三つの可能モデル $\{ land\_animal \}$, $\{ aquatic\_animal \}$, $\{ land\_animal, aquatic\_animal, amphibian \}$ をもつ．ここで最初の二つは極小モデルであるが，三つ目は非極小なモデルになっている．可能モデル意味論のもとでは排他的な選言と包含的な選言はプログラム中に記述した負節によって区別される．例えば，カレーライスとラーメンと野菜サラダの三つのメニューがあり，このうちカレーライスとラーメンを同時に食べるのはカロリ過多のため不可という場合，

$$curry \vee noodles \vee salad \leftarrow$$
$$\leftarrow curry, noodles$$

と記述できる．このプログラムの可能モデルは $\{ curry \}$, $\{ noodles \}$, $\{ salad \}$, $\{ curry, salad \}$, $\{ noodle, salad \}$ の五つとなる．可能モデル意味論は極小モデル意味論と比べて選言の解釈が柔軟に記述できるという知識表現上のメリットがあるばかりでなく，計算量も一般に低く抑えられる [Chan 93]．確定論理プログラムでは可能モデルは最小モデルと一致する．

選言論理プログラムの記述文にデフォルトの否定を導入したものを**標準選言論理プログラム**（normal disjunctive logic program）と呼ぶ．標準選言論理プログラムは以下の形式のルール

$$A_1 \vee \cdots \vee A_l \leftarrow A_{l+1}, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n \qquad (4)$$

（$A_1, \ldots, A_n$ はアトム）の集合である．標準選言論理プログラムの意味論としては，安定モデルと整礎モデルが拡張された意味論が提案され，このうち安定モデル意味論を拡張したものが解集合意味論へと発展していくのである．また可能モデル意味論を標準選言論理プログラムに拡張した意味論 [Sakama 94] が，後に解集合プログラミングの意味論と関係をもつことがわかる（4･2 節）．選言論理プログラムはルールのヘッドが選言を含むため，ゴール駆動によるトップダウン型の証明手続きは一般に複雑になる．そこで選言論理プログラムの極小モデル全体を**モデル生成手続き**によりボトムアップ的に計算する方法が提案され [Manthey 88]，この方法は後に一

＊9 amphibian: 両生類

般選言論理プログラムの安定モデルの計算および解集合の計算手続きに発展した [Inoue 92]．極小モデルや安定モデルのボトムアップ計算は，不動点計算によるプログラムの操作的意味論として形式化されるが [Fernández 95, Inoue 96]，これは確定論理プログラムの不動点意味論の自然な拡張になっている．

3. 解集合意味論

3・1 拡張選言論理プログラム

標準選言論理プログラムにおける *not* はデフォルトの否定を意味したが，これに対して論理否定¬による**明示的否定**（explicit negation）をデフォルトの否定と区別して使うプログラムのクラスを**拡張選言論理プログラム**（extended disjunctive logic program）[Gelfond 91] と呼ぶ．拡張選言論理プログラムは以下の形式のルール

$$L_1 ; \cdots ; L_l \leftarrow L_{l+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \quad (5)$$

の集合である．式（4）と異なり $L_1, \ldots, L_n$ は正負のリテラルで，セミコロン（;）は選言を表す．ルールのヘッドで∨の代わりに“;”が使われている理由は，古典論理では論理式 $p \vee \neg p$ はトートロジーであるが，拡張選言論理プログラムではルール p; $\neg p \leftarrow$はトートロジーではないため両者を異なる記号で区別している[*10]．また古典論理では $p \leftarrow q$ と $\neg q \leftarrow \neg p$ は同値な論理式であるが，拡張選言論理プログラムでは両者は同値ではない．拡張選言論理プログラムのうち，ヘッドに選言を含まないようなルールから構成されるものを特に**拡張論理プログラム**（extended logic program）と呼ぶ．このように2種類の否定を区別して用いることにより，意味が微妙に異なる知識を表現することが可能になる．例えば，ルール

$$cross \leftarrow not\ car$$

は車が走ってくることが認識されなければ道路を渡るという行動を表現しているのに対して

$$cross \leftarrow \neg\ car$$

は車が走ってこないと認識したうえで道路を渡るという行動を表現している．両者を比べるとプログラムから *car* という肯定事実が証明できなければ *cross* を帰結するのが前者であるのに対し，プログラムから¬ *car* という否定事実が証明された後に *cross* を帰結するのが後者である．拡張選言論理プログラムでは明示的否定を使って否定事実がプログラムから直接証明されるため，証明されない事実は否定とみなす閉世界仮説（CWA）は用いられず，証明されない事実は肯定も否定もしない**開世界仮説**（open world assumption: OWA）の立場がとられる．CWAはプログラム中に記述された肯定情報が完全で，それ以外の情報を否定とみなす場合に有効であるが，肯定情報が不完全でCWAを適用したくない場合にはOWAが有効である．拡張選言論理プログラムでは，CWAは必要に応じてプログラム中に

$$\neg\ p \leftarrow not\ p$$

のように宣言的に記述される．これと対象的にルール

$$p \leftarrow not\ \neg\ p$$

は，ある事実 p の明示的否定が証明されない限り p が成り立つとみなす**デフォルトの肯定**を表現している．

拡張選言論理プログラムの宣言的意味は**解集合意味論**（answer set semantics）[Gelfond 91] によって与えられるが，その定義は安定モデル意味論を自然に拡張したものである．まず，P をデフォルトの否定を含まない基礎化されたプログラムとする．このとき基礎リテラルの集合 S が，P に含まれる任意のルール

