1982年 ACM LISP and FP Conference の報告

瀧 和男、 近山 隆、 安川 秀樹、 井田 哲雄
（Institute for New Generation Computer Technology）（理化学研究所）

1982 ACM Symposium on Lisp and Functional Programming が 8月15日から18日までの間、米国ピッツバーグのカーネギーメロン大学で開催された。

今年の発表は32件でLisp関係が約 4分の 1と少なく、CombinatorやFPの話が約半分、PROLOGの話が 4編ほどであった。FP関係では研究上問題とされるところが、ひととおりカバーされており、Combinator関係では、新規な提案がいくつか見られ、新たな研究材料を提供している。Lispに関してはimplementationに関する話題が多数を占め、研究的色彩を持つものが少ないが、MAC Lisp系の新しい標準LispとしてCommon Lisp の“お広め”があった。PROLOG関係はまだ研究の始まりといった感じが強く、多くの異なるアプローチの研究が可能であろう。以下では、発表内容の簡単な要約および若干のコメントをまとめる。

なお、当日は隣接するAAAIの会場ではLisp machineのデモがあり、Symbolics からはLM-2と3600(CPUのみでDiskは未接続）、Xerox からは門外不出だったDoradoと、Dolphin,Dandelion(Starのハードウェアに Inter Lispをのせたもの）が出典された。特にDoradoは、その高速性を生かしたウィンドーシステムのデモが人目を引いていた。（T, I）

——— 発表の要約 ———

1. Programing with Infinite Data Structures

招 待 講 演

David Turner(University of Conterbury)

ターナの設計した言語KRC(Kent Recursive Calculator)による関数的プログラムの例を示し、関数的プログラムの有効性について説明した。ターナの論点は次のとおりである。

① 関数的プログラムは“referential transparency”が成立するため、プログラムを厳密に記述できるのみならず、その意味を容易に推理・論証することができる。

② LISPは関数的プログラム言語にもっとも近く、しかも比較的普及している言語である。しかし、①のポイントに照らして、シンタックスのよくないPROG,SETQ,RPLACA等の関数の導入によって、referencial transparencyが成立しない等の問題をもっており、真に関数的言語ではない。

この認識にたってターナは KRCを設計した。 KRCの特徴は

(a) purely functional
(b) higher order functionを取り扱う。
(c) fully lazy である。…したがって無限データ構造を取り扱うことができる。
(d) set abstractionの機能を持つ。

である。

KRC ではプログラムは non-stric higher-order recursive equationであらわされる。

例. perms [ ] = [[ ]]
perms x = {a:p|a←x;p←perms(x--[a])}

さらにZF set abstractionの機能を通常のラムダ計算の体系に加えることにより、言語の表現力が大きく増加することを、いくつかのプログラムによって示した。（I）

2. Super Combinators: a New Implementation Method for Applicative Languages

R.J.M. Hughes (Oxford Univ.)

コンビネータをマシン語とするアプローチは、ターナの SASL(ST. Andrews Static Language)が始めであるとされている。ターナはコンビネータとしてS.K.I.を基本とし、さらに最適化のため、いくつかのコンビネータを追加している。

本論文では、関数的プログラムをコンビネータへと翻訳するアプローチをとっているという意味では、ターナの研究の流れをくむものであるが、コンビネータをいかに選ぶかという点で新たな発想がみられ、非常に興味深い論文である。ヒューズはコンビネータとして、ラムダ変数列と自由変数を含まないapplicative formの結合で表わされる ラムダ表現をすべて、コンビネータであるとみなし、これをスーパーコンビネータと名付けている。このように作られるコンビネータを基本マシン語とし、グラフリダクション方式で、ラムダ表現をリダクションしていくことによって計算は実行される。ヒューズはさらに、最適化のいくつかの手法をも提案している。(I)

3. A Fixed-Program Machine for Computer Expression Evaluation

Steven S.Muchnick(University of California, Berkeley)

