2.2 一般論文（要旨）

以下の本論文は、FGCS'84予稿集(英文)に収録されている。

(1) 論理型プログラム言語の基礎理論

① Some Practical Properties of Logic Programming Interpreters

D.R. Brough (Imperial College, 英国)
A. Walker (IBM San Jose, 米国)

この論文では Prolog interpreter とそれを改良したinterpreter の性質が論じられる。

まず，SKB （Simple Knowledge Base）と呼ばれる特別な形のProlog programが定義され（2節）以下ＳＫＢ上でのinterpreter のふるまいについて考察される。ＳＫＢとは，述語の引数は定数か 変数であり，fact はground であり，ruleの頭部に現われる変数は本体にも表われる，という制限が加わったProlog program である。この時,質問に対する答は有限個のground の literal であり，それを計算するアルゴリズムは存在する（Theorem 4.1）。

次にSKB 上での preorder interpreter が定義される（2節）。Preorder interpreter とはtop down left-to-right depth first のinterpreter で，goal とrule の頭部が uuifyすれば rule を適用するが，goal と rule のstack の状態によっては適用をやめる。stack の状態によらないinterpreter（即ち標準のProlog interpreter ）を，Ｉｏ ， 現在のgoal と同じgoal がstack にある時適用をやめるinterpreter を I_G，現在適用しようとしているruleの instance が stack にある時適用をやめるinterpreter を I_R とする。I_G もI_R もsound であり（Theorem 4.2），I_G もI_R もＩｏよりよい（Theorem 4.3， 4.5）がI_G とI_R が互いに他よりよいinterpreter とはいえない（3節の例)。そして，SKB のすべてのprogramに対して答を見つける事のできるpreorder interpreter は存在しない事が示される（Theorem 4.7)。

② Qute: A Functional Language Based on Unification

佐藤雅彦，桜井貴文（東京大学）

この論文では，プログラミング言語Qute が定義される。

Qute は Prolog 及び大部分の論理型言語の特徴である unification を取り入れた関数型言語である。即ち大部分の関数型言語が変数に値を与える際(例えば，引数の受け渡し)一方向の代入によるのに対し，Qute は unification によっている。（Unification の厳密な定義は 3.1 節。）

更に，Qute の program は並列に評価する事ができる。Qute の program は expression （2 節）と呼ばれ，その subexpression のうち書き換え可能な部分（evaluable subexpression と呼ばれる; 4.1 節）を書き換えてゆく事により評価が進む。その際同時に environment （変数の値を持っている; 3.2 節）も書き換えられる。並列な and等のため一般には複数個ある evaluable subexpression を同時にどの様な順序で書き換えても結果は同一である事が保証されている。そのため negation と if-then-else の定義は，unification を取り入れた事と相まって普通の言語における場合とは異なる。即ちある条件（変数が〝十分に〟 instantiate される）が満たされるまでは，negation や if-then-elseは書き換えられない（4.2 節)。

評価は，expression が pattern （3.1 節）と呼ばれる値に書き換わった時または書き換えが fail した時（prolog の fail と同様の概念）または suspend した時（Concurrent prolog の suspendと同様の概念で，negation が if−then−else のため起こる）に終わる。

最後にQute の厳密な定義が書き換え規則の形で与えられている（5 節）。

③ Incidence Calculus: A Mechanism for Probabilistic Reasoning

A. Bundy (Univ. of Edinburgh, 英国)

この論文ではあいまいさを扱うための incidence calculus が提案される。

あいまいさを扱う論理体系や expert system のうち大部分のものにおいては，公理に確率（それが成立つ確からしさ）を与えそれから導かれる式が成立つ確率を計算する，という方式をとっている。しかし確率のみを使って計算すると２つの式の独立性を扱いにくく，たとえ相関係数を導入しても（３節）不都合が起こる（６節）。

そこで式に確率を与える代わりにそれが成立つ様な世界（incident）の集合（incidence）を与え，それらの式の negation, and, or 等に与えられるべき incidence を計算する規則 incidence calculus を導入する。これは predicate（propositional）logic の解釈を与える事に相当する。

Incidence calculus では logic の推論規則の前提の incidence から結論の incidence を得る方法は与えられず，一般に結論のincidence は前提の incidence を含むという事しか言えない。従ってある式が証明された時その式のincidence の下界しか計算できない。証明によって得られる下界は異なるのでそれぞれの union をとる事によりより精密な下界を計算できる。そして上界はその式の negation の下界の補集合である。（７節）