$$L_1 ; \cdots ; L_l \leftarrow L_{l+1}, \ldots, L_m \quad (6)$$

について，(i) $\{L_{l+1}, \ldots, L_m\} \subseteq S$ ならば $\{L_1, \ldots, L_l\} \cap S \neq \emptyset$，(ii) S が相反する正負のリテラル L と $\neg L$ を同時に含むならば $S = Lit$（ここで，Lit はすべての基礎リテラルからなる集合）の2条件を満たすような極小の集合であるとき，S を P の**解集合**（answer set）と呼ぶ．次に P をデフォルトの否定を含む任意の基礎化されたプログラムとし，基礎リテラルの集合 S が P に含まれる（5）の形式の任意の基礎ルールについて，$\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$ が成り立つとき，またそのときに限り，基礎ルール（6）がプログラム P^S に含まれるとする．このとき，S が P^S の解集合であるならば，S は P の**解集合**であると定義する．上で P から P^S を構成する変換は，標準論理プログラムで P から P^I を構成する変換と全く同じである．したがって，プログラムが負リテラルを含まない場合は，拡張選言論理プログラムの解集合は標準選言論理プログラムの安定モデルと一致する．安定モデルとの唯一の違いは，解集合 S が相反する正負のリテラル L と¬ L を同時に含む場合に $S = Lit$ と定義されることである[*11]．拡張選言論理プログラムは一般に複数の解集合をもち，解集合をもたないこともある．拡張選言論理プログラムの解集合は基礎リテラルの極小集合であり，Lit 以外の解集合をもつプログラムは**無矛盾である**という．

拡張選言論理プログラムでは否定知識が陽に記述されるため，プログラムが矛盾する可能性がある．そこで矛盾を回避する方法として，相反する帰結を導くルールの間に優先関係を導入するもの [Kowalski 90], プログラムに含まれるルールの無矛盾な極大部分集合の解集合を計算する方式 [Inoue 94], プログラム中の矛盾を局所化する**矛盾許容論理**（paraconsistent logic）に基づく形式化 [Sakama 95] などがある．

3・2 一般拡張選言論理プログラム

拡張選言論理プログラムに含まれるルールのヘッド

*10 [Gelfond 91] では“;”の代わりに“|”が用いられている．“or”を用いている文献 [Baral 03, Gelfond 08] もある．

*11 *Lit* はあまり意味がないことから，解集合を無矛盾な基礎リテラルの集合とする定義もある [Gelfond 08, Lifschitz 02].

に，デフォルトの否定の出現を許容するプログラムのクラスを**一般拡張選言論理プログラム**（general extended disjunctive logic program）と呼ぶ．一般拡張選言論理プログラムは以下の形式のルール

$$L_1; \ldots; L_k; not\ L_{k+1}; \ldots; not\ L_l \leftarrow L_{l+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \quad (7)$$

（$L_1, \ldots, L_n$ は正負のリテラル）の集合である．一般拡張選言論理プログラムは [Lifschitz 92] によって導入され，[Inoue 98] によって知識表現への応用が示された．一般拡張選言論理プログラムの意味論は，拡張選言論理プログラムの解集合を拡張することによって定義される*12．一般拡張選言論理プログラムが拡張選言論理プログラムと異なる点は，ルール（7）のヘッドに出現するデフォルトの否定の効果により解集合が極小とは限らない点である．例えば，プログラム

$$p; not\ p \leftarrow \quad (8)$$

は二つの解集合 $\emptyset$ と $\{p\}$ をもつ．ルール（8）は解集合がアトム p を含むか含まないかという 2 通りの可能性を宣言的に記述しており，この形式のルールは**アブダクティブ論理プログラミング**（abductive logic programming）[Kakas 92] における候補仮説の表現に使えることがわかっている [Inoue 98]．また，一般拡張選言論理プログラムの解集合の非極小性を使うと，標準選言論理プログラムの可能モデルを一般拡張選言論理プログラムの解集合と対応させるプログラム変換も可能である [Inoue 98]．

3・3　順序付き拡張選言論理プログラム

拡張選言論理プログラムは一般に複数の解集合をもつが，本節では解集合の間に優先順位を導入する枠組みとして，**順序付き拡張選言論理プログラム**（logic programs with ordered disjunction: LPOD）[Brewka 02] を紹介する．LPOD は以下の形式のルール

$$L_1 \times \cdots \times L_l \leftarrow L_{l+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \quad (9)$$

（$L_1, \ldots, L_n$ は正負のリテラル）の集合で，×は**順序付き選言**（ordered disjunction）と呼ばれる．ルール（9）における $L_1 \times \cdots \times L_l$ の意味は，L_1 が成り立てば L_1 を帰結し，L_1 が成り立たなければ L_2 を帰結し，L_1, L_2 がともに成り立たなければ L_3 を帰結し，…といった具合である．つまり，LPOD のルールにおいてはヘッドに含まれるリテラルがランク付けされ，ボディの条件が成り立つ場合に，ヘッドの帰結の L_i が L_j $(i > j)$ より優先される．LPOD の解集合は以下のように定義される．いま，r を（9）の形式の基礎ルールとした場合，$k \leqq 1$ について以下のルール

$$L_k \leftarrow L_{l+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n, not\ L_1, \ldots, not\ L_{k-1}$$

を r の k 番目の選択肢と呼び r^k で表す．P を LPOD の基礎プログラムとするとき，P に含まれる各ルールをそれぞれの一つの選択肢で置き換えたプログラム P' を P の**分割プログラム**（split program）と呼ぶ．基礎リテラルの集合 S が，ある分割プログラム P' の無矛盾な解集合であるとき，S を P の**解集合**と呼ぶ．次に S を LPOD P の解集合とし，r を（9）の形式のルールとするとき，S における r の**等級**（degree）を以下で定義する．

（i）$\{L_{l+1}, \ldots, L_m\} \not\subseteq S$ または $\{L_{m+1}, \ldots, L_n\} \cap S \neq \emptyset$ のとき，S は等級 1 で r を満たす．

（ii）$\{L_{l+1}, \ldots, L_m\} \subseteq S$ かつ $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$ のとき，S は等級 j $(1 \leqq j \leqq l)$ で r を満たす．ここで，$j = \min\{k \mid L_k \in S\}$

