報　告

パネル討論会

理論は実践を導けるか，実践は理論を生かせるか？†

第1回プログラミング—言語・基礎・実践—研究会報告*

パネリスト
理論代表：二木　厚吉[1]，　大堀　淳[2]，　柴山　悦哉[3]
実践代表：安村　通晃[4]，　竹内　郁雄[5]，　上田　和紀[6]
村井　純[7]
司会　萩谷　昌己[8]

鈴木則久所長あいさつ　皆さんようこそお出でいただきましてどうもありがとうございます．

プログラム言語の研究で一番昔に始まったのはコンパイラの研究です．一時コンパイラの研究が非常に理論的になり最近ではパーサについてはほとんど問題なく理論的につくれるようになっています．その後プログラム言語の研究には，理論的な研究もあり，実際的な研究もありましたが，しかしいまだにコンパイラは非常に重要な研究だと思います．

最近，レジスタのアロケーションとか，新しいRISC アーキテクチャとか，VLIW アーキテクチャとか，並列処理計算機とか，ベクターマシンとか，コンパイラ技術にも新しい側面がいろいろ出てきました．コンパイラ技術はコンピュータ産業の中ではお金という面で一番影響力が大きいんではないかと思います．最近のように100MFLOPS，100MHz というようなマイクロプロセッサが動き出すようになってきた原因の一つは半導体技術の進歩にありますけれども，そういうことをサポートできるようなアーキテクチャが考えられたし，コンパイラ技術も進んできていたからでしょう．

理論にはいろいろありますが，影響の多い理論も幾つかはあると思います．一方，functional programming とか，denotational semantics とか出てきましたが，これらはまだ今後の活躍を大いに期待しようというところかと思います（ニヤリ）．

10年ぐらい前にジョン・レイノルズとゼロックスでいろいろ討論したときに，やはり理論家がつくってこれでいいという言語と，実際にプラクティショナー(実践家)がこれで使えるという言語にはまだやはり少し開きがあると思いました．たとえば理論家は非常に小さなプリミティブから進めて，そのうちできれいな範囲でエクステンションを行います．一方プラクティショナーは実際に大規模システムをつくる，あるいは実際のハードウェアにマップすることを考えるわけです．たとえば並列処理のコンストラクトなんかをつくると，最初のうちは fork-join あるいは parbegin-parend のようなコンストラクトが入ってきて，それを普通の言語の中にどんと入れる．そうするとポインタが出てきてポインタをパラレルの中で自由に交換するプログラムをプラクティショナーは書くけれども，理論家はそんなことは書けない言語をつくるから，プラクティショナーは一切目を向けない．それが 10 年ぐらい前は近づいてきたかなと思っていたら，最近はますます離れだしています．1980年代ではオブジェクト指向言語とタイプカリキュラスの分野がプログラム言語の二つの大きな仕事だと思います．オブジェクト指向には理論的な影響はほとんどなかったわけですが，タイプのほうではかなり理論的な影響力がありました．その割には本当の実際のプラクティショナーが使う言語までにはその影響は及んでいません．

そういうところで，ぜひ日本の方々も大いに国際的に影響力のある仕事をどんどん進めていっていただきたいと思います．パネルの前に一つごあいさつさせていただきました．どうも．

総合司会　どうもありがとうございました．やはりパネリストとして参加していただきたかったものです．

† 日時　平成3年4月26日（金）14：30～17：00
場所　日本アイ・ビー・エム(株)東京基礎研究所Kビル
1) 電総研，2) 神電気，3) 龍谷大，4) 慶大，5) NTT
6) ICOT，7) 慶大，8) 京大
* ソフトウェア基礎論研究会とプログラミング言語研究会の統合

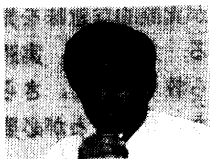
司会・萩谷　パネルの司会をやらせていただきます京都大学の萩谷です．

このパネルはソフトウェア基礎論とプログラミング言語の研究会の合同を記念したもので，理論と実践の関係について討論をしていただきたいと思います．

私はレフェリーをやらせていただきます萩谷です．理論代表の青コーナーは沖電気の大堀さん，龍谷大学の柴山さん，電総研の二木さんです．そして実践代表の赤コーナーは ICOT の上田さん，NTT の竹内さん，慶応大学の村井さん，そして慶応大学の安村先生です．私は一応中立的な立場なんですが，最終的には優勢なほうにつきたいと思っております(笑)．

パネルに関して私の意図したところは2点あります．1点は，この目的だけは果たせると思うんですが，理論派，実践派，それぞれの相手に対するフラストレーションの解消ということで，より分かりやすくいうと，日ごろのうっぷんを晴らすということです．2点目は，そのいろいろな議論の中で，西暦 2000 年ぐらいまでのプログラミングに関する研究の方向づけを行えたら大変ありがたいと思っています．

パネルの題と私の立場



安村　今日の私の部分の題を「プログラミングの理論と実践における自律と共生」という名前をつけたのにはわけがあります．

この「プログラミング―言語・基礎・実践―」という長くて舌をかみそうな研究会は，ソフトウェア基礎論とプログラミング言語の二つの研究会が合体してつくられたもので，いわばいまハネムーンの最中で，主査の私としましては，すぐに離婚というわけにいきませんので，なんとか1年でも2年でも私の任期の間は続けなければいけないという立場が一つございます．

もう一つは，今回のパネル全体の題は，私なら「実践は理論を導けるか．理論は実践を生かせるか．」という題にしたかった，ということがあります．要するに私は主査としての立場もあるんですけれども，やはり実践の立場で物を言いたいということです．

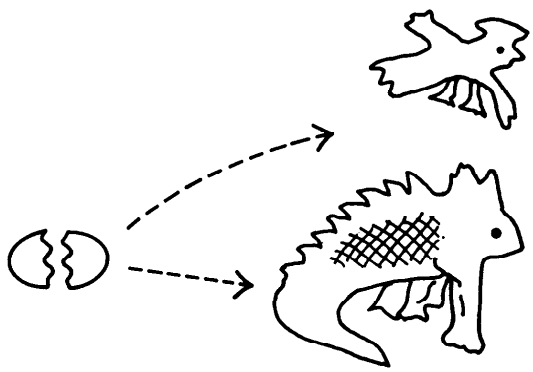
私はスーパコンピュータ用のベクトルコンパイラだとか，Lisp の最適化コンパイラの研究をしてきまして，それは呪縛として抜けられないようなものがあるということです．

理論と実践の定義

一般の科学の場合は，特に経験科学ですけれども，世の中に自然現象として起こっている現象，星の動きとか，洪水が起こるとか，そういったことに対して何か経験則，法則を見つけ出す．理論が起こるわけですね．一方，それを確かめるための実験科学というのがあると思います．それが普通の科学の場合で，そのために，たとえば物理ですと理論物理とか実験物理に分かれてくる．

コンピュータの場合，理論と実験といわず，理論と実践になってしまいますのは，要するに学問のアプローチが，一言でいってしまえば構成的といいますか，計算というものが基本になっている，という特殊性があるからです．ともかくプログラムをつくってみないことには話にならない，自然現象に相当することはプログラムをつくることであるということですね．

理論と実践の自律・共生モデル
- 理論が先か，実践が先か？
  →seeds or needs？
- 理論と実践の自律的成長の理由
  →
  —方法論の違い { 理論の役割 ・明確化 ・限界 ・新しい発見
  —評価法の違い　実践の役割
- 理論と実践の共生の必然性
  →衰退か，発展か
  理論の実践化
  ：
  実践の理論化



それに対してもちろん理論というものがあるわけです．じゃ，ここでいう理論とか実践とかはどういうことかというと，私が勝手にいうのも何ですから，辞典ではどういっているかを参考にしましょう．広辞苑の場合は，実践にひいき目な書き方をしているんです．まず，実践とは，

