(2) 論理型プログラム言語とその方法論

① Multi-Version Structures in Prolog

S. Cohen (Fairchild, 米国)

Prolog に配列やハッシュテーブル，集合等の構造（Multi－Version Structures ；MVS）を導入することは実用上の意義が大きいが，これを効率良くインプリメントすることは容易でない。

ここでは，まず，従来提案されてきた実現方式をアクセス時間，更新時間等の多くの面から検討し，それらの方式が必ずしも十分でないことを示す。

次に 集合や配列を扱うための述語としてsetof や bagof に代わり，set_of_all（E, P, S），one_of（E, S），array_element（A, Index, Value）等が導入される。集合はその性質に応じて，ハッシュテーブルとしても，リストとしても，ビット配列としても表現され得る。

少しずつ値の異なる配列（バージョンが複数ある配列）を表現するには，更新回数を示すカウンタと，更新時とその値を保持するエレメントリストを用いる。これらを用いた配列の更新，アクセス，バックトラック，並列処理の各アルゴリズムが述べられる。この表現方式によれば配列の値を更新した後でも過去の状態へ戻ることが可能である。

② Finding Temporary Terms in Prolog Programs

P. Vataja, E. Ukkonen (Univ. of Helsinki, フィンランド)

Prolog 処理系を作成するとき，項（term）は通常はリンクされた構造で表わされ，各構造は部分構造を共有することができるようになっている。プログラムの実行中にあるデータ構造が他から到達しなくなれば，ガーベジコレクタがこれを除去する。ここではプログラムの実行中に不要な構造を検出する「直接ガーベジコレクタ」を実現するため，テンポラリ・タームの概念を導入する。ある項t がテンポラリであるとは，t が節$P_0 \leftarrow P_1, P_2, \cdots, Pn$ の本体に表われ，この節の実行後にこの節をコールした節の項とデータ構造を共有していないような変数項であるものをいう。

テンポラリ・タームを検出するための２つの方式が提案される。ダイナミックな方式ではプログラムの実行中に検出を行うものである。スタティックな方式では実行前にプログラムを解析し，その結果をガーベジコレクタが用いるものである。テンポラリ・タームの完全な検出は不可能であるが，例を用いてその有効性を示している。

③ Delta-Prolog: A Distributed Logic Programming Language

L.M. Pereira (Univ. Nova de Lisboa, ポルトガル)
R. Nasr (DEC, 米国)

Delta−Prolog は Prolog の拡張として提案された論理型言語である。Prolog はホーン論理を理論的基礎とする言語であるが，Delta−Prolog は分散論理（Distributed Logic）を基礎としている。その特徴は，(1)ゴールを結合するのに逐次ＡＮＤ＇，＇と並列ＡＮＤ＇／＇を用意し，並列性を明示できるようにしたこと。(2)イベント（event）という時間に依存した概念を導入し，プロセス間の通信や同期がとれるようにしたことである。Prolog ではゴールは項Ｇで表わされるが，Delta Prolog ではその他に項Ｇとイベント Ｅを用いて，Ｇ！Ｅあるいは Ｇ′？Ｅの形のゴールが許される。Ｇ！Ｅと Ｇ′？Ｅは同期がとられ，並列処理によって両方が実行される状態になったときにのみ実行が成功する。

Delta Prolog は C−Prolog を基にインプリメントされ，VAX/VMS 上で動いている。Concurrent Prolog と比較したとき，Prolog を完全に包含していること。理論的基盤が整っていること，ストリームによる通信よりもイベントによる通信の方が容易なこと，実際に動いていること，等の利点がある。

④ Unique Features of ESP

近山　隆 (ICOT)

ＥＳＰはＩＣＯＴ で開発中の逐次型推論マシンφの機械語ＫＬ０に基くシステム記述用語である。ＥＳＰでかかれたプログラムはコンパイルされてＫＬ０プログラムとなり，φで実行される。