S における r の等級を $deg_S(r)$ で表す．S がプログラム P の解集合のとき，S は P のすべてのルールをある等級で満たすことがいえ，等級が上がれば S によって満たされるそのルールの選言の優先度は下がる．この等級の概念を使って，P の解集合の間の優先関係を定義する．いま，リテラルの集合 S に対して，$S^i(P) = \{r \in P \mid deg_S(r) = i\}$ とする．LPOD P の解集合 S_1, S_2 に対して，$S_2^i(P) \subset S_1^i(P)$ が成り立ち，任意の $j < i$ について $S_1^j(P) = S_2^j(P)$ のとき，S_1 は S_2 より**優先される**（preferred）といい，$S_1 > S_2$ と書く．LPOD P の解集合 S について，$S' > S$ を満たすような P の解集合 S' が存在しないとき，S を**選好的解集合**（preferred answer set）と呼ぶ．

LPOD を使った知識表現の例を一つ示しておこう．以下のプログラム P_0 が与えられたとする．

（r_1）$cinema \times beach \leftarrow not\ hot$

（r_2）$beach \times cinema \leftarrow hot$

（r_3）$hot \leftarrow summer, not\ \neg hot$

（r_4）$\neg beach \leftarrow rain$

それぞれのルールの意味は，（r_1）暑くなければ海辺に行くより映画に行くほうを好み，（r_2）暑ければその逆で，（r_3）夏は通常暑く，（r_4）雨が降れば海辺には行かない．このとき，P_0 は単一の選好的解集合

$$S_1 = \{cinema\}$$

をもつ．ここで，夏という事実（$summer$）が P_0 に加えられると，$P_1 = P_0 \cup \{summer \leftarrow\}$ の選好的解集合は

$$S_2 = \{summer, hot, beach\}$$

となる．次に暑くない（$\neg hot$）という情報が加わると，$P_2 = P_1 \cup \{\neg hot \leftarrow\}$ の選好的解集合は

$$S_3 = \{summer, \neg hot, cinema\}$$

となる．最後に夏で雨が降っている場合，$P_3 = P_1 \cup \{rain \leftarrow\}$ の選好的解集合は

$$S_4 = \{summer, rain, hot, \neg beach, cinema\}$$

となる．このように，LPOD は状況によって変化する解集合の選好を表現することができる．

解集合の間に優先関係を導入する枠組みとしては LPOD のほかに，拡張論理プログラムにおいてルール間の優先関係を指定する方法 [Brewka 99]，一般拡張選言論理プログラムにおいてリテラルとデフォルト否定されたリテラルの間の優先関係を記述する方法 [Sakama 00] などがある．

*12　詳しい定義は [井上 08] に解説されている．

4. 解集合プログラミング

4・1 ASPと制約プログラミング

論理プログラムによって記述された問題を，解集合意味論の下で解決する方法を**解集合プログラミング**（answer set programming: ASP）と呼ぶ [Lifschitz 08]*13．ASPではプログラムは問題の解が満たすべき制約式の集合であると考えられ，プログラムの解集合が問題の解となる．したがって，ASPでは与えられたプログラムの解集合を計算する手続きが必要になり，このような手続きはASP**ソルバ**と呼ばれる [Leone 06, Simons 02]．典型的なASPソルバは，(i) 変数を含むプログラムを基礎化し，(ii) 基礎化されたプログラムの解集合を計算するという2段階からなる*14．ASPソルバの問題点の一つは，上記 (i) を有限時間で終了するためには基礎項全体の集合が有限でなければならず，関数記号が一般に扱えないという構文上の制約があることである．論理プログラミング言語PROLOGでは関数記号は高階オブジェクトを項として表現し，関数記号の一種であるリスト記号 [・|・] は複雑なオブジェクトを再帰的に定義するうえでも有用であった．一方，ASPソルバでは関数記号の使用が制限されるため，問題を記述するうえで従来のプログラミング方法を変える必要が生じる．そこで，ASPソルバではオブジェクトが満たすべき制約式をプログラム中に記述し，項や再帰的定義を用いて表現していたオブジェクトを解集合として表現する方法をとる．これらの知識表現の違いを例題 [Marek 99] を使って見てみよう．

いま，与えられた無向グラフのクリーク*15を求める問題を以下の確定論理プログラムで記述したとする．

$$clique([\,])$$
$$clique([x]) \leftarrow node(x)$$
$$clique([y|x]) \leftarrow clique(x), allconnected(y,x)$$

ここで，$allconnected(y,x)$ はノード y がリスト x に含まれず，かつ x に含まれるすべてのノードとエッジでつながっていて x の要素に重複がない場合に真になるものとする．いまグラフ G に含まれるノードとエッジの事実が

$$F = \{\, node(a), node(b), node(c), edge(a,b), edge(b,a), edge(b,c), edge(c,b)\,\}$$

として与えられたとき，上のプログラムは G に含まれるクリークを述語 *clique* の引数に含まれるリスト

$clique([\,]), clique([a]), clique([b]), clique([c]), clique([a,b]), clique([b,a]), clique([b,c]), clique([c,b])$

として計算し，これらの解はプログラムの唯一の最小モデルに含まれる．

一方，ASPでは同じ問題が以下のように記述される．

$$clique(x) \leftarrow node(x), not\ \neg clique(x)$$
$$\neg clique(x) \leftarrow node(x), not\ clique(x)$$
$$\leftarrow clique(x), clique(y), x \neq y, not\ edge(x,y)$$

上のプログラムの最初の二つのルールはノードをクリークに含まれるもの（*clique*）とそうでないもの（$\neg$*clique*）に分類する．3番目のルールはクリークに含まれるノードが満たすべき条件をヘッドが空の**制約**（constraint）として記述している．この制約の意味は，*clique* に含まれる二つの異なるノード x と y については $edge(x,y)$ が成り立たなければいけないことを言明している．この結果，クリークを構成するノードは，解集合に含まれる述語 *clique* の引数として表明され，事実 F が与えられたとき，それぞれの解はプログラムの六つの異なる解集合