Neil D.Jones(Aarhus University)

本論文もターナの研究を一歩進めたものである。関数プログラムをコンビネータに変換したあと、さらにコンビネータ表現をスタックマシンコードにコンパイルする手法について述べている。(I)

4. Expressions as Processes

J.R.Kennaway and M.R.Sleep(University of East Anglia)

Sleep らのグループのZAPP(Zero Assignment Parallel Processors) プロジェクトに関連した研究発表。

ZAPPでは多数のプロセッサーを論理的に木状(物理的には一種のcubic ネットワーク)に結合したシステムの構築を考えている。この木状ネットをMilnerのＣＣＳに基づいて、定式化し、(LNETと呼んでいる)さらにＬＮＥＴにコンビネータを分配する手法について述べている。(I)

5. Combinatory Foundation of Functional Programming

Corrado Böhn(University of Rome)

著者のBöhmはラムダ計算法の研究者として、有名な人物である。このコンビネータ論理の体系はいくつかのよく知られた代数的性質をもつため、ＦＰのプログラムを代数的に取り扱う上でもBöhmの手法は強力な手法になりうる。

バッカスの提唱する関数的プログラムシステムＦＰとＦＦＰ(FPはユーザインターフェイス言語で、FFP により基礎付けられる。言語のexpressive powerはFFP のほうがある)の一部を６つの基本的コンビネータをもつコンビネータ論理の体系に埋めこむ手法を提案している。

Böhmは本論文の冒頭にＦＰはＦＰ向き計算機アーキテクチャが開発され、しかもＦＰプログラム代数の手法をさらに進歩させることができれば、ＦＰはプログラミングコミュニティーに多大な影響を与えることができるであろうと述べている。Böhmの提案は後者の研究を一歩すすめるものである。(I)

6. Functional Specifications of a Text Editor

Gary Feldman(Carnegie-Mellon University)

ＦＰをエディタの機能・仕様記述言語として用いた時の使用経験について述べている。本論文では、実際のＦＰによるエディタ仕様記述プログラムを示すとともに、ＦＰの表現能力についてもコメントしている。（Ｉ）

7. Experiments with a Supercompiler

Gary Feldman(Carnegie-Mellon University)

Target Language の定義をinterpretiveに行ない、その定義を用いてsource programのpartial execution プロセスを初期化するものである。"basic configuration"を考え、ソースプログラムの構造の深い変換を行ない、basic configuration によってターゲットプログラムを表現し、生成する。細かい内容についてはよくわからない部分もあるが、partial evaluationの応用例と考えられる。（Ｙ）

8. A Parallel Prolog: the construction of a data driven model

Michael J. Wise(University of New South Wales)

Prologライクの言語のmulti-processor machine アーキテクチャ上でのdata driven execution を行なうシステムEPILOGについて述べている。言語仕様は現在のPrologのコントロール・メカニズムに変更・拡張を施こしたものである。clauseへのinstantiation を "dframe"(clauseのコピーとその変数のbinding 情報から成るデータ構造）で表わし、プロセス間通信をパケットで行なう。パケットのphysical processorへの到着によってのみdframeへの新たなinstantiation が行なわれ、unification が行なわれる。実行結果はパケットにより、親のdframeに戻されるという 制御メカニズムによっている。

また、カット・シンボルは除去され、２階述語ＣＡＮＤ、ＣＯＲ、ＮＯＴ、を導入し、カットに相当する機能を吸収している。論理型言語の並列実行メカニズムとして自然なものと考えられ、参考になる部分もある。（Ｙ）

9. How to Define a Language using PROLOG

Christopher D.S.Hoss(Imperial College)

Colmerauerの提唱したMetamorphosis Grammar を用いて、Prologでどのように言語の定義や認識を行なうかという方法論についてサーベイしている。

言語をLexical, Symtax, Semanticsの３つのレベルから成るものとして捉え、各レベルでの言語の仕様定義の例を示している。この論文で述べられていることは、筆者も示しているとおり過去のいろいろな研究の成果の統合や別の解釈と言えるだろう。（Ｙ）

10. Logic Enhancement: A Method for Extending Logic Programming Languages

Poul R. Eggert(University of California, Santa Barbara & System Dev. Corp.)

