特　集　「定理証明，推論関係の新技術」

論理プログラムの解集合意味論に関する証明系

Proof Procedures of Answer Set Semantics for Logic Programming

岩山　登　（株）富士通研究所
Noboru Iwayama　Fujitsu Laboratories Ltd.
iwayama@acm.org

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

Keywords: stable model, answer set, integrity constraint, logic programming, negation as failure.

1. はじめに

本稿では，失敗による否定を含む論理プログラミングの証明系について，特に解集合意味論による証明系について紹介する．本稿はエッセンスを理解することに重点を置き，厳密な定義については言及しない．

命題論理についての基礎知識を仮定する（本稿の議論は命題の場合のみを説明する．変数と関数記号をもつ場合に拡張して議論が可能な場合もあるが，命題の場合に制限しても要点は十分理解できるようにした）．Prologやその意味論についての知識があれば，理解の助けになる．

まず，基本概念を説明する．「論理プログラミング」とは，「含意記号の意味（と使い方）を制限した論理体系」である．具体的には，論理プログラミングでは通常の命題論理でいう節形式で表現する（ルールと呼ぶ）が，

(1) 後件が空でない（$P \wedge Q \supset false$ という形の論理式がない）

また，

(2) 対偶を考えない

という特徴がある（論理プログラミングによる，個別の節集合のことを論理プログラム，あるいは，プログラムという）．

ここで「証明系」について説明しておく．本稿では通常より広い意味で「証明系」という用語を使っている．融合原理（resolution）によって矛盾を導く場合（トップダウン計算）だけでなく，すべての論理式を満たすモデルを陽に生成する場合（ボトムアップ計算）も「証明系」と考える．

論理プログラミングにつけた制限により，証明が効率的になる．トップダウン計算でルールを後件から前件の向きに適用していくだけで，逆に，ボトムアップ計算ではルールを前件から後件の向きに適用させていくだけで，計算ができる．

特に，ルールの後件部が一つの命題しかもたない節（ホーン節：本稿では $P \leftarrow Q \wedge R$ のように書く）だけを対象とする場合（ホーン節論理プログラミングという）は，非常に単純に証明できる．これがまさにPrologの対象とするところである．以下でも，ホーン節だけの場合について述べる．

論理プログラミングにつけた制限によって証明が単純化できるが，失ったこともある．それは「否定を表現できない」ことである*1．人間にとってごく自然な概念である，「～でない」という表現ができないのである．

そこで，ある種のメタな操作が考案された．それが「失敗による否定（Negation as failure）」である．例えば，命題 P の失敗による否定は $not\ P$ と書き，「命題 P が証明できないならば，P は偽である」とみなすというのがアイディアである．「いかなる証明も“失敗”したら，その命題の否定が成り立つ」というわけである．論理プログラミングは証明が効率的であり，「証明できないこと」は（効率的であるはずの）証明からわかるだろう，というのが着眼点である．例えば，$\{P \leftarrow not\ Q\}$ という論理プログラムでは，Q は証明できないので Q は偽である．

失敗による否定がない場合，ホーン節論理プログラミングでは宣言的意味と証明との対応が示されている*2．失敗による否定を含む場合も，宣言的意味と対応がとれ，かつ，効率の良い証明系が必要とされることは論を待た

*1 正確にいうと，論理プログラミングの範囲内では「～でない」という主旨の言明を表現できない．

*2 最小エルブランモデル，ボトムアップ計算の結果，SLD resolutionによるトップダウン計算の結果の，三者が一致する．

ない．

失敗による否定の意味論の研究には，多大な努力が払われた（その歴史的な経緯は [Minker 93] が参考になる）．解集合意味論はその歴史上で重要な位置を占める．

自己認識論理の stable expansion にヒントを得て，Gelfond と Lifschitz が安定モデル意味論（Stable model semantics）[Gelfond 88]（のちに解集合意味論，Answer set semantics [Gelfond 91] と呼ばれる）を 1988 年に提案した*3．また，解集合意味論と対照的な性質をもつ Well-founded semantics が同時期に提案され [van Gelder 91]，それ以後，解集合意味論と Well-founded semantics の両者を軸に，失敗による否定に関わる意味論，証明系，非単調推論との関係などの研究が進展した．