$\{\neg clique(a), \neg clique(b), \neg clique(c)\} \cup F$,
$\{clique(a), \neg clique(b), \neg clique(c)\} \cup F$,
$\{\neg clique(a), clique(b), \neg clique(c)\} \cup F$,
$\{\neg clique(a), \neg clique(b), clique(c)\} \cup F$,
$\{clique(a), clique(b), \neg clique(c)\} \cup F$,
$\{\neg clique(a), clique(b), clique(c)\} \cup F$

として計算される．

上例で見たように，ASPではヘッドが空の制約をプログラム中に記述することによって，求めるべき解集合を計算する．一般的には，制約

$$\leftarrow L \qquad (10)$$

は L であってはならないという禁止を表現し，

$$\leftarrow not\ L \qquad (11)$$

は L でなければならないという強制を表現する*16．制約（10）によりリテラル L を含む集合は解候補から除かれ，制約（11）によりリテラル L を含まない集合は解候補から除かれる．ヘッドが空のルールは，論理プログラミングでは [Kowalski 74] によるプログラムの手続的解釈に従って問合せを表現するゴールとして用いられ，演繹データベースやアブダクティブ論理プログラミングにおける**一貫性制約**（integrity constraint）[Kakas 92, Sadri 88] として導入された経緯はあるが，プログラム中に記述されて推論の過程で用いられるようになったのはASPによる知識表現がポピュラーになってからである*17．しかし歴史的に見ると，[Sakai 84] はヘッドが空の負節を含むホーン論理プログラムによる知識表現

---

*13 解集合は安定モデルと本質的に同じものであることから，安定モデル意味論の下での問題解決方式も通常ASPと呼ばれる [Marek 99, Niemela 99]．ASPはまたAnsPrologと呼ばれることもある [Baral 03]．

*14 ASPソルバのより詳しい手続きについては [井上 08, 佐藤 08] を参照されたい．

*15 無向グラフ $G=(V,E)$ におけるノードの部分集合 $C \subseteq V$ のうち，C に属する任意の二つのノードを結ぶエッジが存在する場合，C をクリーク（clique）と呼ぶ．ここでは C が空集合の場合やノードを一つしか含まない場合もクリークとする．

*16 これらの制約はそれぞれ，$A \leftarrow L, not\ A$ あるいは $A \leftarrow not\ L, not\ A$（ここで A は新たに導入された基礎アトム）と記述しても同じ意味をもつ．

*17 実際，解集合意味論が提案された論文 [Gelfond 91] においてもヘッドが空のルールの記述は見られない．

と計算手続きを導入し，[Sakama 90] は排他的選言を包含的選言と区別して表現するためにプログラム中に負節を導入し推論に用いている．また，[Inoue 92, Manthey 88] もプログラム中に負節を記述し，選言論理プログラムからのモデル生成手続きにおいて不要な解釈の枝刈りに用いている．これらの研究は，推論過程で制約を用いるという視点では ASP に先駆けていたといえるだろう．ASP ソルバは関数記号を用いた再帰的定義の代わりに，制約による記述式を使って問題解決を行っていることから，ASP を**制約プログラミング**と捉える見方もある [Niemelä 99]．最近の研究では，関数記号を含むプログラムの安定モデルや解集合の計算方法も提案されている [Bonatti 04].

計算量の観点からは，有限の基礎ルールから構成される標準論理プログラムが安定モデルをもつかどうかの決定問題は NP **完全**である [Marek 91]．このことから NP のクラスに属する決定問題は原理的に変数を含まない標準論理プログラムで記述でき，決定問題の解の存在はプログラムの安定モデルの存在判定に帰着させることができる．一方，有限の基礎ルールから構成される標準選言論理プログラムが安定モデルをもつかどうかの決定問題は $\sum_P^2$ **完全**である [Eiter 95]．このことから，標準選言論理プログラムは標準論理プログラムよりも**多項式時間階層**（polynomial hierarchy）において一段階上位のクラスに所属する言語クラスであり，階層の包含関係が真である限り，標準選言論理プログラムは標準論理プログラムよりも一般に表現能力は高いと考えられる*18.

4･2　ASP の言語拡張

ASP ソルバが開発され ASP がプログラミング言語として問題解決に利用されるようになると，プログラムをコンパクトに記述し，問題を効率的に処理するための ASP 言語の構文の拡張が行われるようになった．以下では，現在 ASP で採用されている代表的な言語拡張について解説する．SMODELS[Simons 02] は標準論理プログラムの安定モデルを計算するシステムであるが，その構文では**個数制約**（cardinarity constraint）と呼ばれる以下の式が採用されている．

$$U\,\{A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\}\,V \qquad (12)$$

ここで，各 A_i はアトムで式（12）の意味は，A_1, …, A_m *not* A_{m+1}, …, *not* A_n のうちモデルによって充足されるものの個数が U 以上 V 以下という意味である．ここで，上限あるいは下限が存在しない場合は，それぞれ V あるいは U が省略されることもある．この個数制約をルールを記述するアトムの代わりに用いる．例えば，ルール

$$1\{p, q\} \leftarrow 2\{r, s, \mathit{not}\ t\}$$

はボディに含まれる r, s, *not* t のうち二つ以上が成り立てば，ヘッドに含まれる p, q のうち少なくとも一つが成り立つという意味になる．またアトムが変数を含む場合は，変数に対する**条件づけ**を行う．例えば，

$$\{\mathit{clique}(x) : \mathit{node}(x)\} \leftarrow$$

はクリーク *clique* に含まれる変数 x は *node*(x) であることを記述している．このとき，上式に制約

$$\leftarrow \mathit{clique}(x), \mathit{clique}(y), x \neq y, \mathit{not}\ \mathit{edge}(x, y)$$

を加えた二つのルールから構成されるプログラムは，*node* と *edge* の事実が与えられたときにクリークを計算する．このプログラムを 4･1 節で述べたクリーク計算のプログラムと比較すると，記述文がコンパクトになっているのがわかる．個数制約（12）はさらに**重み付き制約**（weighted constraint）と呼ばれる以下の制約式

$$U\,\{A_1 = w_1, \ldots, A_m = w_m, \mathit{not}\ A_{m+1} = w_{m+1}, \ldots, \mathit{not}\ A_n = w_n\}\,V \qquad (13)$$