D. Val Schorre(System Development Corporation)

Logic enhancement を現行のprologにsyntax sugarを導入することにより、実現しようというものである。

具体的には、

modules
macros
functional notation
repetition
tracing
execution profiling
common bug, "lint", checking

を拡張機能としてPrologを用いてインプリメントしている。

syntax sugarは使い易さ、理解し易さ等の点で重要なものと思うが、システムの効率が気になるということも言え、そのあたりのtrade-off をどう捉えるかが問題だろう。（Y）

11. Im plemetation of Interlisp on the VAX

Ramond L. Bates, David Dyer and Johannes A. G. M. Koomen
(University of Southern California, Information Sciences Institute)

VAX のBerkeley UNIX上にInterlisp を移植した話。

特色は

- Interlisp 10 と完全にcompatible
- 他機への移植性を重視
- 巨大な記憶空間

にある、システムはアセンブリ語1000行、Cで1,1000行、Interlisp 83,000 行（うち16,000行が新規）からなる。（C）

12. ＰＳＬ：A Portable LISP System

Martin L. Griss, Eric Bensob and Gerald Q. Maguire, Jr. (University of Utah)

Standard LISP のPortable版ＰＳＬの解説。ＰＳＬは自分自身で書いたコンパイラ・システムで、コード生成のみが機械依存であるが、機械依存の最適化もパラメタ化して組込まれている。現在Dec20, VAX, 68000 で稼働中で、Cray-1, IBM-370 への移植も考慮中である。Dec 上ではMaclisp, Lisp 1.6と同程度、VAX 上ではFanz Lisp よりやや速いという高速性も備えている。（C）

13. An Overview of Common LISP

Guy L. Steele, Jr. (Carnegie- Mellon University)

MAC LISP系の新しい標準LISPとして開発を進めているCommon LISP について概要を述べているが、LISPの歴史的変遷やIntreLISP との違いに関する記述も興味深い。The Common LISP Group は主に、ＬＩＳＰ Machine LISP、 VAX NIL 、S=1 NIL 、 SPICE LISPの研究者が中心となって組織されており、ポータビリティと、MAC LISP系LISPの特長であるシステムソフトウェアの記述力の高さを主眼にした仕様が決められ、すでに次のようなシステムが構築されつつある。

LISP Machine(MIT, Symbolics, LMI)――on symbolics 3600 etc.,
Spice LISP(C-MU)――on PERQ 1A,
VAX Common LISP (DEC, C-MU),

VAX NIL (MIT),
TOPS-20 Coomon Lisp (Rutgers, DEC),
S-1 Lisp(Lawrence Livermore National Labo., C-MU) (T)

14. S-1 Common Lisp Implementation

Rodney A. Brooks(M.I.T.)
Richard P. Gabriel(Stanford University & Lawrence Livermore National Laboratory)
Guy L. Steele,Jr.(Carnegie-Mellon University)

S-1スーパコンピュータ上に作成中のCommon Lisp 処理系の概要を述べている。高等関数、行列計算、FFT などの組込関数やマルチプロセッサのサポートなどがスーパコンピュータらしい。言語の上ではmulti value の関数やlexically scopedの変数も導入している。(T)

15. T:A Dialect of Lisp or,LAMBDA :The Ultimate Software Tool

Jonathan A. Rees and Norman I. Adams IV(Yale University)

Scheme[Steele,Sussman] をベースにしたlexical scopeを持つ言語“T”の設計と、VAX 、68000 上にポータブルな形で処理系を実現したことの報告。Lispライクな言語でありながら、高いモジュラリティや従来型マシン上でも効率良く動くことなどの特長を有する。(T)