いくつかの式に incidence を与えた時それらが矛盾しているかどうか判定するアルゴリズムが与えられるが，これは incompleteである。predicate incidence calculus の場合 predicate logic を含むので必然的に incomplete になる。（８節）

Expert system に incidence calculus を採用する場合ユーザが incidence を指定するのは困難なので確率から incidence の割り当てを行う方法が考察されるが，まだ問題点が多い。（９節）

④ A Theory of Complete Logic Programs with Equality

J. Jaffar (Monash Univ., オーストラリア)
J.L. Lassez, M.J. Maher (Univ. of Melbourne, オーストラリア)

この論文では unification を一般化した logic program の性質が論じられる。

Logic program とは definite clause logic program P と definite clause equality theory E の組（P，E）である。即ち，標準的な unification の代わりにE から導かれるequality を使った一般的な unification を用いた logic program を考えるのである。（P，E）に対し Herbrand universe に相当する domain が存在し（Theorem 1），interpretation が定義できる。（P，E）に関する derivation もEがない場合と同様に定義でき，success set SS (P, E)，finite failure set FF (P, E)，general failure set GF (P, E) が定義される。FF（P，E），GF（P，E）はいずれもすべての derivation が有限失敗する ground atom の集合であるが，FF（P，E）は derivation の長さがある自然数で押えられるという制限がある。また標準的な場合と同様に interpretation 間の関数 T(P, E) が定義でき同様な結果：(P, E) の最小モデルは T(P, E) の最小不動点に等しい (Theorem 2)，ground atom $P(\tilde{t})$ について (P, E) $FP(\tilde{t})$ iff $P(\tilde{t}) \in SS(P,E)$ (Theorem 3)，$P(\tilde{t}) \in FF(P,E)$ iff $P(\tilde{t}) \in T(P,E) \downarrow W$ (Theorem 4) が成立つ。

Complete logic program とは augmented definite clause logic program P * と unification complete equality theory E* の組 (P*, E*) である。P* はいわゆる complete program であり，E* は下雑把に言って等式 S = t が正しい事と s と t が unifiable である事が同値である様な equality theory である。(P*, E*) に対応する logic program を（P，E）とすると 2 つの重要な結果：成功する derivatiotn の soundness と completeness 即ち ground atom $P(\tilde{t})$ に対し (P*, E*) に $P(\tilde{t})$ iff $P(\tilde{t}) \in SS(P,E)$ (Theorem 5) 及び negation-as-failure の soundness と completeness 即ち ground atom $P(\tilde{t})$ に対し (P*, E*) にて $P(\tilde{t})$ iff $P(\tilde{t}) \in GF(P,E)$ (Theorem 6) を得る。

⑤ A Program Transformation from Equational Programs into Logic Programs

富樫敦、野口正一（東北大学）

この論文ではequational programからlogic programへの変換が論じられる。

Equational programは項書き換えシステムとして定義され書き換え規則は F（E₁，…En）→E^{n+1}という形をしている。但し,function symbol は constructor か defined function symbolでありFは defined function symbol,E₁ …E^{n+1} は termである（Definition 2）。特にE₁ ……En が function symbol がすべて constructor である termの時 recursive であるという（Definition 3）。以下では term をequationald program によって書き換える時 primitive execution strategy と呼ばれるいわゆる最内戦略に相当する戦略に従うとする。

Logic programとは cluster sequent と呼ばれる式の集合として定義される（Definition 7）。Cluster sequent とは左辺に複数個のatomを許す様拡張されたHorn節でMi:－N（M, Nはatomの集合でclusterと呼ばれる）という形で表わされる。更に2種類の変数 fixed variable と inferred variable があり，Nの fixed variable はMの fixed variable に含まれ，Mの fixed variable を X₁ …Xk，Mの fixed variable を Y₁ …Yp，Nの fixed variable を Z₁ …Zqとすればcluster sequent の意味は∀X₁ …Xk（∃Z₁ …ZqN⊃∃Y₁…YpM）である。従ってgoalに cluster sequent を適用するためにはcluster sequent の左辺の fixed variable 及びgoal の inferred variable に代入を行って同じclusterにならなければならない（Definition 8）。

この時equational programからlogic programへの変換が自然に定義でき（5節Algorithm A B），特に recursive equational programからはHorn programが得られる（Proposition 4）。そしてequation programにおける書き換えが可能なら変換されてできたlogic programでの実行が可能である事（Theorem 1）及びrecursive の時はその逆も成立つ事（Theorem 4）が示される。