に拡張される．ここで，各 w_i はアトム A_i あるいはデフォルト否定 *not* A_i の重みを表す．式（13）の意味は，A_1, …, A_m, *not* A_{m+1},…, *not* A_n のうち解集合によって充足されるものの重みの合計が U 以上 V 以下という意味である．個体制約（12）は重み付き制約（13）において $w_1 = \cdots = w_n = 1$ である特殊な場合と考えることができる．重み付き制約を使うと標準論理プログラムのルール

$$A \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$$

は，

$$1\{A = 1\} \leftarrow m + n\,\{A_1 = 1, \ldots, A_m = 1, \mathit{not}\ A_{m+1} = 1, \ldots, \mathit{not}\ A_n = 1\}$$

と解釈される．一般に以下の形式のルール

$$C \leftarrow C_1, \ldots, C_m, \mathit{not}\ C_{m+1}, \ldots, \mathit{not}\ C_n \qquad (14)$$

（C, C_1, …, C_n は重み付き制約）から構成されるプログラムを**重み付きプログラム**（weighted program）と呼ぶ．重み付き制約の中で特に重要なのは，**重み付きルール**（weighted rule）と呼ばれる以下の形式のルール

$$A \leftarrow w\,\{A_1 = w_1, \ldots, A_m = w_m, \mathit{not}\ A_{m+1} = w_{m+1}, \ldots, \mathit{not}\ A_n = w_n\} \qquad (15)$$

および**選択ルール**（choice rule）と呼ばれる以下の形式のルール

$$\{A_1, \ldots, A_l\} \leftarrow A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \qquad (16)$$

である．重み付きルールは重み付き制約からなる条件部の下限を指定しており，選択ルールはボディの条件が成り立つときにヘッドのアトム集合の任意の部分集合が選択されることを表現している．ルール（15）と（16）は**基礎制約**（basic constraint）と呼ばれ，重み付きプログラムのルール（14）はこれら 2 種類の基礎制約に変換可能である [Simons 02]．なお個体制約や重み付き制約から構成されるルールは，適当なアトムを新たに導入することで通常の標準論理プログラムのルールに書き換えることができる [Baral 03].

---

*18 解集合は安定モデルと本質的に同じため，この計算量の結果は拡張選言論理プログラムと拡張論理プログラムにもそのまま適用される．一方，一般拡張選言論理プログラムは拡張選言論理プログラムと同じ計算量クラスに属し，前者から後者への多項式時間変換が可能である [Inoue 98].

重み付きプログラムの宣言的意味は，安定モデルを拡張することによって与えられる．しかし，標準論理プログラムの安定モデルと異なり，重み付きプログラムの安定モデルは一般に非極小になる．例えば，選択ルール

$$\{p\} \leftarrow \qquad (17)$$

は二つの安定モデル $\{\}$, $\{p\}$ をもつが，このうち $\{p\}$ は極小集合ではない．3·2節で一般拡張選言論理プログラムは非極小な解集合をもつことを示したが，ルール（17）は一般拡張選言論理プログラムのルール

$$p\,;\, not\ p \leftarrow$$

と同じ意味をもつ．重み付きプログラムは標準論理プログラムのアトムを重み付き制約で置き換えたものであるが，その構文は標準選言論理プログラムを含む．すなわち，標準選言論理プログラムのルール

$$A_1 \vee \cdots \vee A_l \leftarrow A_{l+1}, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$$

は個数制約を用いた以下のルール

$$1\{A_1, \ldots, A_l\} \leftarrow m+n\,\{A_{l+1}, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\} \qquad (18)$$

と等価である．このとき，ルール（18）から構成されるプログラムの安定モデルは標準選言論理プログラムの可能モデルと一致する [Marek 07].

拡張選言論理プログラムの解集合を計算するDLV[Leone 06] では，算術計算のための**アグリゲート関数**（aggregate function）が導入されている [Dell'Armi 03]．例として，パーティーの客が着席場所を決める問題を考えよう．各テーブルは N 人着席可能であり，友人同士は同じテーブルに着くとする．いま，$guest(x)$, $table(x)$, $at(x, y)$ をそれぞれ客，テーブル，着席場所を表すとすると問題は以下のように表現される．

$$at(x,y)\ ;\ \neg\ at(x,y)\ \leftarrow guest(x), table(y)$$
$$\leftarrow friend(x,y), at(x,z), not\ at(y,z)$$
$$\leftarrow table(y), \#count\{x: at(x,y)\} > N$$
$$\leftarrow guest(x), not\ \#count\{y: at(x,y)\} = 1$$

上のプログラムで1番目のルールはそれぞれの客があるテーブルに着席するかしないかを表し，2番目の制約は友人同士は同じテーブルに着席することを表す．3番目と4番目の制約に出現する $\#count\{x: c(x)\}$ は条件 c を満たす個体をカウントする関数で，3番目のルールはテーブルに着席する客の数は N を超えてはいけないことを表し，4番目のルールは客の着席場所が1か所である制約を表している．DLV ではこの他に $\#min$, $\#max$, $\#sum$, $\#times$ のアグリゲート関数が用意されている．