「実際に履行すること．実行．行為．働き．実際的適用．また，働き出された結果としての現実．実際．理論が―ここからすごいんですね―一生の抽象的自己否定，静止の局面において成り立つのに対し，この否定をさらに揚棄，否定の否定する生の本源的契機―すごいことが書いてある―したがって知と知の具体的媒介者として理論的真偽を検証するもの」

というふうに実践は理論の上を行っているという感じなんですね．じゃ理論のほうはどうなのか引いてみますと，

「1．単なる経験や個々の事実に関するばらばらの知識でなくて，それらを法則的，統一的に理解させるところの多少とも整合的な原理的認識の体系」

多少ともとかね．その次もすごいんですよ．

「2．実践的妥当性の検証を離れ，純観念的に組み上げられた論理．静観」

こういう感じになっているわけですね．

理論と実践の自律と共生

次に問題として，卵が先か，鶏が先かということで，理論が先に生まれるのか，実践が先に生まれるのか．こんなのはどちらが先でもいいという話もあります．要するにシーズが先か，ニーズが先か．最初にプログラムをつくるというか，実践が先にあって，それをやっていく過程で理論的な課題が生まれてくるんじゃないかという考えが一般的だと思いますが，予稿にも書いておきましたけれども，いろいろ考えていくと必ずしもそうではなく，純理論的なものが先にある場合もあるようで，これは一概にはいえません．

その次に，そうやって研究が始まってしまうと，途端に理論と実践が分かれて進み始めるわけです．何で分かれるかいろんな理由があるんですけれども，一つは目的とか方法論，評価が違う．要するに理論は物事の筋道をはっきりさせる必要がある．それに精力を使う必要がある．実践のほうはやったことが新しい効果とか実際的な意味をもたなければいけないということで，方法論的にも違う．評価もまったく違ってくるわけです．

当然技術とか知識が発達しますと，それは分かれてどんどん進まざるをえなくなる．そういうときに理論とか実践のそれぞれの役割は何かというと，理論のほうは物事を明確にしたり，限界を明らかにしたり，できれば新しい発見をそれから導いてくるのが望まれるわけですけれども，その辺はソフトウェア科学においてはどうだったかというのはよく考えてみる必要がある．最近の物理学でいいますと，実験では分からないけれども理論的に先に解明されて，それを後で実験が追いついていくという場面が少なくはないと思います．

そういうふうに発展しますと分かれて，そこから片や理論が空を飛び始めるわけですね．理論は空を華やかに飛んでいるわけです．実践は地面をのたうちまわって，はいずりまわっているんですけれども，獲物がいっぱいありますからだんだんでかくなっていくわけです．

空を飛んでいる理論のほうは食うものがないし，そのうちに疲れてきて，やっぱりやせ細ってくるわけですね．下を歩いている実践のほうも太ってはくるんだけれども，よく見通しがきかない．だからどこを歩いているか分からなくなったりするということで，ほっとくと，両方とも実は衰退するわけです．衰退するときに，理論のほうが実際的な意味を見つけてきて，それで共に生きる．共生する．実践のほうも自分の限界を理論から教えられて，さらに行き先を明らかにするというようなことが必要であります．

理論の実践化と実践の理論化

それと同時に，実践をやっていても結局論文を書く必要がありまして，論文を書くときには物事を整理する必要があるわけです．当然実践の理論化ということが一つ進むわけです．

一方，これはコンピュータで特に特異なことですけれども，理論をやっていましてもそれが本当に正しいかどうか，数学だったら手計算するところを，計算機で本当にやってしまう，つまり手計算をプログラムとして実際に試すということで，そういうことでも実践化が起こってくると思います．もう一つは，本当にそれが正しいのか問われ

たとき，デモンストレーションが要るということで理論の実践化が始まると思います．

ソフトウェアの歴史をたどるとコンパイラの理論と実践を抜いて語るわけにはいかなくて，1950年代の後半でプログラミング言語がつくられコンパイラがつくられたというのが一番大きな発展だと思うんです．その場合も，言語のほうは言語理論とかオートマトンとかというのがあって必ずしも実践が先というわけではないんですが，コンパイラに関しましては，あるいはプログラミング言語そのものに関しましては実践があって，それから理論的な興味があって発展してきたということだと思うんです．

その次に，より速くするということで，最適化コンパイラということが出てきたと思います．この中においては実際に速くする，何が何でも速くするんだということと同時に，たとえばレジスタ割りつけの問題にしろ理論的に解明する必要があって理論が進んできた．最近特にリスクなんていうアーキテクチャが出ますと，もう一度つくられた理論をうまく実践が使うといういいパターンになってきていると思います．最後に，私がやってきたベクトルコンパイラということですけれども，これは一つは並列とかそう理論的な枠組みがありますけれども，何が何でも速くしたいという実践的な要求からきているわけです．その中で理論的なものが芽生えてきたんではないかと思います．

理論の役割

大堀　先ほど安村さんから，理論と実践についてのすばらしい広辞苑の定義がありました．この定義でおもしろいと思ったのは，「実践は理論を揚棄するもの」との規定です．揚棄というのは非常に哲学的な用語なんですが，10年前哲学教室で勉強したことを思い出してみますと，それは，何かある固定された理論体系を見直して捨てて新しい体系をつくるという行為です．実践，すなわちプログラムというのは何かのアルゴリズムをインプリメントするわけですね．その，プログラムがインプリメントするもの，それが理論だと思います．理論とは通常呼ばれないかもしれませんが，なんらかの理論的なものと捉えることができると思います．理論は，形式的なステップ・バイ・ステップのものとは限らないわけです．そこで先ほどの広辞苑の定義は，実践によって理論を見直して新しい理論をつくり実践を繰り返すという，理論と実践の関係のあるべきすがたを述べていると解釈できるわけです．

現在のようにソフトウェアのシステムが非常に大きくなってきますと，実践がその対象としてつくるものは，そう簡単に直感などでは対処しきれなくなってきていると思われます．それらをなんらかの形で分析する道具が必要になってきているわけです．それが理論にほかならない，と思います．少し狭く考えてプログラム言語の設計を例にとりますと，理論の役割は非常に大きいと思います．なぜかといいますと，プログラム言語には非常にいろいろな要素が絡み合っています．たとえば高階の関数ですとか，多相型であるとか，抽象データ型であるとか．そういうものの関係がよく分かっていないとちゃんとしたものが作れないというのは明らかであると思われます．すなわち，言語のいろいろな機能の数学的な性質の分析，一言でいうと理論的な分析，が重要と思われます．

私は，現実的にも，理論と実践の間に乖離はそんなにないと思います．ましてフラストレーションは私個人はあまり感じたことはありません．ただ，理論と実践に関するいろんな誤解があるのではないかとは思います．先ほども言いましたが，理論に基づく設計というのは，何もステップ・バイ・ステップの定理証明系に基づいた設計とは限らないわけです．たとえば，デ・ミロなどもいっておりますように，理論の代表の数学をとりましても，非常に個人的なプロセスであって，しかもそれが理論としての信頼性を獲得するのも，実は社会的なプロセスなわけです．ですから，理論は，定理・証明系とかそういうものに限るのではなくて，ある社会的な信頼性をもった枠組，手法ととらえるべきだと思います．

そうしますと，理論に基づく設計というのは，理想的にはこうなってほしいわけです．まず最初は，懇親会などで有識者がこんな言語をつくってみようと話し合うわけです．ここまではすべて共通なのですが，決定的に重要なのは，「では」といって始める前に，確かに多相型とポインタはう

まく両立するのかとか，そういうようなことをまず確かめてみるわけです．さらに，それがうまく働くことを確認したら，実際コーティングする前に，その証明に基づいてアルゴリズムを抽出するわけです．それが理論の役割であると思います．ところが現実はというと，懇親会での有識者の雑談や望ましい機能の選定，ここは共通なんですが，次が違っていて，有識者が委員会を開いて決定するわけです(笑)．これは能率はいいかも知れませんが，どちらかというと前者のようでありたいと思います．