⑥ Transformational Lgoic Program Synthesis

佐藤　泰介（電子技術総合研究所）
玉木　久夫（茨城大学）

一階述語論理で書かれた仕様を満たす論理プログラムの新しい合成法について述べている。仕様φ(χ)は基本述語（論理プログラムが既にあって，それにより計算（定義）されている述語）の組み合せであり，最小モデルによりφが定義する関係が定っているが，この関係を計算する論理プログラムを，プログラムの等価変換システムを使って合成する。

一番困難な部分は，Ayφ（χ，y）の形の仕様に対する合成であるが，これはまずEyLφ（χ，y）を満たす論理プログラムを合成し，次にこの合成されたプログラムを〝手続き的に否定〟（Negation technique）することにより得られる。Negation technique を適用する為にはプログラムがある形をしていなければならないが，その形にする為，論理プログラムの等価変換システムが使われる。この変換部分は非決定的であり，常に成功するとは限らない。一方Negation technique 自体は決定性のアルゴリズムである。

合成は演繹的手段，特に帰納法なしに行なわれ，合成されたプログラムと共にそれが仕様を満たす条件が同時に得られる。例としてN-queens問題が扱われている。

⑦ Efficient Unification with Infinite Terms in Logic Programming

A. Martellic, G. Rossi (Univ. di Torino, イタリア)

有理木（rational tree，部分木が有限個しかない木）の単一化アルゴリズムの提案である。木は{χ1=t1，……，χn=tn}（χi は変数，ti は項）の形で表わす。単一化問題は{t1=s1，…，tR=sR}の形の等式群を同時に満たすような変数への有限木の最も一般的な代入を見つける事で，その為この群を２つの操作〝compaction〟と〝reduction〟により変形して簡単化し（途中で失敗がなければ）{χ1=t1，…，χn=tn｜但しχi はすべて相異なる}という形にすると，これが単一化子になっている。

〝compaction〟は，例えばχ，y，z，を変数として，χ=y=t，y=z=s という２つの等式があった時，これらをχ=y=t=s の形にまとめる操作であり，〝reduction〟は例えばz=f（χ，g(y)）=f（g(y)，y）なる等式があった時，これを共通部分 z=f（χ，y），及び残りの{χ=g(y)，y=g(y)}に分ける操作である。２つの操作は無限回行うことが不可能なので，このアルゴリズムの停止性が保証される。計算のオーダは殆んどリニアである（Ackerman 関数の逆関数＋リニア）。

⑧ Automatic Implementation of Abstract Data Types Specified by the Logic Programming Language

N. Heck, J. Avenhaus (Univ. of Kaiserslautern, 西独)

抽象データ型のインプリメンテーションについて述べている。データタイプPが台集合とその上の演算（関数）の組として与えられている。台も演算も論理プログラムで記述されている（等号理論で台を割ることはない）。問題は，Pを仕様と見て，より具体的なデータタイプqによりPの仕様を満たすものを実現することである。その為qの台Dqからの台Dpへの全射関数を用意し（これも論理プログラムで書く），Dp上の演算（関数，関係）をDq上の演算に引き戻せば良いが，その引き戻し方を論じている。

これは「data structure mapping」に他ならないが，抽象データタイプを規定する論理プログラムのsyntacticな形を制限することにより，常にこのような引き戻しが可能であることを示している。又Dq上の演算に引き戻したとしても，それを表わす論理プログラムの能率が良いとは限らず，その為プログラム変換を行うが，それはunfolding，ある種の節のsubsumption及び述語の名前替えの3種で，folding操作は入っていないようである。

⑨ Programs as Executable Predicates

C.A.R. Hoare, A.W. Roscoe (Oxford Univ., 英国)

並行プロセス記述言語Occamの形式的な記述について述べている。プロセスは内部状態 "st" や，外部から入出力を観察した時のトレイス "tr" 等を使うことにより記述される。

例えば，stop プロセスは，

$$stop \triangleq st = waiting \wedge tr = \langle\rangle$$

と書かれる。これは状態が "waiting" で何も入出力がないことを言っている。

代入 $x := e$ は $(e \succ x \rightarrow \cdot)$ により，分岐は$P \lhd b \rhd Q$（it b then P else Q）により表わす。

入出力に関して，例えば，出力は，

$$c!e = (st = waiting \wedge C \leftarrow ref) \langle tr = \langle\rangle \rhd (tro = c.e \wedge (tr' \succ tr \rightarrow SKIP))$$