重み付き制約やアグリゲート関数は，さらに**抽象制約**（abstract constraint）[Marek 07, Son 06] という一般化された枠組みで統一的に扱われる．**抽象制約アトム**（abstract constraint atom）とは（D, C）の形式をした表現で，D はアトムの集合，C はアトムのべき集合で $C \subseteq 2^D$ を満たすものとする．抽象制約アトム（D, C）は領域 D における解候補の集合 C を表している．例えば，個数制約

$$1\{p, q\}1$$

は抽象制約アトムを使うと

$$(\{p, q\}, \{\{p\}, \{q\}\})$$

と表現される．上式は，$\{p, q\}$ から1個のアトムを選択する解は $\{p\}$ もしくは $\{q\}$ であることを表している．また，アグリゲート

$$\#count\{x: p(x)\} > 2$$

は変数の定義域を $\{a, b, c\}$ とした場合，抽象制約アトム

$$(\{p(a), p(b), p(c)\}, \{\{p(a), p(b), p(c)\}\})$$

と表現される．上式は $p(x)$ のインスタンスが2よりも大きくなるような解は $\{p(a), p(b), p(c)\}$ のみであることを表している．抽象制約アトムを使った表現では，アトム p は（$\{p\}$, $\{\{p\}\}$）に等しく，デフォルトの否定 $not\ p$ は

$$(\{p\}, \{\emptyset\})$$

のように表現される．[Marek 07, Son 06] では，このような抽象制約から構成されるプログラムの理論が展開され，解集合意味論が導入されている．

5. おわりに

論理プログラミングから解集合プログラミングに至るまでの発展の経緯と現状について概説した．ASP の特徴と意義をまとめると以下のようになる．

（a）**宣言的プログラミングの実現**

ASP は知識表現言語に必要とされる**宣言性**（declarativeness）を具備している．論理プログラミングの概念が導入された当初は，論理と制御の分離 [Kowalski 79] が最大の特徴の一つといわれたが，PROLOG ではプログラムの書き方に依存した実行手順や制御子（カット）の導入などにより，論理プログラミングの宣言性が犠牲にされた．これに対して ASP はその宣言性を保ちつつ，ASP ソルバ上で拡張言語が実装されている点が PROLOG とは異なる．

（b）**制約プログラミングとの融合**

ASP は従来の論理プログラミングの発展という見方もできる一方で，制約プログラミングの一種と捉えることができ，**制約充足問題**の計算言語としても有用である．論理プログラミングに制約の概念を組み込んだ枠組みとしては，**制約論理プログラミング**（constraint logic programming: CLP）[Jaffer 94] がある．ASP は CLP とは言語発展の経緯，構文論，意味論がいずれも異なるものの，思想的には両者は接近してきており，二つの枠組みを統合する試みもなされている [Elkabani 04, Mellarkod 08].

（c）**プログラミング理論の発展と開発環境の整備**

ASP は論理に基づく厳密な数学的意味論をもち，プログラムの性質や計算量に関する理論的研究が進んでいる [Baral 03, Gelfond 08]．一方，ASP ソルバによるプログラミング環境も整備されてきており，ASP の処理系はフリーウェアとして Web を通

じて全世界のユーザに公開されている*19.

ASPの最近の動向としては，4·2節で述べたようにプログラム中にさまざまな制約を記述する言語拡張が行われているほかに，セマンティックWebにおけるオントロジーを記述するためにASPと**記述論理**（description logic）を結合した言語 [Eiter 08] や，ASPと**確率推論**を融合する研究 [Baral 09]，ASPによる**マルチエージェントシステム**の理論 [De Vos 04, Sakama 08] なども研究されている．

確定論理プログラムからASPに至る30年余の発展の過程は，論理プログラミングの研究に従事する世界中の研究者による切磋琢磨の歴史でもある．こうした発展を可能にしているのは，論理プログラミングが論理という世界共通言語によって記述されているからであり，この事実は論理プログラミングの大きな長所の一つである．論理プログラミングはプログラミング言語としては実応用に使われることがこれまで少なかったが，その背景には処理速度が遅いという技術的な問題のほかに，1980～90年代は応用を顧みることなく理論研究が先行したという経緯がある．ASPに関しては理論研究と処理系の効率化が進められる一方で，さまざまな分野への応用が報告され始めており [佐藤 08]，その成否は論理プログラミングの今後の発展を占ううえで興味深い．なお，本分野の最新の研究成果はIJCAI, AAAI, ECAI, KRといった人工知能や知識表現に関する国際会議のほか，論理プログラミングに関する国際会議（ICLP），論理プログラミングと非単調推論に関する国際会議（LPNMR）などで報告されている．

◇ 参 考 文 献 ◇

[Apt 88] Apt, K. R., Blair, H. A. and Walker, A.: Towards a theory of declarative knowledge, In: Minker J.（ed.）, *Foundations of Deductive Databases and Logic Programming*, pp. 89-148, Morgan Kaufmann（1988）

[Baral 03] Baral, C.: *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press（2003）

[Baral 09] Baral, C., Gelfond, M. and Rushton, N.: Probabilistic reasoning with answer sets, *Theory and Practice of Logic Programming*, Vol. 9, No. 1, pp. 57-144（2009）

[Bonatti 04] Bonatti, P. A.: Reasoning with infinite stable models, *Artificial Intelligence*, Vol. 156, No. 1, pp. 75-111（2004）

[Brewka 99] Brewka, G. and Eiter, T.: Preferred answer sets for extended logic programs, *Artificial Intelligence*, Vol. 109, Nos. 1-2, pp. 297-356（1999）

[Brewka 02] Brewka, G.: Logic programming with ordered disjunction, *Proc. 18th National Conf. on Artificial Intelligence*, pp. 100-105, MIT Press（2002）

[Chan 93] Chan, E. P. F.: A possible world semantics for disjunctive databases, *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, No. 2, pp. 282-292（1993）

[Clark 78] Clark, K. L.: Negation as failure, In: H. Gallaire and J. Minker（eds.）, *Logic and Data Bases*, pp. 293-322, Plenum（1978）

[Dell'Armi 03] Dell'Armi, T., Faber, W., Ielpa, G., Leone, N. and Pfeifer, G.: Aggregate functions in disjunctive logic programming: semantics, complexity, and implementation in DLV, *Proc. 18th Int. Joint Conf. on Artificial Intelligence*, pp. 847-852, Morgan Kaufmann（2003）

[De Vos 04] De Vos, M. and Vermeir, D.: Extending answer sets for logic programming agents, *Annals of Mathematics and Artificial Intelligence*, Vol. 42, Nos. 1-3, pp. 103-139（2004）