本稿では，宣言的意味として「解集合意味論」に基づいた証明系について述べる．

2章で解集合意味論について Well-founded semantics と対比させながら説明する．3章では基本となる証明系について，4章では高速化のために基本の証明系に加えた工夫について説明する．

2. 解集合意味論

ある論理プログラムの解集合とは，下に述べる定義の条件を満たす命題の集合である．解集合に含まれる命題はその論理プログラムで真であり，解集合に含まれない命題は偽とみなされる．

あるルールに対し，*not* が付かない前件部の命題のことをそのルールの正リテラル，逆に *not* が付く前件部の命題をそのルールの負リテラルという．

2·1 解集合意味論の定義

【定義 2·1】 プログラム T の解集合 M とは，以下の条件を満たす命題の集合である：

[条件] 次の二つの操作によって得られる，失敗による否定を含まないプログラム T^M で真となる命題の集合（最小エルブランモデル）と，M が等しい．

(1) T に含まれるルールのうち，そのルールの負リテラルで M に含まれるものがあれば，そのルールを T から取り除く．

(2) 残ったルールの前件部から，すべての負リテラルを取り去る．

定義の意図は，どの命題が真になるかを仮定し（同時に真と仮定しなかった命題は偽であることを仮定したことになる），仮定した命題が依然導けること（仮定しなかった命題については導けないこと）を保証するところにある．

$T=\{P \leftarrow not\ Q,\ Q \leftarrow not\ R\}$ に対し，$M=\{Q\}$ を考えると，$T^M=\{Q \leftarrow\}$ となる．M と T^M の最小エルブランモデルは一致する．一方 $M=\{P, R\}$ を考えると，$T^M=\{P \leftarrow\}$ となる．この場合，M と T^M の最小エルブランモデルは一致しない．

例を示す．

(1) $\{P \leftarrow not\ Q\}$ は解集合 $\{P\}$ をもつ．つまり，命題 P は真，解集合に入っていない命題 Q は，偽になる．

(2) $\{P \leftarrow not\ Q,\ Q \leftarrow not\ R\}$ は解集合 $\{Q\}$ をもつ．つまり，命題 Q は真，解集合に入っていない命題 P, R は，偽になる．この例は，失敗による否定をもつプログラムの意味が，極小モデルでは正確に表現できない説明にもなっている．*not* を通常の否定とみなすと，このプログラムは二つの極小モデル $\{Q\}$ と $\{P, R\}$ をもつ．後者は，R を導くルールがないにもかかわらず R が真となり，「失敗による否定」という観点からすると不自然である．

(3) $\{P \leftarrow Q\}$ は解集合 $\{\ \}$ をもつ．つまり，すべての命題 P, Q は，偽になる．

(4) $\{P \leftarrow not\ P\}$ には解集合がない．$\{P \leftarrow not\ Q,\ Q \leftarrow not\ R,\ R \leftarrow not\ P\}$ にも解集合がない．「解集合がない」ことと，「空の解集合がある」ことは区別する必要がある．「解集合がない」ことは，すべての命題に対し，値を決定できないとみなしてもよい．

(5) $\{P \leftarrow not\ Q,\ Q \leftarrow not\ P\}$ は解集合を二つもつ．$\{P\}$ と $\{Q\}$ である．すなわち，それぞれの解集合では，P か Q のいずれか一方が真で，他方が偽になる．

2·2 解集合意味論の特徴

解集合意味論には次の特徴があり，対象とする応用領域や解集合の計算にとって重要な意味がある．

(1) 複数の解集合をもつプログラムがある．例 (5) に示したように，一つのプログラムが複数の解集合をもつ場合がある．これにより対象領域における曖昧さを表現できる．しかし，後述するように計算量が増える要因でもある．

(2) 2値である．一般にはプログラムは複数の解集合をもつため，証明の過程で場合分けを行うことになる．場合分けをうまく行えれば，計算量が抑えられる可能性がある．解集合意味論では，すべての命題についてその値が真か偽のいずれかに決まるという性質がある．この性質を利用すると，冗長な場合分けを回避できる可能性がある．