16. The Importance of Brevity 招 待 講 演

Alan J.Perlis(Yale University)

「関数的プログラムの基本はreferential transparencyにあるとすると、関数的プログラミングは、計算におけるstate の変化をパラメータの引き渡しの過程にとじこめ、プログラマーからは、state 変化をみせないようにするプログラムのスタイルであると考えられる。」(井田の見解)

これに対して、パーレスは、state の変化の履歴こそ，計算というものの本質であることを指摘した。そして、incremental evaluationの手法を提示した。

proceedingには、パーレスの論文は掲載されていないため、正確なincremental evaluationの内容は不明である。(I)

17. Performance of Lisp Systems

Richard P. Gabriel(Stanford SUniversity & Lawrence Livermore National Laboratory)
Larry M.Masinter(Xerox Palo Alto Research Ctr.)

Lispシステムの性能比較の仕方について、どのような点に気をつけて行なうべきかを論じている。現在利用されている諸システムで用いられている様々の実現手法の特性を分析し、ベンチマークテストは、Lispシステムの種々の側面や、ハードウェアについて詳細に解析した上で行なうなら有意義であり得る、と結論している。
(Y)

18. Prolog Compared with LISP

Claudio Gutierrez(The University of Costa Rica)

LISPとPROLOGのパフォーマンスの比較を、命題論理式の妥当性を検証するプログラムを題材として行なったものである。異なる言語間のパフォーマンス測定というものは、このように安易に捉えられるものなのかという疑問は生じるが、何らかの比較を行なわねばならないとすれば、1つのアプローチとして、ここで述べられているような方法も考えられると言える。

Prolog版のプログラムはLisp版の焼き直しとしか見えず、同じアルゴリズムを実現するとは言え、よりPrologに適した（効率の良い）インプリメンテーションをすべきではないだろうか。（Y）

19. An Investigation of the Relative Efficiencies of Combinators and Lambda Expressions

Simon L. Peyton Jones(Beale Electronic Systems Ltd.)

本論文はコンビネータによる関数プログラムシステムのインプリメンテーション手法の有効性の評価結果について述べている。検討した方法は、normal order λ-reduction, applicative order λ-reductionとコンビネータによる reductionである。対象としたプログラムは比較的小さいものであること、最適化によっては評価結果が変わることの前提条件の下に、コンビネータによる reductionが最も実行効率が良いと結論付けている。（I）

20. Improved Effectiveness from a Real Time LISP Garbage Collector

Jeffrey L. Dawson(Wang Laboratories, Inc.)

Henry BaKer流のIncremental Copying GCの改良版、コピーを両端から用いることによって、データの linearity を上げるようにしている。（C）

21. Garbage Callection and Task Deletion in Distributed Applicative Processing Systems

Paul Hudak and Robert M.Keller(University of Utah)

ユタ大学ケラーの設計したＦＧＬ（Functional Graph Language)のマシンモデル（木状に結合されたマルチプロセッサシステム）におけるカーベジコレクションとタスクの消去の問題を扱った論文。ＦＧＬはグラフリダクションに
より実行される。本論文で示されるアルゴリズムがどれだけ一般性をもつものかは明確でないが、マルチプロセッサーによるグラフリダクション・システムを考える上では、本論文で述べられる問題を避けては通れないであろう。（I）

22. Functional Geometry　　招待講演

Peter Henderson(Oxford University)

Maurits Escherの彫刻模様を関数的プログラミングによって生成する手法について述べた。聴衆はHenderson の雄弁なpresentationと次々と示されるEscherの模様に魅せられて、関数的プログラミングについて考えるのをしばし忘れた。（I）

23. A Scheme for Implementing Functional Values on a Stack Machine

M.P. Georgeff(Monash University)

引数としてfunctional valueを持つような関数の評価を行う場合に、直ちに評価にはいらずに、引数をnon-functionalな値に落とせるまで関数評価を延期する方法をとることにより、それをスタックマシン上に実現できることを示している。(T)

24. Constant Time Interpretation for Shallow-bound Variables in the Presence of Mixed SPECIAL/LOCAL Declarations

Jon L.White(M.I.T.)