ＥＳＰの特徴は(1)論理型言語の性質をもち，(2)オブジェクト指向型のプログラミングができ，(3)マクロ定義ができるという点にある。すなわち，ＥＳＰにはユニフィケーションや，バックトラックによるＡＮＤ－ＯＲ木の深さ優先探索等の Prolog の機能が組み込まれている。また，ＥＳＰプログラムは１つ以上のクラス定義から成り，クラス間はサブクラス・スーパークラスの関係を持つ階層構造を構成することができる。メリッドやスロットはスーパークラスから継承される。マクロ機能を用いると，add（Ｘ，Ｙ，Ｚ），ｐ（Ｚ）とかかれていたものを ｐ(Ｘ＋Ｙ）と表わすことができ，プログラムが理解しやすくなる。

φのオペレーティングシステムであるＳＩＭ ＰＯＳ は ＥＳＰ でかかれており，多重継承の機能によって非常に明解なコーディングが可能となっている。また，自然言語のパーザへの応用もなされてきている。

⑤ Notes on Systems Programming in PARLOG

K. Clark, S. Gregory (Imperial College, 英国)

PARLOG は並列性を組み込んだ論理型言語である。Prolog とは(i)並列処理, (ii)解を 1 つだけ探索, (iii)モード宣言の点で異なっている。

この論文は, PARLOG を用いてシステムプログラム（UNIX 的な shell プログラム）を記述した体験を報告したものである。初めに簡単な shell プログラムを示し, このプログラムではコマンドの実行が失敗した場合には対処できないことから, meta-call (goal ?, status) を導入する。これは goal を実行し, その結果 (suceeded , failed , error) を status へ返すものである。

次にABORT コマンドによる実行の中止を扱うために, 新しいmeta-call (goal ?, status, control ?) が導入される。goal の実行中に control に ' STOP ' がインスタンシエートされると, この実行は停止し, そのとき status は ' STOPPED ' になる。

最後に, 優先度を扱う shell を記述するためにこのメタコールは拡張され, control に SUSPEND や CONTINUE から成るリストがインスタンシエートされると goal の実行がそのリストの内容によって制御されるようになる。

⑥ Directed Relations and Inversion of Prolog Programs

Y. Shoham, D. V. McDermott (Yale Univ.米国)

Prologの述語は，理想的には，引数間の関係を述べているだけで，入出力の方向性は持っていない。しかし，実際には数値演算や大小比較が行なわれるので，そこに出現する変数間の方向性を持った関係を表わしていると考えるべきである。関数はこの特殊な例で，方向性が一意に定まるものである。"完全な"関係は変数間のあらゆる方向性の組みあわせを含むものであると考えられる。

本論文は以上のような，Prologの現実的見方に基づいて書かれている。方向性の拡張，特に関数の方向を機械的に逆転する方法について述べている。関数を逆転するためのアルゴリズムと，そのあまり単純でない例（assertやretractを含むもの）に対する適用例，アルゴリズムの限界などが示されている。

最後に，インタラクティブに計算木を辿るシステムが紹介されている。これは計算のシンボリック・トレースを行なっているものと考えられる。

⑦ Efficient Stream/Array Processing in Logic Programming Languages

上田　和紀(日本電気)
近山　　隆 (ICOT)

Concurrent Prologにおけるn本のストリームのマージはコストの高い計算であると考えられてきたが，ここでは効率の良いコードを得るためのコンパイル技法が示されている。本技法に従うと，データの転送はnによらない遅れで可能である。更に，ストリームの追加，削除が平均$\theta(1)$で可能なことが示されている。入力をn本の出力に分配する述語も同様にn本マージとして実現される。

このコンパイル技法は更に配列の実現に応用でき，配列要素のコンスタント時間の参照と更新が可能になる。

上記の効率は高度なコンパイラによって始めて可能になるものであるが，システムが対応するコードを直接サポートすることによって，ソースプログラムの量とコンパイル時間を減少させることができる。

⑧ What is a Variable in Prolog?