*3 解集合意味論は安定モデル意味論を拡張して explicit negation（「失敗による否定」より強い否定の一種）を扱えるようにしたものである．安定モデルの「モデル」という表現が通常の論理でいうモデルとは異なる意味で用いられており，誤解を招くため，「解集合」と呼ばれるようになった．この辺りの事情は [Lifschitz 00] に若干説明されている．なお，explicit negation を考慮しない通常の論理プログラムでは，解集合と安定モデルは同一のものを指す．本稿では，explicit negation について言及していないが，「解集合」に統一している．

(3) 解集合をもたないプログラムがある．例（4）に示すプログラムだけでなく，例（4）に示したルールに，それらと命題を共有しないルールを加えたプログラムも，解集合をもたない．この性質は解集合の計算にとって大きな意味がある．

例えば，$\{P \leftarrow not\ P, Q \leftarrow\}$ は解集合をもたない．Q についてトップダウンに証明木を構成できるが，それだけでは Q を含む解集合の存在を示したことにならない．すなわち，一般には解集合の存在を示すにはトップダウンの計算だけでは不十分であり，ボトムアップ計算が必須である．

2·3 Well-founded semantics の特徴

比較のために，Well-founded semantics の性質について紹介する．

直観的にいうと，Well-founded semantics は，Prolog の計算によって「失敗による否定」の計算が停止した場合を，適切に説明する意味論ということができる．

「失敗による否定」の計算が停止しない場合とは，ある命題の「失敗による否定」の計算の過程で，その命題の「失敗による否定」の計算が必要になる場合をいう．例（5）に対し，ゴール P を計算する場合を考える．P が真になるためには，Q の失敗による否定の計算が必要になる．ゴール Q を計算するためには，P の失敗による否定の計算が必要になる．さらにゴール P を計算するためには，Q の失敗による否定の計算が必要になる．

Well-founded semantics では，「失敗による否定」の計算が停止する場合，命題は真，偽のいずれかの値になるが，「失敗による否定」の計算が停止しない場合は，命題の値は未定義になる．

Well-founded semantics では，一つのプログラムが必ず一つの well-founded モデルをもつ．例（5）の well-founded モデルでは P も Q も未定義である．

XSB [Rao 97] という処理系は，Prolog を拡張し，「失敗による否定」の計算が停止するかどうかを検出する仕組みを備え，Well-founded semantics を計算できる．

3. 解集合の計算

3·1 最初のボトムアップ証明系

解集合意味論に関する証明系のうち，最も初期に提案されたのが [Fages 90, Saccà 90] の手法である．この手法によって，すべての解集合をボトムアップに構成して求められる．

この手法では現在のモデル候補に照らして，適用可能なルールがあるとき，そのルールを用いてモデル候補を拡張していく．複数適用可能なルールがあるときは，非決定的に選択する．つまり，場合分けが生じることになる．この手法は以下で述べる証明系の基本になるので，やや詳しく説明する．

「モデル候補」とは，計算の過程で，それまでに真偽値が確定した命題の集合を指す．モデル候補に対し「ルールが適用が可能」であるとは，ルールの後件部（ルールのヘッドである命題）がモデル候補で真でなく，ルールの正リテラルがすべてモデル候補で真であり，かつ，ルールの負リテラルがすべてモデル候補で真でない（偽であると確定しているか，その時点で値が決まっていない）ことをいう．また，適用可能なルールで「モデルを拡張」するとは，ルールの後件部を真，ルールの負リテラルを偽として，命題をモデル候補に追加する操作をいう．モデル「拡張が失敗」するとは，命題が真と偽の両方の値をもつことになった場合を指す．

プログラム $\{Q \leftarrow not\ R, P \leftarrow not\ P\}$ を考える（このプログラムは解集合をもたない）．この手続きは，初期状態のモデル候補を { } として計算を開始する．二つのルールはいずれもモデル候補 { } に対し適用可能である．ルール $Q \leftarrow not\ R$ でそのモデル候補を拡張すると，新しいモデル候補は $\{Q, not\ R\}$ になる（Q は真，R は偽であるという意味）．新しいモデル候補はルール $P \leftarrow not\ P$ でさらに拡張可能で，拡張後のモデル候補は$\{Q, not\ R, P, not\ P\}$ になり，この拡張は失敗となる．このため，バックトラックし，最初に適用したルールとは異なるルール（つまり $P \leftarrow not\ P$）を選びなおして計算を進める．しかし $\{P, not\ P\}$ も拡張が失敗しているので，結局解集合がないことがわかる．