Shallow binding のLispシステムでは変数のスコープについてコンパイラとインタプリタの規則に差がでてしまう。インタプリタの方が歩み寄って、この差を埋める高速の算法を提出した。この手法はVAX 上のLispシステムNIL で用いたものであるが、変数の個数によらず、一定の手間で変数を参照できるというshallow bindingの性格を保持している。(C)

25. Data Sharing in a FFP Machine

Gyula Mago(The University of North Carolina)

ＦＦＰマシンはバッカスのＦＦＰを並列的に実行するセル構造を持ったストリングリダクションマシンである。このＦＦＰマシンにおいて、データの共有をいかに行なうかについて、Peterson-Wegman のユニフィケーションのアルゴリズムを例にとって説明している。(I)

26. Design of a Lisp Machine-FLATS

E.Goto,T.Soma, N.Inada, T.Ida, and M.Idesawa (The Institute of Physical & Chemical Research)
E.Goto,K.Hiraki,M. Suzuki, K. Shimizu and B. Philipov (University of Tokyo)

ハッシングハードウェアを実装した 10 MIPS Lisp Mashine-FLATS の概要とLisp関数の実行時間が紹介されている。
国内では、理研シンポジウム「リスプ・マシン」(昭和５７年３月１９日 理科学研究所)の予稿集に詳細な記述がある。(T)

27. Highlights of the History of the Lambda-Calculus　　招待講演

Honoring Alonzo Church and Haskell B.Curry
J.Barkley Rosser(University of Wisconsin)

28. Teaching the Contorol of Complexity　　招待講演

Gerald J. Sussman(M.I.T.)

29. Nondeterministic Call by Need is Neither Lazy Nor by Name

William Clinger(Indiana University)

非決定性計算におけるパラメータの引き渡しの場合には、従来の計算法で使われているcall by value, by name, by need, by lazy というパラメータの引き渡し法を再検討する必要がある。可能な計算路をすべて最後まで追及するか否か、あるいは得られた答のいずれか1つ、または全部を選択するかといった新たな視点を加えて検討しなくてはならない。著者の得た結論は標題に明白に示されている。論文では簡単な言語の形式化をも新たな視点にたって行なっている。(I)

30. Toward an Algebra of Nondeterministic Programs

A. Toni Cohen and Thomas J. Myers(University of Delaware)

Nondeterministicな計算においては、"referential transparency"が失われるためにバッカスが示しているようなプログラムの代数規則が成立しない。本論文ではnondeterministicな場合における新たな代数系をバッカスのFPに沿って再構成している。(I)

31. A Semantic Model of Types for Applicative Languages

D.B.MacQueen and Ravi Scthi (Bell Laboratories, Murray Hill)

ラムダ計算法に基礎をもつapplicative なプログラミング言語におけるタイプ付けの問題を取り扱っている。(I)

32. The Semantics of Lazy(and Industrious)Evaluation

Robert Cartwright(Rice University)
James Donahue(Xerox Palo Alto Research Ctr.)

lazy evaluation の考え方は、その動作的な意味からは比較的容易に把握できる。そして、無限データ構造を扱う上で、あるいはマルチプロセッサーにおける計算の高速化法としても有効な手法であることが知られている。しかし、動作的意味から離れて、lazy evaluation の意味を定式化しようとすると困難な問題が多い。本論文はlazy evalua-tionの意味を理論的に取り扱おうというものである。(I)

[文献] Conference Record of the 1982 ACM Symposium on Lisp and Functional Programming, ACM order no, 552820.