中島　秀之、戸村　　啓（電子技術総合研究所）
上田　和紀（日本電気）

Prolog における変数の取り扱いについての見直しが行なわれる。現存するProlog のプリミティブだけでは，変数をデータ・オブジェクトとして扱うのに不足である。本論文は二つの新しい概念：freeze と melt を導入することにより，この解決を計るものである。同時に，これまでこの機能を暗黙裏に用いていたシステム述語（入出力やデータベース操作）の見直しをし，これらの機能が分離される。

プリミティブの見直しにより，prolog の変数や節をデータとして扱う能力が向上する。これにより，より高度のデバッガ，非局所論理変数，効率の良いメタ推論などが可能になる。これらの例が示される。

最後に，freeze と melt の効率の良い実現法が示される。特に melt に関しては，項の大きさによらないコンスタント時間での実行が可能である。

⑨ A Note on the Set Abstraction in Logic Programming Language

横森　　貴（富士通）

ラムダ計算におけるラムダ・アブストラクションとの類推により，セット・アブストラクションの概念が示される。このセット・アブストラクションはprologの言語仕様の“集合表現”と“述語変数”に関する拡張に関するものである。Warren により，集合表現は言語の表現力を高めるものであるが，述語変数はprologの能力を高めるものではないことが既に示されている。

本論文では，これら二つの概念を統合して，セット・アブストラクションを，述語変数がデータとして許されるような集合表現として定義している。従って，セット・アブストラクションは高次述語論理に関するものとなる。述語変数とenumerateという述語を導入することによりセット・アブストラクションは2回述語論理を扱うことができることが示されている。また，Prologや Concurrent Prolog による実現プログラムが示されている。

⑩ RF-Maple: A Logic Programming Language with Functions, Types and Concurrency

P.J. Voda, B. Yu (Univ. of British Columbia,カナダ)

関数型プログラムを論理型プログラムと結合すると，(1)関数型プログラムの関数を組合わせて全体を構成できる点と(2)論理型プログラムの関係によって非決定性を表現できる点の両方が得られる。

言語 RF－Maple は論理型プログラミング・スタイルを関数型プログラミング・スタイルとを結合したものである。〝ＲＦ〟とは〝Relational and Functional〟を表わしている。ＲＦ－Maple は別々に設計された二つの言語，関係型プログラミング言語Ｒ－Maple と関数型プログラミング言語Ｆ－Maple ，が互いに独立に一体化したものである。

Ｒ－Maple は関係に基づく論理型プログラミング言語であり，並列動作を表現できる。論理的意味を失うことなく，実行に関する制御ができることを目的にしている。プログラムの逐次実行およびパラレル実行を Concurrent Prolog よりも細かく指定することができる。

Ｒ－Maple は存在限量子に相当する機能を持ち，否定を表明できる。その結果として，Prolog のcut やConcurrent －Prolog の commit はプログラムの宣言的解釈ができないのに対し，Ｒ－Maple では，プログラムが停止した場合には，プログラムの宣言的解釈は成立する。

Ｆ－Maple は型付きの簡単な関数型プログラミング言語であり，４つの構成要素しかない。またＦ－Maple は構文規則を拡張できる言語であり，プログラマは型の構文と関数の構文を定義することができる。

Ｒ－Maple とＦ－Maple の二つの概念を結合してＲＦ－Maple をつくることで，プログラムの読み易さと実行速度とが改善された。実行速度が改善されるのは，関係の多くが関数的であり，従ってバックトラックを必要としないためである。

⑪ The Compilation of Prolog Programs without the Use of a Prolog Compiler

K.M. Kahn, M. Carlsson (Uppsala Univ., スウェーデン)