3·2 初期のトップダウン処理系

ゴールを含む解集合が存在するかどうかを計算する初期のトップダウン処理系に，[Eshghi 89] がある．この方法では，トップダウン計算において否定すべき命題を記録しループチェックを行うことにより計算を進める．

例えば $\{P \leftarrow not\ Q, Q \leftarrow not\ P\}$ に対し，P をゴールとして計算する場合を考える．P を導くためには，$not\ Q$ であることが必要である．そこで $not\ Q$ を記録する．次に，「失敗による否定」の用件を満たすために，Q が導けないことを調べる．Q が導けないことは $not\ P$ でないこと，すなわち，P であることにより保証される．$not\ Q$ であることを記録していたので，P であることが保証される．つまり，$not\ Q$ であるような解集合において P が真となることが計算できたことになる．

この方法には，解集合意味論の証明として問題がある．プログラム $\{P \leftarrow not\ Q, Q \leftarrow not\ P, R \leftarrow P \wedge not\ R\}$ を考えて，ゴールを P としても，上記の計算過程とまったく同様になる．このプログラムには P を含む解集合が存在しないにもかかわらず，P を含む解集合が存在すると計算されてしまっている．

この手続きが計算しているのは，解集合意味論とは若干異なる意味論（preferred extension）であり，その性質は [Dung 91] によって明らかにされている．

4. 解集合証明系の効率化

4·1 証明系の複雑さと効率化のポイント

解集合の計算は一般には容易でないことが理論的に示されている．あるプログラムに対して「解集合が存在するかどうかを決定する問題」は，NP完全である[Dantsin 01][*4]．

解集合計算の効率の悪さは，前章で述べたルール選択の非決定性に関連している．これを改善するために，以下の3種類の方法が考案されてきた．

(1) ゴール指向（トップダウン計算）：与えられたゴールを満たす解集合を求めたい場合，ゴールが導出できないモデル候補を計算から除外することによって，計算を効率化できる．前節で述べたようにトップダウン計算だけでは解集合の存在を保証できないので，ボトムアップ計算と組み合わせる必要がある．

(2) 一貫性制約（integrity constraint）：論理プログラミングでは「否定を表現できない」が，ある種の便法を用いると「～でない」という言明を表現することが可能になる．それが，一貫性制約である．例えば，「Pでない」という言明は$\bot \leftarrow P$と書く．$\bot$は特別な「命題記号」で，矛盾を意味すると考えればよい．

一貫性制約を「～でない」という意味に解釈するためには，ある種のメタ的な取扱いが必要なことに注意してほしい．それは，$\bot$を含む解集合を破棄することである．これがなぜ「～でない」の表現になるかというと，$\bot \leftarrow P$を含む論理プログラムのモデルのうち，$\bot$が含まれないものだけを考えると，それらのモデルでは必ずPが含まれない．つまり，Pが真であるモデルを排除している．

一貫性制約によって計算の効率化が図れる．計算のできるだけ早い段階で制約に反していることがわかると，それ以上不要な計算が必要なくなり，特に計算の過程での分岐を抑制できるからである．

(3) 命題による分岐：解集合意味論では，各命題は真か偽かのいずれかの値をとる．このことを利用すると，計算の過程で分岐が必要になったとき，適用可能なルールで場合分けするより，命題の真偽で場合分けしたほうが計算が効率的になることが多いと考えられる．その理由は，第一に不必要な分岐を考慮する必要がなくなる，第二に命題の真偽で場合分けすることにより，計算の状態により強い制約を与えられるからである．