ただ，そういっただけでは皆さん信じてもらえないと思うので，広辞苑ほど権威はないかもしれませんが，こういう記事を見つけましたので紹介します．これは去年の1月 AT & T のネットワークが長時間ダウンした事故を報じたニューヨークタイムズの記事です．原因はプログラマのミスなんですが，AT & T はその分析の結果，もうちょっといいプログラミング言語があったら防げたんじゃないかと信じたらしいんです．この後 AT & T はソフトウェアの理論の研究を強化すべくベル研の再編成を行い，ソフトウェアエンジニアリング・デパートメントという部を新しく作ったのですが，これは実はプログラム理論の研究部なのだそうです．

先ほど何回か，理論に基づく言語がないという声がありましたがそんなことはなくて，一つすばらしい言語で「Standard ML of New Jersey」というのがあります．能率のいいコードを出す実用コンパイラです．これを使えば先ほどの例のような事故は防げる，と言う意図は毛頭ないんです．ないんですが(笑)少しはよくなるんじゃないかと．ならないかもしれませんが努力をしたほうがいいと認める人は少しいるんじゃないかと思います．

僕は理論と実践の関係について非常に楽観的な見方をしていますが，しかし問題点は幾つかあると思います．そのなかで最も深刻と思われるものは，いわゆる理論派の無関心です．たとえば二階のラムダ計算というのは，ジラールの提案から数えれば，20年近く研究されてきたのに，最近まで，たとえばレコード型とか，バリアント型とか，ポインタというのをどうやって扱っていいかまったく分からなかった，というよりはまったく研究されなかったというのは少し異常ではないかと思います．理論の研究者は，こういう実用的なものにも関心をもつべきであると思います．しかしながら私の個人的な意見としては，幸いにこの傾向は変わりつつあると思います．たとえば型理論をやっていらっしゃる方々も急速に実践的なものへの関心をもち始めていると思われます．たとえばレコード型とかポインタなど，領域理論などよりは泥臭いんですが，それらを領域理論などと同じ信頼性をもつ仕方で扱おうという努力が急速に始められています．

というわけで結論ですが，「理論派」と「実践派」という不毛な二者択一ではなくて，実践からみれば，理論は実践の一部ととらえ，理論側と実践側が一緒になって，実践の経験から生まれた種々の有用なアイデアを効率的に，しかも堅牢な形で実用化するための技術の確立を目指すべきではないかと思うわけです．たとえば，オブジェクト指向言語の経験でインヘリタンスとか，クラスといった直観的な有用な概念が生まれています．既存言語にとにかくそれらをつけ加えてみるというのではなく，それらを整合性をもって効率良く実装するための技術の確立を目指すべきではないかと考えます．あまりまとまりませんでしたが以上です．

理論と論理

竹内　理論は不勉強ですがそんなに嫌いではないんです．きょう電車の中でどうやって理論派をやっつけようかと考えながら，「理論，理論」を頭の中でくるくる回しているうちに，だんだん「論理，論理」というふうになってきました．そうか，論理をやっつけようと．

理論には二つのタイプがあるようです．今度 MIT プレスから出た鉛みたいに重い本がありますね．Handbook of Theoretical Computer Science という2巻本ですが，Aがアルゴリズムとコンプレキシティといういわば定量的理論，Bが論理といういわば定性的理論（というより all or nothing 的な理論）です．

さて，理論が役に立たないという立場で話せといわれても，私は定量的な理論は十分役に立っているような気がします．構文解析といった話も，

論理というより定量的なアルゴリズムの理論なのでしょう．プログラミングは記号をいじっているという発想がありますが，記号は質量ゼロの物体ですから，論理だけで考えると，いとも簡単に自己矛盾を起こしてしまいます．これは論理が自然法則の及ばない架空の世界だからといったら言い過ぎでしょうか．ところが実際に扱う記号は，たくさん集まってなんらかの物理量というか質量みたいなものをもち始めます．これを扱うのが結局アルゴリズムとかコンプレクシティの話だと思うんですね．予稿にも書きましたけれども，論理と物理がケンカしたら物理が勝つに決まってます．

このごろプログラミングに関して，理論といえないまでも，いろんなモデルの話を聞きます．でも，ちっとも分からないんですね．分からない原因は，多分60%が私の不勉強で，あとの40%はやはり役に立たないからだと思っています．これは私の直観以上の何物でもありません．理論をやっている人は，というより論理の人は，なんらかの意味で物理量をもった記号を扱ってほしいと思います．あまりにも茫漠とした意見で申しわけありませんが…．

柴山　理論の世界でよくみかける話題に，証明とプログラムの間の同相関係から出発したものがあります．この同相関係からいろいろなことが示せるのですが，今日のパネルに関連したこととしては，「普通の数学者は証明論を気にしない」という事実と，「普通のプログラマはプログラム論を気にしない」という事実の間の同相関係が示せます(笑)．理論的にもプログラムの理論が役に立たないということが証明できるんですね(笑)．

さて，いままで応用数学者の多い学科にずっと席をおいていたのですが，「なぜコンピュータサイエンスの人たちはそんなに難しい数学を使うのか？」と質問されることがしばしばありました．代数構造を使うにしても環や体なら数学者にも分かるのですが，モノイドやセミグループでは共感が得られません．T_0トポロジーなんて最低です．やたら変な記号がいっぱい出てきて，とても普通の数学者には分からないとかいわれています(笑)．やはり質量がない世界だけあって，やっていることが世間一般とずれているような気がします．

なぜずれてしまうのかと考えてみますと，プログラムには，静的な側面と動的な側面があるのですが，この動的な部分がいけないのではないかと思います．静的に扱えるような側面，あえてプログラムをデータとコントロールに分けるならデータ的な部分ですね．これは理論的にも扱いやすいものです．たとえばタイプチェッキングとか，コンパイラの構文解析は静的に扱えるわけで，このような分野の研究はかなり進んでいます．

ところが，プログラムの字面を眺めただけではそう簡単には分からない動的な部分が残ってしまいます．たとえばプログラムの検証などがそうです．枠組みとしては数学くらいしか使うものがないので，静的な記号を使います．でも，扱う相手は非常に動的なんです．大きなギャップがあって，とても扱えない．竹内さんの予稿には，ゲーテルの不完全性というようなことが書いてありましたが，そういう壁に必ずぶちあたってしまいます．

さらに嫌なのは，問題になるプログラムは大体複雑なんです．単純なプログラムは理論でも実践でもなんとか扱える．別にどうでもいいんです．でも複雑なものは困る．何千万ステップとかのコード量を誇るオペレーティングシステムを対象にして理論を展開しろといわれても，そんなことできるわけがない．

実践的なレベルでは，このような複雑さを解消するために，分割統治法とか，似たものを共有するとか幾つかのテクニックがあります．だから，曲がりなりにもメガステップのソフトウェアができてしまいます．でも，そういうテクニックだけでは，うまく理論を構築できません．

生物とか物理とか経済とかの分野だと，些細な要素を切り捨てるとか，離散的なものを連続的なもので近似するとかして，うまく単純なモデルをつくります．でも，プログラム相手に，そういうテクニックは使えません．なにしろ文字を1個入れかえると，もうそれで全然違うプログラムになりますから，近似なんかできない．あえて近似をすると，型のような概念を使うことになります．

証明 ～ プログラム
↓
普通の数学者は証明論を気にしない ～ 普通のプログラマはプログラム論を気にしない

つまり，普通は，1という結果を返すプログラムと2という結果を返すプログラムは違うんだけれど，整数値を返すという意味では同じじゃないかと考えるわけです．これくらいアバウトな感覚で済めば，理論も展開できる．実際，型チェックの理論はかなり成功しています．

でも，プログラムの返り値が1になることをどうしても保証しろといわれると，困ってしまいます．特にメガステップのプログラムが対象だったりすると，どうしようもない．単純化できるところがないメガステップの塊を与えられても，理論側としては手の出しようがないわけです．