[Eiter 95] Eiter, T. and Gottlob, T.: On the computational cost of disjunctive logic programming: propositional case, *Annals of Mathematics and Artificial Intelligence*, Vol. 15, Nos. 3-4, pp. 289-323（1995）

[Eiter 08] Eiter, T., Ianni, G., Lukasiewicz, T, Schindlauer, R. and Tompits, H.: Combining answer set programming with description logics for the semantic Web, *Artificial Intelligence*, Vol. 172, Nos. 12-13, pp. 1495-1539（2008）

[Elkabani 04] Elkabani, I., Pontelli, E. and Son, T. C.: Smodels with CLP and its applications: a simple and effective approach to aggregates in ASP, *Proc. 20th Int. Conf. on Logic Programming, Lecture Notes in Computer Science,* 3132, pp. 73-89, Springer（2004）

[Fernández 95] Fernández, J. A. and Minker, J.: Bottom-up compuation of perfect models for disjunctive theories, *J. Logic Programming*, Vol. 25, No. 1, pp. 33-51（1995）

[古川 08] Russell, S. J. and Norvig, P. 著，古川康一 監訳：エージェントアプローチ人工知能，第2版，共立出版（2008）

[Gelfond 88] Gelfond, M. and Lifschitz, V.: The stable model semantics for logic programming, *Proc. 5th Int. Conf. and Symp. on Logic Programming*, pp. 1070-1080, MIT Press（1988）

[Gelfond 91] Gelfond, M. and Lifschitz, V.: Classical negation in logic programs and disjunctive databases, *New Generation Computing*, Vol. 9, Nos. 3-4, pp. 365-385（1991）

[Gelfond 08] Gelfond, M.: Answer sets, In: F. van Harmelen, V. Lifschitz, and B. Porter（eds.）, *Handbook of Knowledge Representation*, pp. 285-316, Elsevier（2008）

[Inoue 92] Inoue, K., Koshimura, M. and Hasegawa, R.: Embedding negation as failure into a model generation theorem prover, *Proc. 11th Int. Conf. on Automated Deduction, Lecture Notes in Artificial Intelligence,* 607, pp. 400-415, Springer（1992）

[Inoue 94] Inoue, K.: Hypothetical reasoning in logic programs, *J. Logic Programming*, Vol. 18, No. 3, pp. 191-227（1994）

[Inoue 96] Inoue, K. and Sakama, C.: A fixpoint characterization of abductive logic programs, *J. Logic Programming*, Vol. 27, No. 2, pp. 107-136（1996）

[Inoue 98] Inoue, K. and Sakama, C.: Negation as failure in the head, *J. Logic Programming*, Vol. 35, No. 1, pp. 39-78（1998）

[井上 08] 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, pp. 20-32（2008）

[井上 10] 井上克巳：アブダクションとインダクション，人工知能学会誌，Vol. 25, No. 3, pp. 389-399（2010）

[Jaffer 94] Jaffer, J. and Maher, M. J.: Constraint logic programming: a survey, *J. Logic Programming*, Vols. 19-20, pp. 503-581（1994）

[Kakas 92] Kakas, A. C., Kowalski, R. A. and Toni, F.: Abductive logic programming, *J. Logic and Computation*, Vol. 2, No. 6, pp. 719-770（1992）

[勝野 90] 勝野裕文：演繹データベースの形式的意味論，情報処理，Vol. 31, No. 2, pp. 198-205（1990）

[Kowalski 74] Kowalski, R. A.: Predicate logic as a programming language, *Information Processing*, Vol. 74, pp. 569-574, North-Holland（1974）

[Kowalski 79] Kowalski, R. A. Algorithm = Logic + Control, *Communications of the ACM*, Vol. 22, No. 7, pp. 424-436（1979）

---

*19 例えば，clasp: `http://www.cs.uni-potsdam.de/clasp/`
cmodels: `http://www.cs.utexas.edu/~tag/cmodels/`
dlv: `http://www.dbai.tuwien.ac.at/proj/dlv/`
smodels: `http://www.tcs.hut.fi/Software/smodels/`
などがある．