*4 さまざまな論理プログラムについての計算の複雑さについてサーベイされている．

5. 効率化された解集合証明系

以下では，提案された解集合証明系を，これらの三つの方法の観点から分類して紹介する．

5·1 ボトムアップ証明系にトップダウンの要素を追加

[Iwayama 00]の基本的アイディアは，与えられたプログラムの解集合を求めるために[Fages 90, Saccà 90]のボトムアップ証明系に改良を加え，適用可能なルールのうちゴール導出に関係するものを優先して選択することにある．ゴール導出に関係するルールとは，そのゴールからトップダウンに証明を構成した際に使われる可能性のあるルールをいう．適用可能だがゴール導出に関係しないルールについては，適用を後回しにする．このことにより，ゴールの導出に貢献しない場合分けをできるだけ遅らせることができる．もしゴールの導出が不可能であることがわかれば，場合分けを遅延させた分だけ計算を効率化できる．

すなわち，[Iwayama 00]では，複数の適用可能なルールがあるときは，ゴールからトップダウンに証明を構成し，その証明に使われるルールを優先適用してモデル候補を拡張することにより，不必要な場合分けを回避できる．

さらに，一貫性制約をチェックすることで，できるだけ早い段階でモデル拡張の失敗を検出するようにしている．[Iwayama 00]において特徴的なのは，ルールを動的に一貫性制約とみなすことにより，一層の効率化を図っていることである．例えば，$P \leftarrow Q$というルールがあって，計算の過程でPが偽であることがわかったとき，このルールは$\bot \leftarrow Q$という一貫性制約とみなせる．

また同様に，$P \leftarrow not\ Q$というルールがあって，計算の過程でPが偽であることがわかったとき，このルールは$\bot \leftarrow not\ Q$という一貫性制約とみなせる．この一貫性制約はQが導かれなければならないことを意味する．つまり，Qはゴールであることになり，先に述べた，ゴール導出に関係するルールを優先して選択する手法を適用できる．このように，特にゴールが与えられていない場合でも，トップダウン計算による効率化の恩恵を享受できる．

例を示す．プログラム$\{P \leftarrow not\ Q, Q \leftarrow not\ P, R \leftarrow P \wedge not\ R\}$について，$P$をゴールとして計算してみる．このプログラムの解集合は$\{Q\}$だけで，命題Pが含まれる解集合は存在しないため，ゴールPは失敗することになる．初期モデル候補$\{\}$に適用可能なルールは二つあるが，ゴールPに関係するルール（Pを導くことのできるルール）は，$P \leftarrow not\ Q$のみである．よって，このルールでモデルを拡張する．この時点でのモデル候補$\{not\ Q, P\}$を拡張可能なルールは，$R \leftarrow P \wedge not\ R$のみであるが，このルールによるモデル拡張は失敗する．よ

って，命題Pが含まれる解集合は存在しないことが計算できた．最初のモデル拡張において，ルール$Q \leftarrow not\ P$の場合を考慮しなくてよいことに注意してほしい．なぜなら，このルールはゴールのトップダウン証明の構成に無関係であるので，適用が遅延されなければならないからである．

5·2　トップダウン証明系にボトムアップの要素を追加

[Satoh 96] では，ゴールを充足する解集合が存在するかどうかを計算するために，[Eshghi 89] のトップダウン処理系を用い，さらにボトムアップ計算などを補うことにより，[Eshghi 89] が解集合意味論と整合しないという問題を回避している．

トップダウン計算によってゴールに対する証明木を構成しながら，その証明木に現れた命題が出現するルールすべてをチェックする．ルールがモデル候補を充足する（ルールの前件が偽になるか，前件が真かつ後件も真になる）ように，未決定の命題の値が決められる．

この手法では，プログラムが少なくとも解集合を一つもつ場合のみ，正しく振る舞う．

さきほどの例を再度考える．プログラム$\{P \leftarrow not\ Q, Q \leftarrow not\ P, R \leftarrow P \wedge not\ R\}$について，$P$をゴールとして計算する．ゴール$P$を導くためには，$not\ Q$が必要となる．

3·2節の計算と同様に，ここで$not\ Q$を記録する．次に，「失敗による否定」の用件を満たすために，Qが導けないことを調べる．Qが導けないことは$not\ P$でないこと，すなわち，Pが真であることにより保証される．$not\ Q$を記録していたので，Pが真であることが保証されたことになる．ここまでは [Eshghi 89] の方法と同じである．次に$not\ Q$からボトムアップに計算を行う．すると，Pが真となる．さらに，Pが真となったことで，ルール$R \leftarrow P \wedge not\ R$について，ルールが充足可能であるかを確認する必要がある．しかしながら，ほかにRを後件部にもつルールがないため，Rは真偽いずれでも矛盾することになり，このルールは充足できない．よって，ゴールPが真になる解集合は存在しないことが計算できた．