但し $tr = \langle tro \rangle \wedge tr'$

等と表わされる。これはトレイスが$\langle\ \rangle$の時は何も出力されず，トレイスが$tr = \langle tro \rangle \wedge tr'$即ち，troとtr'の連結である時は，troが1つの単位出力で，それはチャネルCにeの値を出力した事を表わしている。

このようにして，以下入力，recursion，プロセスの連結（;），並行（PAR），選択（ALT）の記述が与えられ，Occamの subset の意味が記述される。

⑩ Logical Derivation of a Prolog Interpreter

渕　一博（ICOT）

本論文では，主制御構造に論理式の変換を含むようなprologインタプリタの導出法について述べている。prologインタプリタの仕様の中で用いる基本述語に、「継続（continuation）」を表わす変数を加える。この変数の導入によって，ＡＮＤ演算子を消去することができる。ＯＲ演算子も同様に消去できる。

このＡＮＤとＯＲの消去の副次効果として，２種のスタックを持った逐次型のインタプリタに対応する論理式を導くことができる。このインタプリタにおいては，テイルリカージョンの最適化技法が簡単かつ自然に導入できる。導出されたプログラムは，通常のif−then−else文に似た要素からなるので，普通のプログラム言語に容易に変換することができる。

他に提案されているprologインタプリタの導出法と較べて，この方法は，はるかに簡潔である。この簡潔さの１つの大きな要因は，callスタックおよびbacktrackスタックと呼ばれる２つのスタック上の操作にあるといえる。

⑪ On Parallel Computational Complexity of Unification

安浦　寛人（京都大学）

一階述語論理の単一化（ユニフィケーション）の並列計算モデル上での計算の複雑さを述べた論文である。並列計算モデルとしては，最も安定なモデルである組合せ論理回路を用いている。まず，このモデルの上で新しい並列ユニフィケーションアルゴリズムを与え，その計算時間を調べている。入力される項の長さをn，その中に含まれる異なる変数の数をn′とする時，$O((\log n)^2 + n' \log n')$ の計算時間がかかることを示している。このアルゴリズムは，単一化をハイパーグラフ上の到達可能性判定問題に帰着して解いている。

次に，単一化の並列計算時間の下界について議論している。ここでは，単一化が，多項式オーダの素子数（または多項式時間の逐次型アルゴリズム）で解ける問題のクラスの中で最も困難なものであることを示している。すなわち，単一化は，このクラスに対し，log 一段数完全であることを示している。現在のところ，このような問題に対して $O((\log n)^k)$ の計算時間を持つ並列アルゴリズムは作れないであろうと予想されている。よって，単一化を並列処理によって一般的に大幅に高速化するのは難しいであろうと結論している。

⑫ Database Updates in Pure PROLOG

D.S. Warren (State Univ. of New York, 米国)

Prolog の assert オペレータは，非論理的であることで有名である。この assert を使って，プログラマは prolog できわめて手続き的なプログラムを書くことができるが，プログラムが宣言的であるという prolog 本来の大きな長所を失うことにもなる。これに対し，本論文では，多くの assert の使い方に対してこれに代わることができる新しいオペレータとして、assume を導入する。assume の大きな特徴は，宣言的なセマンティクスを与えることができる点である。Prolog のプログラムは一階述語論理の文であり，その実行は一階述語論理上の演繹となる。これに対し，assume を含む純粋 Prolog プログラムは，一種の様相論理の中の文となり，その実行はその様相論理上の演繹となる。これは，関係データベースにおける更新操作の理論を与えていると見ることもできる。この理論によって，データベースにおけるある種の空値の役割をよく理解することができる。

⑬ DAL-A Logic for Data Analysis

L. FARINAS-DEL CERRO (Univ. Paul Sabatier, フランス),
E. ORLOWSKA (Academy of Science, ポーランド)

この論文は，論理的な手法でデータ解析を行う方法を提案している。ここでは，古典的な論理型プログラミングの枠組に従い，データ解析問題を表わす論理型言語を定義し，この問題の解を与える証明系のための推論システムを作っている。

データ解析は，データの集合の中からパターンを抽出する作業である。著者らは，これを次のような2つの部分に分けて考えている。

(1) データをその性質によって分類する。

(2) 各分類を特徴付ける性質を決める。

分類されたデータ集合は，各々論理型言語ＤＡＬの論理式によって表わされ，その性質はその言語の上の関係式として表現される。

ＤＡＬの推論規則に従って，上記(1)と(2)を証明の形で実行する。