ともかく，理論が進むべき方向の一つとして，プログラムの動的な部分を切り捨てるというものがあります．これは，ある意味で情けない方向です．もう一つの方向として，動的なものを扱うために独自の体系を考える可能性もあります．独自路線を行く場合には，数学をあんまりあてにしないほうがいいと思います．

さて，実践の人に対するフラストレーションですが，ソフトウェアをいったんつくってしまい，かつそのソフトウェアがいったん広まってしまうと，20年，30年と生き続けることになります．だから，変なものが広まってしまうと苦労するわけで，なるべく変なものはつくらないようにしてください．ふだんフラストレーションを感じることというと，どうしてもまだ生き残っているのか理解できないソフトウェアがまだいっぱいあることです．具体的な名前を出すと差障りがあるでしょうね．

理論と実践の不一致については，私は結構感じています．では，どうすればいいかというと，まず，理論派とか実践派に分けないことです．一人の人間に理論も実践も両方やらせないといけません．そうすると，少しは，溝が埋まるのではないかと思います．おしまい．

司会　一応ここでひと休みということで，会場から何かご意見があれば，また後でたくさん時間をとりますのでいまいいたいということがあればお願いします．

近藤・日立基礎研究所　安村先生の話で，理論と実践を物理の理論と実験に対応させてたと思うんですが，物理の場合，自然対象に対して照らさなければいけないという判定をするために実験を行うのであって，あれは理論を構築する一つの手段なわけです．理論と実践は僕は科学と工学というふうに対応させるべきだと思うんですけれども．そうすると，ソフトウェアの場合はある意味で非常に不幸な生まれ方をしていまして，応用から先に始まったわけですね．要するにほかの分野だと，たとえば電子工学なんていうのは物理学の存在なくして，要するに科学の存在なくして考えられないわけですけれども，ソフトウェアの場合は先にとにかくなんか記号を書けば計算機が動くと．全然ソフトウェアの科学なしに始まってしまって，その後，その後追いの形でソフトウェアの科学が始まっている．そのために現実のプログラマからは疎んじられるんです．この関係はまさに数学というプラクティスが先に始まって，その数学の整合性を本当に大丈夫なのかと問い直して始めた数学基礎論が数学者に，いまも話がありましたが，徹底的にばかにされて敵視されているというそういう構造と僕は非常に似ていると思うんです．

ただ，科学と工学という立場からみれば，基本的に工学というのは科学なくしては健全な発展は少なくともほかの分野ではしていませんし，そういう意味ではソフトウェアの場合も．だから理論という言葉を僕は科学という意味にとらえています．そうとらえれば，それがなくしては健全な工学，つまり実践というのは僕は存在しないと思うんですが，これについて実践派の方々から何か伺いたいんですが．

安村　自然科学は私は具体的な自然現象があって，それから理論ないし実践が生まれてきたと思うんです．つまり熱力学でいうと蒸気だとか，電気でいうと電気現象が分かって，それから後で理論が構築されるんだと思うんです．

あと，科学と工学という分け方ですけれども，普通よくあるのは，数学会があって応用数学会があって，物理学会があって応用物理学会があって，何か基礎的なものと応用があるという感じですけれども，応用ソフトウェア学会とか応用情報処理学会ってのはないんですよね．だから何かやっぱりそこはちょっと違うと思うんですね．やはりコンピュータ科学の場合は，ソフトウェアでものをつくるというところがほかの科学にない重要な違いじゃないかと思うんですね．

竹内　いまのお話に関して，科学と工学という分け方のほかの考え方もあると思います．ソフトウェアはやはり言葉の問題ですから，人間の文学創作活動に近いというのがもともとの私の立場です．つまり，理論が音楽学で，実践が作曲だという見方なのですね．音楽学というのは作品を生みません．分析はします．たとえば，バルトークの管弦楽のためのコンチェルトの構成は，入れ子になった黄金分割になっているなどといった話は後追い的に出てくるんだけれども，音楽は生みません．もっとも，バルトークが密かにそういう「理論武装」をしていたとも言えますが…．

実践というのは，やっぱり必要があって物を生み出す，作ってナンボの世界です．理論の人，あるいは理論っぽい人のものの言い方では，最初に大規模なソフトウェアを正しくつくるためにこういうものを考えるんだという前置きがあります．それから自分のつくったモデルないしは言語ないしは理論みたいなものの紹介が続きます．最後に，しかしこれはまだ現実の問題には使えないレベルであるとくるんですね．トリオ形式というわけです．つまり，やはり現実の複雑さに対処するにはギャップが大きすぎる．このギャップはいつまでたっても埋まらないような気がしています．これは創作と評論の間のギャップと同類のものかもしれません．

本田・慶応大学　理論的に真の複雑さに対処できないのはそのとおりだと思います．しかし，実践的に対処する方法があるんでしょうか．

竹内　さっきデ・ミロ，リプトンの話が出てきましたけれども，やはりソーシャルプロセスしかないと思いますね．これは別の機会に言ったことがあるんですが，ソフトウェアの検証の理論には，やっぱり保険屋が最大の貢献をするんじゃないでしょうか．バグが出たらナンボの被害や，これしかないんですね．この人がつくったソフトは保険率何％とか，あいつだったらそれの2倍だとか．

だけど私は理論を否定しているわけじゃない．理論は実践を行う人にとって大事な教養です．「教養は発明の母」ってなかったっけ．そうか，あれは「必要は発明の母」(笑)．でもとにかく「教養は創作の母」です．教養のない人が作ったものは絶対だめです．理論屋さんの本領は，実践をやっている人を導き教えること．宗教活動みたいなものですね．言葉を用意したり，概念を用意したり，概念の整理の仕方を用意したり，そういう教養だと思うんですね．それ自身は何も生み出さないんだけれども，ありがたい教えを受けた人は自分の行動の規範にそれを取り込む．結局，ソフトウェアの進歩はほとんどこれでしたものね．別に理論の体裁をしてなくても，概念の整理に役立って，みんなが思考を節約できる教えです．昔だったらとても一人で作れないようなものが簡単に作れてしまうのも，そういうものの積み重ねの結果でしょう．まさに人間の文化の蓄積なんですよね．

たとえば文学とか法律などにフォーマルな理論があったかというと，過去2000年ないしは3000年の歴史があるんだけれども，やはりないですよね．ソフトウェアもこれと似た宿命をもっていると思います．

二木　皆さん同じようなことを言っていて，ケンカになっていないという印象です．理論の有用性に関しては大堀さんが非常にパンチのある意見を述べられて，皆さんそれで少したじろいだんじゃないでしょうか．いま竹内さんがおっしゃった宗教というのは私も常々感じています．

しかし，安村さんがおっしゃったように，宗教よりもっと具体的に役に立つ部分もわれわれは経験してきたわけですよね，コンパイラの話とか．そういう部分が広がらないというのはあまりに偏狭すぎて，これからどんどん広がるべきでしょう．どの辺が広がるかというのはやっぱり議論するべきだし，実践の人にわれわれはそういうことを言っていくべきです．竹内さんもこれには異論を差しはさまれないんじゃないかと思うので議論にならないかもしれませんが．

竹内　私はプログラムあるいはプログラミングの理論と，それ以前の問題を解くための理論を区別して考えないとここで議論がかみ合わないと思います．コンパイラのための構文解析論などは，問題を解くための理論ではないでしょうか．それに対して，大堀さんや二木さんの話は比較的純粋にプログラムの理論だという気がします．

ダイクストラはクセのある人ですが，彼が言いたいのは，問題が与えられたらプログラムをつくる前にその問題自身についてよく考えろということでしょう．こういうところまでプログラミングの理論の中に入れたら話が拡散してしまうと思います．

柴山　しかしそこまで限ってしまうと，プログラミングの理論にはほとんど何も残らないんじゃないでしょうか．ドメインが決まればそれに応じていろんなモデルがありますが，計算のモデルはラムダ計算とかコンビネータとかで終わってしまいません？　それで話がかみあいますかね．