5·3　場合分けを各命題の真偽で行うボトムアップ証明系

[Niemelä 96, Niemelä 97]による計算は，与えられたプログラムに対する解集合を求めるために，ボトムアップを基本としてモデルを構成していく．この手法は，上に述べたボトムアップ計算[Fages 90, Iwayama 00, Saccà 90] とは次の2点で異なっている．

(1) 場合分けを適応可能なルールではなく，各命題の真偽で行う．場合分けを適応可能なルールに出現するnotつきの命題についての真偽で行う．これにより，適応可能なルールに，同じ命題が出現していたとしても，その同じ命題について繰り返し計算する必要がなくなる．

(2) ある命題について場合分けを行い，真であると仮定したら，実際にその命題が真であることをボトムアップ計算に利用する．このことにより，計算過程のプログラムサイズが連続的に縮小していくので，計算の高速化が期待できる．しかしながら，真であると仮定した命題はあくまでも「仮定」しただけであり，現時点のモデル候補のもとで導けるかどうかを確認する必要がある．

[Niemelä 96, Niemelä 97]による計算はC++で実装され，当時としてはかなり高速であったため，解集合意味論だけでなく非単調推論システムの証明系について研究が盛んになるきっかけとなった．

5·4　モデル生成定理証明系を利用する手法

[Inoue 92] は，失敗による否定を含む論理プログラムを，失敗による否定を含まない命題論理式に変換し，変換後の論理式のモデルのうち，ある条件を満たすものがもとの論理プログラムの解集合に対応することを示した．変換後の論理式のモデルは，モデル生成定理証明系によってボトムアップに計算できる．この方法がほかの手法と比べて特徴的なのは，計算の場合分けが，モデル生成定理証明系におけるモデル生成過程で行われることである．

6. 一階の場合への拡張

プログラムが命題ではなく，変数をもつ述語からなる場合の取扱いについて簡単に触れる．上記で述べた解集合意味論の証明手法において変数の扱いが問題になるのは，負リテラルに束縛されていない変数が残ってしまうことである．ボトムアップ計算の場合，モデル拡張において適用可能なルール中で，負リテラルに束縛されていない変数が残っていると，エルブラン空間が無限である場合モデル拡張において適用可能なルールが無限個存在することになり，計算が停止しない．トップダウン計算の場合も同様な問題が生じる．

この問題はルールに構文的な制限をつけることによって回避できる．それは，ルールの後件部と負リテラルに出現する変数は，必ずそのルールの正リテラルに出現する（range-restricted）という条件である．この条件を満たすプログラムに対しては正リテラルの計算を先に行うことにより，ルールの後件部と負リテラルに出現する変数は必ず束縛されることになる．

7. まとめにかえて

失敗による否定を含む論理プログラミングの証明系について，特に解集合意味論による証明系について紹介した．解集合意味論の計算がどういった問題に役立つかに

ついて触れて，締めくくる．

失敗による否定を含む論理プログラミングの主な役割として知識表現がある．失敗による否定を含む論理プログラミングによって問題を記述し，解集合を計算することによって問題の解を求める枠組みは，「解集合プログラミング」と呼ばれており [Lifschitz 00]，プランニングやアクションなどの応用が取り上げられている．

また，アブダクションと解集合意味論の関係も重要である．論理プログラムの簡単な拡張でアブダクションが表現でき*5，解集合意味論もアブダクション用に拡張されている [Kakas 90]．

前節で述べた [Iwayama 00, Satoh 96] の手続きは，本来アブダクション用に拡張された論理プログラムを計算するためのものである．アブダクションとは，そもそも「ゴールを導くのに必要な仮説を求めること」である以上，手続きもゴール指向であることが望ましい．[Iwayama 00, Satoh 96] では，解集合計算の高速化と，アブダクションのためのゴール指向計算が同じ仕組みで実現できていることになる．