[Kowalski 90] Kowalski, R. A. and Sadri, F.: Logic programs with exception, *Proc. 7th Int. Conf. on Logic Programming*, pp. 598-613, MIT Press（1990）
[Leone 06] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F: The DLV system for knowledge representation and reasoning, *ACM Trans. on Computational Logic*, Vol. 7, No. 3, pp. 1-57（2006）
[Lifschitz 92] Lifschitz, V. and Woo, T. Y. C.: Answer sets in general nonmonotonic reasoning（preliminary report）, *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pp. 603-614, Morgan Kaufmann（1992）
[Lifschitz 02] Lifschitz, V.: Answer set programming and plan generation, *Artificial Intelligence*, Vol. 138, No. 1, pp. 39-54（2002）
[Lifschitz 08] Lifschitz, V.: What is answer set programming? *Proc. 23rd AAAI Conf. on Artificial Intelligence*, pp. 1594-1597（2008）
[Lloyd 87] Lloyd, J. W.: *Foundations of Logic Programming*, 2nd edition, Springer（1987）
[Manthey 88] Manthey, R. and Bry, F.: SATCHMO: a theorem prover implemented in Prolog, *Proc. 9th Int. Conf. on Automated Deduction, Lecture Notes in Computer Science,* 310, pp. 415-434, Springer（1988）
[Marek 91] Marek, V. W. and Truszczyński, M.: Autoepistemic logic, *J. ACM*, Vol. 38, No. 3, pp. 588-619（1991）
[Marek 99] Marek, V. W. and Truszczyński, M.: Stable models and an alternative logic programming paradigm, In: K. R. Apt, V. M. Marek, M. Truszczynski and D. S. Warren（eds.）, *The Logic Programming Paradigm ―A 25 Year Perspective*, pp. 375-398, Springer（1999）
[Marek 07] Marek, V. W., Niemelä, I. and Truszczyński, M.: Logic programs with monotone abstract constraint atoms, *Theory and Practice of Logic Programming*, Vol. 8, No. 2, pp. 167-199（2007）
[Mellarkod 08] Mellarkod, V. S., Gelfond, M. and Zhang, Y.: Integrating answer set programming and constraint logic programming, *Annals of Mathematics and Artificial Intelligence*, Vol. 53, Nos. 1-4, pp. 251-287（2008）
[Minker 82] Minker, J.: On indefinite data bases and the closed world assumption, *Proc. 6th Int. Conf. on Automated Deduction, Lecture Notes in Computer Science,* 138, pp. 292-308, Springer（1982）
[Niemelä 99] Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm, *Annals of Mathematics and Artificial Intelligence*, Vol. 25, Nos. 3-4, pp. 241-273（1999）
[Przymusinski 88] Przymusinski, T. C.: On the declarative semantics of deductive databases and logic programs, In: J. Minker（ed.）, *Foundations of Deductive Databases and Logic Programming*, pp. 193-216, Morgan Kaufmann（1988）
[Przymusinski 89] Przymusinski, T. C.: On the declarative and procedural semantics of logic programs, *J. Automated Reasoning*, Vol. 5, No. 2, pp. 167-205（1989）
[Rajasekar 89] Rajasekar, A., Lobo, J. and Minker, J.: Weak generalized closed world assumption, *J. Automated Reasoning*, Vol. 5, No. 3, pp. 293-307（1989）
[Reiter 78] Reiter, R.: On closed world databases, In: H. Gallaire and J. Minker（eds.）, *Logic and Data Bases*, pp. 55-76, Plenum（1978）
[Ross 88] Ross, K. A. and Topor, R. W.: Inferring negative information from disjunctive databases, *J. Automated Reasoning*, Vol. 4, No. 4, pp. 397-424（1988）
[Sadri 88] Sadri, F. and Kowalski, R.: A theorem-proving approach to database integrity. In: J. Minker（ed.）, *Foundations of Deductive Databases and Logic Programming*, pp. 313-362, Morgan Kaufmann（1988）
[Sakai 84] Sakai, K. and Miyachi, T.: Incorporating naive negation into Prolog, *RIMS Symposia on Software Science and Engineering II, Lecture Notes in Computer Science,* 220, pp. 130-143, Springer（1984）
[Sakama 90] Sakama, C.: Possible model semantics for disjunctive databases, In: W. Kim, J.-M. Nicolas and S. Nishio（eds.）, *Deductive and Object-Oriented Databases*, *Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases*（*DOOD'89*）, pp. 369-383, North-Holland（1990）
[Sakama 94] Sakama, C. and Inoue, K.: An alternative approach to the semantics of disjunctive logic programs and deductive databases, *J. Automated Reasoning*, Vol. 13, No. 1, pp. 145-172（1994）
[Sakama 95] Sakama, C. and Inoue, K.: Paraconsistent stable semantics for extended disjunctive programs, *J. Logic and Computation*, Vol. 5, No. 3, pp. 265-285（1995）
[Sakama 00] Sakama, C. and Inoue, K.: Prioritized logic programming and its application to commonsense reasoning, *Artificial Intelligence*, Vol. 123, Nos. 1-2, pp. 185-222（2000）
[Sakama 08] Sakama, C. and Inoue, K.: Coordination in answer set programming, *ACM Trans. Computational Logic*, Vol. 9, No. 2, Article No. 9（2008）
[Sato 90] Sato, T.: Completed logic programs and their consistency, *J. Logic Programming*, Vol. 9, No. 1, pp. 33-44（1990）
[佐藤 08] 佐藤 健，デ・ヴォス，マリナ：論理コンピューティング，人工知能学会誌，Vol. 23, No. 5, pp. 677-686（2008）
[Shepherdson 84] Shepherdson, J. C.: Negation as failure: a comparison of Clark's completed database and Reiter's closed world assumption, *J. Logic Programming*, Vol. 1, No. 1, pp. 51-79（1984）
[Simons 02] Simons, P., Niemelä, I. and Soininen, T.: Extending and implementing the stable model semantics, *Artificial Intelligence*, Vol. 138, Nos. 1-2, pp. 181-234（2002）
[Son 06] Son, T. C., Pontelli, E. and Tu, P. H.: Answer sets for logic programs with arbitrary abstract constraint atoms, *Proc. 21st National Conf. on Artificial Intelligence*, pp. 129-134（2006）
[van Emden 76] van Emden, M. H. and Kowalski, R. A.: The semantics of predicate logic as a programming language, *J. ACM*, Vol. 23, No. 4, pp. 733-742（1976）
[van Gelder 91] van Gelder, A., Ross, K. and Schlipf, J. S.: The well-founded semantics for general logic programs, *J. ACM*, Vol. 38, No. 3, pp. 620-650（1991）

2010年3月1日 受理

著 者 紹 介

坂間 千秋（正会員）

1985年京都大学工学部情報工学科卒業．同年（株）東芝総合研究所入所．その後，（財）新世代コンピュータ技術開発機構（ICOT）（1986～89），（財）京都高度技術研究所（1989～95）を経て，1995年より和歌山大学システム工学部．現在，情報通信システム学科教授（知能情報処理講座）．博士（工学，京都大学）．人工知能基礎，マルチエージェントシステムなどの研究に従事．

井上 克巳（正会員）

1982年京都大学工学部数理工学科卒業．1984年同大学院工学研究科数理工学専攻修士課程修了．松下電器産業（株），（財）新世代コンピュータ技術開発機構（ICOT），豊橋技術科学大学，神戸大学を経て，2004年より国立情報学研究所．現在，同情報学プリンシプル研究系教授および総合研究大学院大学複合科学研究科情報学専攻教授（併任）．2008年より東京工業大学大学院情報理工学研究科計算工学専攻連携教授．博士（工学，京都大学）．情報処理学会，AAAI各会員．人工知能，システム生物学などの研究に従事．