二木　ドメインに依存した話は切り分けようという竹内さんの論旨はよく分かるんですが，コンパイラに関してはなかなか微妙だと思います．たとえばパージングの話は，パージングというドメインに確かに依存しますが，プログラム全体からみるとやっぱりメタセオリなんですよね．したがってそう簡単には切れない．タイプの話も同様です．タイプの話は，確かにタイプ理論というドメインには依存しますが，プログラムとかソフトウェア構成からみるとメタセオリであって，プログラムの理論にもなるかもしれない．ですからそこはまた後で議論したらいいんじゃないかなと思うんです．

大堀　僕は実は逆で，理論というのはすべて問題を解くための理論でそれ以外ではないと思います．それを切り捨てると何も残らなくて戦争は起こらないんじゃないかと思います．ユニフケーションの理論にしろ，ボトムアップパーザの理論にしろすべて問題を解くための理論です．

司会　後半戦をお願いしたいと思います．

村井　皆さんこんにちは．慶応大学の村井です．

役割分担は現場からの辛辣な意見というわけで，とび職の職人が建築学会で話をしろといっているような雰囲気がありまして大変誇りに思って今日は来ました．

私は理論派じゃないなんてちっとも思ったことないんですけれども，私は私なりのプロセスの中で生きているわけです．それを分かっていただくためにまず最初に自己紹介を．昔から私は何をやりたかったかというと，これからのコンピュータシステムはどうなればいいかな，次世代の環境はどうなればいいかなということを考えていたわけです．しかし，いい環境はないし，いいOSもない．それからネットワークもない．そこで，84年に始めたのがワイドプロジェクトです．ワイドプロジェクトは，分散システム，ネットワーク，OSに関心のある人間が集まって，それでよい世の中をつくる，こういう目的しかもっていないような研究者の集まりです．

現場から理論と実践

そんな自己紹介を終わりまして，現場から理論と実践，こういう話をしなければならないんですが，理論と実践というのは何なのかという話はすでにさんざんされましたのでだいたいいいんじゃないかと思いますが，僕は理論と実践というテーマを与えられたときに最初に考えたのは，熊がかゆくて背中を木にこすりつけるときの理論と実践は何だろうということを考えたんです．やっぱり状況をとらえたり環境をとらえたりして，何がどうなっているのかということを分かるというのが理論で，じゃ背中をこすりつけちゃおうというこの行動，こういうのが実践に当たるんだろうと思います．いずれにせよその問題意識が分かって何をやるかという話なんです．

となると，理論がなくて実践があるというのは何なのか．つまり考えなしに何かをやるというのが，これが実は現場では一番恐ろしいことなんですね．つまりシステムをつくり出すのに何も考えずにシステムをつくり出すことこそ怖くて，実はこれはしょっちゅうあるわけです．さっきの動物，熊に戻ってみると，いきなり何かするというのは，熊じゃなくてもいいです．人間でも．いきなり何かするというのはわれわれはどういうときにやるかというと，泥酔状態とか薬物中毒とか考えなしにいきなりやる．こういうのは普通狂気とこういいます．狂気という言葉はきつくてためらったんですけれども，とりあえずそういう意味で狂気とこういっています．

つまり理論がなくて実践があるというのは，ほとんど許せないわけです．ところがこれがかなりまかり通っている．理論だけで実践がない．これはもう全然しようがないと思っています．理論だけで実践がないというのは要するにしようがない

わけです．

以上は理論と実践の話で，さっきからごっちゃになっているのは，「理論派と実践派」という話と「理論と実践」の話です．これは全然別な話で人間の話ですね，派というのは，派の話は後でします．

つまりいろいろな目的と思想があって理論を考えていってそれが実践に結びつかない．かわいそうに．残念だったね．まあ，しようがないかと．こういう話です．だけど大いにやってくれないと困るわけです．

それからシステムプログラミングでわれわれシステムをつくるのに何かというと，性能の問題が非常にあるし，それから複雑さという話もありましたね．われわれネットワークをつくったり，OSをつくる場合もそうですけれども，実際にそういうものをつくり上げるために一般に役立つといわれている理論というのはあんまり役に立たない．そこでいろいろなことを試行錯誤しながらそのフィードバックというのは必然的に非常に頻繁に行われるわけです．そういう意味だとシステムプログラミングの分野では理論と実践の関係は必然的にかなり近いわけです．

理論派，実践派，情報派

理論派，実践派，情報派，狂気と書いてあります．基本的には理論派というのは自分の頭のために仕事をする人ですね．だから自分の頭に気持ちよければいいわけで，だいたいそういう人を理論派とこういいます．全然これは否定していないわけです．つまり自分の頭に一番気持ちいいことをやってくださいね．その結果いいものがあったらもちろんいろいろ役に立つでしょうし，そうじゃないということも幾らあってもいいわけで，それが理論派の人．

実践派の人．頭以外のためも考えてしまう人ですね．これは体が気持ちがいいとか，世の中がよくなるとか，そういうことを考えるのは実践派．そういう意味で私はあそこに座っているわけです．つまり頭が気持ちいいだけじゃ嫌だなと，こう思ってしまう．これが実践派．こうだいたいいっているんじゃないかと思うんです．

それからさっきこちらのお話からちょっと出たけれども，つまり言語の標準化の話が出ましたけれども，そういったような問題というのは大変問題ですが，それを実践派の問題点としてさっきあげられましたけれども，そういうことがないように私は情報派というのをつくりまして，これは流行と政治と商売のために仕事をしたい人．これを情報派とこう呼びたいわけです．実践派の仲間に入れてあげない，こういう人は．ここの特徴は何かというと，一貫性のない理論や実践．理論も実践ももっているようだが一貫性がない．つまり今日このことをやって，この理論でやってもいい．この実践をしてもいい．あしたは違うことをやっている．つまり一貫性がない．こういう人を情報派と呼んで理論派と実践派と区別したい．

狂気というのはそうなってくると何なのかというと，今度情報派を含めて夢のない理論．そして理論のない実践．これはさっき言ったとおり狂気である．

そうすると重要なことは何かというと，理論にしろ実践にしろというか，つまり研究者にとって派の，ですから人間にとって重要なことは何かというと，根源になる思想と哲学なんですね．そうすると一貫性とかそういったものがちゃんとできてきて，そういう中で行動していく．これが研究者として，つまり派というのは研究者のことですから，心構えじゃないかなと，こういうふうに考えるわけです．以上．

形式仕様の理論と実践

二木　形式仕様という大変に理論的と思われるテーマが，現在から将来にわたって実践で役立つ技術である，という立場でお話します．

まず，仕様には非形式的なものと形式的なものがある，ということから始めます．非形式仕様 (informal specification) というのは，ふつう自然語で書かれていて，常識とかその場の状況とかに依存してしか意味が特定できない，というものです．形式仕様 (formal specification) というのは，これは大変なものでして，何かあらかじめ与えられたルールがあって，それに従っていれば意味が特定できるというものです．

では，理論が何を提供すれば形式仕様なんて得体の知れないものができるかというと，形式仕様言語 (formal specification language) の三つの構成要素として，構文規則，意味モデルと意味規則，構造化法を明らかにしてくれれば良いということ

になっています.

じゃあ，この三つは何なのかということになるわけですが，たとえば意味モデルというのは形式仕様言語の意味定義のための数学モデルです．静的なモデル，つまりデータのような空間的な「もの」のモデル，は数学からの借り物が多いんです．集合とか関数とか関係とか．ただ，抽象データ型などのモデルである多ソート代数とか，順序ソート代数なんかになると，数学プロパーの人が考えたんじゃなくて，計算機科学の人がつくったものです.