プランニング，アクション，アブダクションは知識表現手法であって，その手法を用いてどんな問題を記述するかが問題となる．著者の一人は，法的推論やソフトウェア工学になどについて取り組んできた [佐藤 97, 佐藤 00]．本稿をきっかけとして，新たな証明系の提案や，新しい応用への取組みがなされることを期待する．

◇ 参 考 文 献 ◇

[Dantsin 01] Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A.: Complexity and expressive power of logic programming, to be appeared in *ACM Computing Surveys* (2001)

[Dung 91] Dung, P. M.: Negations as hypotheses:an abductive foundation for logic programming, *Proc. of ICLP'91,* pp. 3-17 (1991)

[Eshghi 89] Eshghi, K. and Kowalski, R. A.: Abduction Compared with Negation by Failure, *Proc. of ICLP'89,* pp. 234-254 (1989)

[Fages 90] Fages, F.: A New Fixpoint Semantics for General Logic Programs Compared with the Well-Founded and the Stable Model Semantics, *Proc. of ICLP'90,* pp. 442-458 (1990)

[Gelfond 88] Gelfond, M. and Lifschitz, V.: The Stable Model Semantics for Logic Programming, *Proc. of LP'88,* pp. 1070-1080 (1988)

[Gelfond 91] Gelfond, M. and Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing,* pp. 365-385 (1991)

[Inoue 92] Inoue, K., Koshimura, M. and Hasegawa, R.: Embedding negation as failure into a model generation theorem prover, *11th Int. Conf. Automated Deduction,* Lecture Notes in Artificial Intelligence, 697, pp. 400-415 (1992)

[Iwayama 00] Iwayama, N. and Satoh, K.: Computing abduction by using TMS with top-down expectation, *Journal of Logic Programming,* Vol. 44, Nos 1-3, pp. 179-206 (2000)

[Lifschitz 00] Lifschitz, V.: Answer set programming and plan generation, *Artificial Intelligence,* to appear (2000), http://www.cs.utexas.edu/users/vl/mypapers/asppg.ps

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

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

[Minker 93] Minker, J.: An overview of nonmonotonic reasoning and logic programming, *Journal of Logic Programming,* Vol. 17, Nos 1-4, pp. 95-126 (1993)

[Niemelä 96] Niemelä, I., Simons, P.: Efficient Implementation of the Well-founded and Stable Model Semantics, *Proc. of JICSLP'96,* pp. 671-685 (1996)

[Niemelä 97] Niemelä I. and Simons, P.: Smodels -an implementation of the stable model and well-founded semantics for normal logic programs. *Proc. of LPNMR'97* (1997)

[Rao 97] Rao, P., Sagonoas, K., Swift, T., Warren, D. S. and Freire, J.: XSB: A System for Efficiently Computing Well-Founded Semantics, *LPNMR'97* (1997)

[Saccà 90] Saccà, D. and Zaniolo, C.: Stable Models and Non-Determinism in Logic Programs with Negation, *Proc. of PODS'90,* pp. 205-217 (1990)

[Satoh 96] Satoh, K. and Iwayama, N.: A Query Evaluation Method for Abductive Logic Programming Based on Generalized Stable Models, 人工知能学会誌, Vol. 11, No. 1, pp. 137-147 (1996)

[佐藤 97] 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)

[佐藤 00] 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, pp. 109-121 (2000)

[van Gelder 91] van Gelder, A., Ross, K. A. and Schlipf, J. S.: The well-founded semantics for general logic programs, *Journal of the ACM,* Vol. 38, pp. 620-650 (1991)

2001年7月10日 受理

*5 アブダクションと論理プログラミングの関係については [Kakas 92] が詳しい．

著 者 紹 介

岩山 登（正会員）

1988年名古屋大学大学院工学研究科前期課程修了．工学修士．以後現在まで（株）富士通研究所所属．途中1989～93年（財）新世代コンピュータ技術開発機構（ICOT）出向，1997～2000年名古屋大学大学院工学研究科後期課程在学．現在インターネットアプリケーション，特にワイヤレスやメッセージング分野の研究開発，技術標準化に従事．HCI一般と人工知能技術一般，特に論理的手法に興味をもつ．

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．1987～92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）．