本論文では，まずLispで書いた効率のよいPrologインタプリンタを作り，次にこのインタプリタを個別のPrologの述語の実行専用に特殊化したインタプリタをつくる方法について述べる。この特殊化は，〝Partial Lisp〟と呼ぶLispプログラムに対する部分評価系（Partial evaluator）が自動的に行なう。〝Partial Lisp〟はLispプログラムを同じ意味を持つ他のLispプログラムに変換するが，Prologについては何も知らない。

まず，インタプリタを部分評価することはコンパイレイションに代わることについて論ずる。次に簡単なPrologの述語に対してPrologのインタプリタを部分評価した結果を示す。特殊化したインタプリタの実行速度は通常のインタプリタによる実行速度に比べて約十倍速くなっている。また，この速度はPrologの最適化コンパイラが生成するコードの実行速度に匹敵する。

インタプリタを部分評価をすることの利点は，Prologのインプリメンテーションをかなり小さくすること，またインプリメンテーションの変更を容易にすること等である。他方，この方法の主な問題点は，この方式では各述語ごとに小さな特殊化したインタプリタを生成することになるが，現時点ではこの部分評価による特殊化したインタプリタを生成するのにかかる時間が，最適化コンパイラがコンパイルするの場合の時間に比べて二桁程度長い時間がかかる点である。この部分評価にかかる時間を短縮する方法についてふれる。また一つのPrologの述語の使い方を限定した場合にインタプリタをさらに特殊化する可能性について検討する。

⑫ Two-Level PROLOG

A. Porto (Univ. Nova de Lisboa, ポルトガル)

Prologに対して代わるものの一つとしてTwo-level Prologを導入する。

Two-level Prologの特徴は節がheadとbodyとconditionとを持つことである。この節は二つのレレベルで解釈できる。オブジェクトレベルではconditionとbodyの連言からheadへの含意として解釈する。またメタルレベルではconditionからメタ述語〝execution step〟への含意として解釈する。〝execution step〟は引数としてheadとbodyを持つのである。

含意には，両方のレベル共に普通の含意と排他的含意との二つの種類がある。排他的含意を用いると，Prologのcutと同じ効果を得ることができる。

Two-level PrologはPrologの上に実現されている。実現方法としては，Two-level Prologでの節を，オブジェクトレベルの解釈とメタルレベルの解釈に対応する二つのPrologの節に変換する方法を用いている。Two-level PrologからPrologへの変換方法を示す。

Two-level PrologはPrologよりも信頼できるプログラミングスタイルをもたらし，インタプリタを書くのに適しているを，例題によって示す。

⑬ Metacontrol of Logic Programs in METALOG

M. Dincbas, J-P le Pape (CNET,フランス)

本論文では効率のよいMETALOG の新しい実現方法を示す。 METALOG とはHorn 節に基づく論理プログラミングにメタレベルの実行順序に関する制御を表現する能力を提供するものである。

METALOG では，実行順序の制御を指定する論理プログラムを，オブジェクトレベルの知識の使い方を表現するメタ知識と見る。メタレベルで制御の情報を書くことで，ユーザは推論の過程を調整したり，推論の戦略を定義することで，インタプリタを指定することができる。この理由からMETALOG はmeta−PROLOG と見ることができる。

二つのレベルの知識はHorn 節の形で書く：オブジェクトレベルの知識は通常の節として書き，メタレベルの知識はメタ節として書く。

METALOG は制御のための述語を提供する。これらの述語はメタ節で使い，戦略と発見的方法とをメタインタプリタに指示する。

次にMETALOG の持つ，推論過程の制御に関する強力な表現力を例を用いて示す。また例題についてMETALOG の処理能力を示し，標準のPROLOG インタプリタと比較する。

これらの例からわかるように，オブジェクトレベルとメタレベルとを組合わせることによって，他の方法に比べて自然でかつ強力な制御に関する表現能力をシステムが持つことができる。また，プログラムの実行も効率的になる。