静的なモデルに対して，動的なモデル，つまりプロセスのような時間的な「もの」のモデル，は間違いなく全部計算機科学で数学ではないと思うんです．有限状態機械というのは，単純なものですけれども，計算機科学の金字塔ですよね．その他，ペトリネットとかプロセス代数といったものが，動的な意味モデルです．このような，形式的な意味モデルを理論が提供してくれれば，常識とか，周りの状況とか，その日の気分とか，組織の状態とか，そういったものにディペンドしないで意味を決めることができますよ，ということです.

構造化法というのは，最近ですと，オブジェクト指向あるいは抽象データ型に代表されるものです．これも計算機科学での宗教といえばそうかも知れません．でも，最初は手続きみたいなものをビルディングブロックにして構造化していたものを，抽象データ型とかオブジェクトを単位にして構造化しようというような進歩はあります.

それと，仕様を書くときの記述の視点というものに一応分類があります．モデル指向とか制約・性質指向とか，ハウ (how) を書くのかホワット (what) を書くのか，といった分類です．仕様だからホワットなんだというのが優勢な考え方なのですが，ハウを書いても仕様だという派もあります.

これは計算機関連の科学技術全般について言えることだと思いますが，ソフトウェアや計算機を作るということが先にあって，すべてはそのためにあるという雰囲気が強いということです．形式仕様についても，仕様を書くというのは，システムづくりが始まって以来，もう営々と自然語や図を使ってやってきているわけです．それをいまのままじゃちょっと問題が多いというので，なんとかちゃんとしようという話ですから，理論が先にありきという話ではないんです．やっぱり実践屋が偉い，やっぱりいい物をつくっていい会社つくってお金持ちになった人が偉い，という雰囲気があるんですね.

そういう中で理論派が提供すべきものというのは意味モデルや構造化法みたいなものです．それをちゃんと創り上げてくれれば役に立つし，それを創ること自体は，それができる人にとっては非常におもしろいだろうし，あんまりフラストレーションもたまらないだろうと思います．それが実践派からどうみられるか，あるいは情報派からどうみられるかというのは別の問題だと思うんです.

ちょっとおもしろいのは，形式仕様に関する立場というかテイストが，ヨーロッパとアメリカとはっきり違うんですね．ヨーロッパはなんといいますか，貴族の社会というのか，原理原則をキチンとやるという雰囲気があり，全般的に形式仕様の研究が盛んです．VDM, Z, 代数仕様なども盛んですし，SDL, Estelle, LOTOS なんていう，最近の形式仕様言語も全部ヨーロッパが中心です.

アメリカは，国防システムなどの高い信頼性を要求されるシステム用の検証ツールなどを盛んに研究しています．この分野でアメリカは非常に強力で，形式仕様とそれに基づくシステムの検証という大変理論的な研究の一番のアプリケーションをもっているんじゃないかと思います．しかし，アメリカのソフトウェア開発の一般的なテイストは原理原則よりは現実重視で形式仕様には向いていないようにみえます.

形式仕様と理論計算機科学の活動の例をみると，将来もそんな捨てたものじゃないじゃないかという例が幾つかあると思います．竹内さんへの盾として幾つかあげておきます．まず，言語システムには，コンパイラに代表されるように，理論が非常に役に立っています．これは皆さん認めていらっしゃると思います.

つぎに，OSI (Open Systems Interconnection) 関連のプロトコルの標準化のための形式仕様言語の開発です．こうした言語は計算機にかけることを目的にしていないんです．100 ページぐらいの仕様書をきちんと書いて，世界中のシステムエンジニアに間違いなくそれを理解してもらうというのが目的です．そのためには，どんな形式仕様言語

でどのように書いたらいいのか，ということをやっているんです．たとえば，SDL，Estelle，LOTOS というのは，そのために設計された言語です．これらの言語は動かすために設計されたんじゃなしに，形式仕様を書くために設計されたんです．人間に読まれるために設計されたといってもいいと思います．

それと，一般的な形式仕様言語としての VDM と Z，それに OBJ などの代数仕様言語の回りの活動があげられます．

代数仕様言語の利点の一つに，ちゃんとしたモデルに基づいた系統的なモジュール化がしやすいということがあります．自然語では，系統的なモジュール化はなかなか難しいと思います．

当然ですが，形式仕様にはその基本的な考え方から必然的に生ずる欠点があります．それは，現実とモデルの乖離ということです．形式仕様の基本は，現実世界を抽象化したモデルの世界で活動しようという決断なわけですから，そのモデルの世界からはずれたものはどうしようもないということです．

ダイクストラは，この現実世界を抽象化したモデルや形式体系の教育こそが計算機科学の本質的な性格を反映した問題だ，という主張を最近の C. ACM 誌上で行っています☆．形式仕様はまさにこの問題を抱えているわけです．こうした問題点はあっても，他の技術との比較で判断すれば，やはり形式仕様は現在から将来に向かって実践でいかされるべき，理論に裏付けされた技術であると思っています．

理論派，実践派，言語派

上田　今日のパネルでは実践側の席に座ることになったのですが，竹内さんは，私はなぜ実践代表なのだろうかと思っていらっしゃるそうで，実際，理論か実践かと言われると私の立場は微妙です．そこで，新しい研究会のタイトルを見ると「プログラミング―言語・基礎・実践―」と書いてあるんですね．じゃ，言語派で出てくるのがいいんじゃないかと思ってきました．

基礎と実践というのは一つの分け方なんですが，どちらをやるにもプログラム言語が土台になりますし，計算モデルも一種の言語と言えます．こういうプログラミングの土台になるものを，この研究会では大事にしていこうということで，一見不自然にみえるかもしれませんが，「言語」が「基礎」「実践」に並んで研究会の名前に入っているわけです．私の仕事もちょうどプログラム言語の研究ですから，そういう立場から言語と基礎と実践の話をしていきます．あんまりメタな話をして議論が空を切ると寂しくなってしまうので，自分のやってきたことと結びつけて話そうと思います．

私の仕事というのは，第5世代コンピュータプロジェクトで，並列ハードウェアを作りたいという人と知識情報処理をやりたいという人の橋渡しをすることです．この間のギャップは非常に大きくて，トップダウンやボトムアップのアプローチは難しそうであるというので，ミドルアウトというアプローチをとって，核となるプログラム言語を先につくることになったわけです．

そのとき，言語というのは単にものが書けて実装できるだけのものではありません．ものを書くという仕事は，思考の表現とか概念の整理とかいろんな副次効果をともなうわけです．さらに，これからつくる言語は実装するとか応用プログラムを書くとかいう実践面だけではなくて，理論に支えられるものでもあってほしいという前提があったわけです．

ところで，新しい言語や計算モデルというのは

言語設計における理論と実践の同時性

▲ 与えられた言語／計算モデルに対して（速い）処理系，
↓ （正確な）理論を開発する（分業）
● 処理系（の作りやすさ），理論（の簡明さ）を言語設計に反映させる
　• 複雑怪奇な理論 ｛定式化のまずさ／記述対象のまずさ
　• 〃　処理系についても同様
▲ Ad hoc な最適化／プログラム解析技法の開発
↓ 系統的なプログラム解析技法
●
　→体系を言語機能の形で明示（＝概念の提示）
　• 静的な型体系（手続き型，関数型，...）
　• 静的な入出力モード体系（GHC）
　　ユニフィケーションの（動的）双方向性
　　• 有害論（処理系側）
　　• 不要論（プログラミング側）
　　→情報の流れの静的解析（理論側）
　　→入出力モード体系（moded Flat GHC）（言語側）

☆ A debate on teaching computing science, C. ACM, December 1989, p. 1397-1414.

そうそう軽々しくつくるものではないという風潮が，最近ではずいぶん浸透してきました．つくって5年もたせるのは結構大変な話なんです．私が設計したGHCという言語は幸いにして5年はもったわけですが，理論でも実践でもなくて，なんというかな，原則主義なんですね．これは日本ではあまりはやらないやり方ですが，まず原則を明確にして，それはよっぽどのことがないと変えないわけです．たとえば単純さであるとか，副作用は入れないとか，並列と並行という概念は違うから分けろとか，いろいろな原則をぶち上げて，それを動かさないことにしたわけです．また，ミドルアウトで始めましたから，設計した言語を正当化することも非常に重要です．そこでどうしたかというと，記述力，処理系，理論の三面から言語を検討して支える，あるいは言語を出発点にしてこれら三方面に同時に発展することを目指したわけです．

なぜこういう多面性が重要かというと，たとえばラムダ計算とLispとか，pure Prologと普通のPrologとか，およそ手続き型以外のパラダイムというのは，理論版と実践版に分化して互いの関係がなくなっていくことが普通なわけですが，そういう状況をできるだけ避けたかったわけです．実際には，GHCにも理論版と実践版とができているんですが，少なくともそのふたつが悲劇的に無関係なんじゃなくて，きちんとどういう関係にあるかが分かっているということが大切です．

言語が多面的に支えられていると，それに新しい機能を入れるとか，あるいはサブセットをつくるとかいうときに，いろんな角度からその設計の正当性を判断できるという利便があります．

もう一つの多面性として，言語に複数の説明を与えることをしました．GHCは並行論理型言語と呼ばれていますが，データフロ言語にもみえるし，オブジェクト指向にもみえるし，アセンブリ言語にもみえるという話です．通信や同期の機構についても，論理的にも代数的にも解釈ができるという多面性をもっています．さらに，これは非同期な並行処理言語なわけですけれども，他の並行処理の理論とどういう関係にあるかということを，非常に基本的なレベルではっきりさせておくべきだと考えました．そういうふうに，いろいろな手だてを打っているわけです．

では，プログラム言語において理論と実践がどう関係しているかといいますと，私としては，言語設計に役に立つ理論と実践をやりたいわけです．ふつう現場では，言語はISOあたりで決まったものをそのとおり，そつなく作ればいいわけです．一方理論屋さんの中には，与えられた言語に正確な意味を与えることを使命にしている人もいます．

私のやりたいのは，そういう言語設計と処理系作成と理論の分業ではなくて，言語のこの部分をどう変えると処理系がどうつくりやすくなるか，あるいは理論がどう簡明になるかということをほとんど同時に考えて，いい落とし所を探すということです．ここが物理学みたいに議論の対象が働かせないものと違うところで，言語設計では，議論の対象は人間が作るものですから，対象が動かせるということを最大限に利用したいわけです．複雑怪奇な理論をつくってしまったら，定式化がまずいんでなければ記述される対象がまずいんじゃないかという見方をすることが必要だと思います．処理系にしても，変な処理系ができたら，それは言語が悪いんじゃないかという見方をする必要があるわけです．

今度は処理系側の話に移ります．たとえば速い処理系を作りたいとしますと，最適化技術や，そのための理論としてのプログラム解析などが出てきます．それが役に立てば，理論が実践を導いたということでまあ喜ばしくはあるのですが，言語派はこれでは満足できないわけです．なぜか．プログラム解析を使って処理系が速くなっても，それは言語の上には残らないんですね．ところが，プログラム解析技法がシステマティックに記述できて，それをいまつくろうとしているプログラム言語の機能として明示できれば，それは概念として後々残るわけです．だから，そういうものを目指していきたい．

たとえば関数型や手続き型では，最初に鈴木さんが言われたように，タイプシステムが非常に発展したわけです．それとまったく独立ではあるのですが，精神として非常に似たモードシステムなるものを，GHCのために最近つくりました．GHCでは，単一化(unification)を使ってプロセスが通信するわけですが，単一化というのはご存じのとおり，情報がどちら向きにも流れることがで

きる．ですが，実用的なプログラムをよく解析すると，たいがいの場合どの単一化によってどういう情報の流れが起きるかが分かるんです．だったら，そういう解析が気持ちよくできるようにするにはどういう入出力モードの体系を入れればいいかを考えるのが言語派としての立場なわけです．

この仕事には背景があって，処理系を作る側から，一般の単一化があると処理系が作りにくいという意見があり，プログラムを書く人からはそんなものは要らないという意見が出ていたのです．それではということで少し理論から攻めてみると，情報の流れは静的に解析できることが分かった．しかも，すぐそばに実践の人がいると，そういう静的解析というのはグローバル解析をしなければいけないのでは困るということを言ってくれるんです．つまり分割コンパイルできないものはだめであると．そういうことが制約条件になって落とし所が決まってくる．このように，言語をつくることと，プログラミングをする，あるいは処理系を作ることと，理論をやることとのインタラクションは，言語設計に非常に役立つ場合があるわけです．

司会 ありがとうございました．あと 30 分ほどありますので，ぜひいままでの発表に対して会場のほうからお願いします．

本田・山梨大学 今日のこの題で一番僕聞きたかったというか期待していたのは，全体的なコンピュータにかかわる流れとして，理論をやっている方から何かちょっと道がおかしいんじゃないかとか，これからあっちのほうに行かなければいけないんじゃないかみたいな何か気がついたことが出たときに，本当に実践派のほうにそれを大きな声で言えるだろうかとか，逆に実践のほうとしては，おれたちはいつもこうやってぐちゃぐちゃ自分の実務をやっているけれども，本当は何か大きな流れとしては違うかもしれない．理論をやっているやつらがそれを分かるんだったら，そっちのほうが分かってくれるんじゃないか．

そのときに本当にそれをお互いに言ったりとかという状況ができているのかなという不安が今日のパネルなんじゃないかなと一つ思いまして，それをやるには私，コンピュータサイエンスという言葉が悪いんじゃないかなと前々から思っていまして，コンピュータといった以上はエンジニアリングしかないんじゃないかというふうに思って，それが両方がくっついていく一つの道じゃないかなというふうに考えているんですが，先生方のご意見をお伺いしたいと思います．

村井 僕もさっきから同じようなことを思っていましたけれども，サイエンスかエンジニアリングかという問題ではなくて，それぞれが好きなことをやっていて，それぞれの分野の仕事をしているのは全然構わないと思いますけれども，それがどうほかに伝わるかという問題は，もし理論派と実践派と仮に分けたとしても非常に重要な問題で，なぜかというと，多分それは本質的にやりたいこと，要するに仕事の内容からいえばある程度オーバーヘッドになるかと思うんです．たとえばわれわれがやっているようなこともオーバーヘッドが非常にあるんですけれども，われわれが本当にやりたいことに対してオーバーヘッドというのをやっていかなければいけないだろうということでやっているわけです．

こういうシステムをつくったり，物を作ったり，プログラムを書いたり，ソフトウェアをつくり出したりするのだってそうで，つくり出して，それが動いて，メンテナンスしてバクとって，そういうことをやっていくのはやっぱりある意味ではオーバーヘッドだけれども，その表現，そういうオーバーヘッドを引き受けた上での表現というのは，もし理論と実践という二つの実態があるとしたら，その間の非常に重要な橋渡しと切り口になるんだと思っています．それがサイエンスとエンジニアリングにどうかかわるか，僕は分からないですけれども．

二木 いまは理論でも実践でも，テクノロジというか，テクニックがものすごく高度になっていると思うんですね．分かりやすい言葉に翻訳すると，実践でやっていることも理論でやっていることも本当はそんなに大したことをやってないにもかかわらず，テクニックの積み重ねがあるもんだからお互いの分野が分かりにくくなっている．それが非常に残念なことだと思います．

村井 オーバーヘッドといったのは，このタイトルはまさにそうなんですけれども，実践派の人は理論を生かせと，こういっているタイトルだと思います．つまり，もし生かしてほしいと思ったなら，このオーバーヘッドは理論派がもたなけれ

ばいけないと，こういうことをいいたかったわけです．理論を生かしたいなと実践派が思ったら，それは取りにいかなければならない．要するに実践が理論にフィードバックがかからなければいけないなと思ったら，その努力は実践がやらなければいけない．つまり思ったやつがやらなければいけなくて，それは自分の本当にやりたいことから比べたらオーバーヘッドかもしれないけれども，その努力をやらないと．要するに受け入れてもらいたいんだから，それは自分の責任でやらなければいけないわけです．たとえばわれわれもネットワークをつくってオペレーションする．これはオーバーヘッドだけれども，われわれがやりたいことを受け入れてもらうためには，これをやっていくことが重要だと考えているわけです．そういった意味でのオーバーヘッドという言葉を使ったわけです．

多田・電気通信大学　電気通信大の多田ですが，いまの話に関してなんですけれども，どうも僕の思っているところでは，理論の人の言っている理論と実践がやろうとしていることとが非常に離れている．セマンティクスのギャップがあるといってもいいと思うんですけれども．

僕はどっちかというと，ここで分けるならば多分実践派に入ると思うんですが，たとえば何年か前に情報処理学会でスタックのメモリアロケーションの理論的な文章を自分のシステムに使おうかなと思って読んでいると，最後にその理論はスタックの行動というのは，スタックポイントの動きはランダムウオークを仮定してあって，世の中の計算機にランダムウオークするスタックポイントがあったら一度見てみたいと思うんですが．そういうのですとか，それからOSのキューイングセオリでも，キューイングセオリはいろんないいセオリがあるんですけれども，あれを実際にOSのスケジューリングに使ったらろくなOSはできないと．

そういう意味でまだいまのところ，ここにいらっしゃる方は大体近いだろうとみているんですけれども，実際の理論屋さんと実際の実践屋さんではまだ非常に距離があるのが原因だと思います．もちろん実践をやっていて理論は考えているんですけれども，それはいまの理論屋さんがやっている理論とはやっぱりまだ近づき切れないでいるというのが意見なんです．

そこでちょっと質問したいのは，実践の方々が一実践終わった合間にどういう理論的なことを考えていらっしゃるか．それから理論の方々は一理論終わった後にたまに実践面に思いを巡らすと思うんですが，どういうことを考えているかというのを聞いてみたいなと思ったんです，この機会に．

大堀　私は上田さんと非常に近いという感じをもったんですが，理論屋というよりは，何かちゃんとした道具を使って言語をつくってみたいというのが私の関心で，私のやっているのは，ラムダ計算の型理論を実践に使えるように拡張するというような仕事です．その形式的な体系をインプリメントする言語のインタプリタを Standard ML of New Jersey で書いたことがあります．

二木　なんで日々プログラムを書いているかといえば，時間的には Nemacs と LaTeX が主で，たまに自分が好きな OBJ という言語で例題を書いているというくらいです．Lisp はたまにコードをながめる程度．ML は周りに好きな人がいて，いい，いいというので，論文は読んで勉強しておもしろいなと思っています．

竹内　いかなる人にも精神分裂症的な二面性がありますよね．私は何を隠そういまだにマイクロプログラムを書いています．それもまだアーキテクチャも決まっていないマシンのものでした．仕様の決まっていないマイクロコードを一生懸命書いていたんです．つまり，マシンのスペックを考えながら，毎日テニヲハを直すような感じでプログラムを書き直していた．こういうふうにマシンを究極的に制御するという生活をしていると，さっき言った二面性によって，計算機に対して絶対の制御権を握っている王様の立場からパッと翻って，今度は計算機に弄ばれてみたいという被制御へ楽しみが移るわけです．

さっき私は盛んに量のことを言いました．ENIAC の時代にアナログコンピュータからデジタルコンピュータに転換したわけですけれども，これからの時代はまたデジタルからアナログへ回帰するような気がしています．2000年にかけてこの回帰が起こると私は信じていて，そういうアナログっぽい話のヒントになりそうな理論とか論文みたいなものを，これどうかなと思いながら少しずつサーベイしているといったところです．もっ

と言うと，可制御性が視野の外に行ってしまうようなシステムの理論にあこがれているんですね．

村井　理論についてお前ら何を考えているのかという質問ですね，これは．

一応いろんなことがありますけれども，信念をもってよく考えてやっているんで，それが私の理論ですけれども．だからいつも理論のことを考えているんです．もっと確立した世界からいうと，ノード数の N が 10 とかそれ以上になったとしても役に立つあらゆるネットワーク理論は大歓迎なんですが，こういったものが．

つまり2点間のコミュニケーションを議論して3点間のコミュニケーションには役に立っても100 点間のコミュニケーションには役に立たないというものがいっぱいあるんで，そのあたりの問題と毎日闘っていると，そういった問題が非常に関心が高いのでよく考えます．

それから，いろいろなコミュニケーションのパラメータの多さ，順番を議論されることはよくあるんですけれども，やっぱり時間の枠というものを考えたときのあらゆる方法がなかなかないわけです．これは実際のシステムには非常に重要なんです．そのあたりのことが理論的な方法に関して関心のあることです．

上田　理論にも興味があるんですが，それは先ほど申しあげたように，一つのもの—たとえば私の場合はプログラム言語ですが—に対するいろんなアプローチの一つにしかすぎません．コーディングも理論もやるわけです．何のために理論をやるかというと，言語設計において判断を誤らないためというのが一番重要な理由かと思います．では，何のためにコーディングをするか．私が最近にらめっこしているのは 80386 の機械語なんですが，現実のハードウェアを忘れてしまうとまともな処理系もまともな言語もつくれなくなると思うんです．少しずつでも理論と実践を両方やるというのは，研究生活における健康法のようなものだと思っています．

司会　残念ながら予定していた時間を過ぎてしまったんですけれども，これで．

じゃ，最後にパネリストの方から一言ずつ．

二木　このパネルを通じての私の総括は，少なくともここにいるパネルメンバにはそんなにひどい理論と実践の乖離はないし，相手の立場と方向性を尊重した理論派と実践派の相互乗り入れの可能性は高い，といういささか希望的かつ楽観的なものとしたいと思います．

柴山　最後ですから今日話さなかったことを一言．私は大学の人間ですので，研究のほかに教育という問題を抱えています．大学で教育をやっている立場から，何が理論で何が実践かといいますと，100 人，200 人を相手にマスプロ教育できるのが理論で，それができないのが実践なんです．本来実践的なことを教えるべき演習を100 人相手にやれとかいわれると途方に暮れてしまいます．でも私学のつらいところで，マスプロで実践を教えないといけないこともあります．実践が教科書化，マニュアル化，理論化されると正直いって助かります．こういうのって，かなり邪悪なことのような気もしますけれど．

大堀　僕は先ほども言いましたように，理論と実践とはそんなに乖離しておらず，実践は理論を十分生かしているんではないかと思います．しかしまだ生かし得ていない理論が世の中にはたくさんあると思えます．たとえばオブジェクト指向プログラミングというのが，非常なエネルギーをコンピュータエンジニアリングに注ぎ込んで，成功しています．このエネルギーが消え失せないためには，それに対応するようないろんな理論がつくられて，新しいアイデアを発展させていくべきであると思います．既存の理論，たとえばアブストラクト・インタープリテーションですとか，モナド理論，多相型理論などを，新しいアイデアを採り入れて発展させていく可能性はいくらでもあるんじゃないかと思います．今後ますます理論と実践の乖離がなくなっていくことを期待しています．

安村　ロジックプログラミングはわりかし，いま理論と実践が手を取り合ってうまくやっている．関数型はどちらかというと理論優先できているから，さっき言われたように ML で本当に実用的なものができるかと．オブジェクト指向は逆に実践的なものが先行していますから，型理論でうまくオブジェクトのいろんなインヘリタンスとか書けるかどうかというのは興味あるところ．

ちょっと取り残されていて非常に実用的に重要なのにかかわらず，スプレッドシートというか，表型といいましたけれども，あれは結構おもしろいんじゃないかと思うんですけれども，あまり理

プログラミングの研究テーマ
- プログラミングパラダイム
  ロジックプログラミング，関数型言語，オブジェクト指向，表型
- 処理系の理論と実践
  コンパイラ自動生成，最適化，属性文法，支援環境
- 言語の理論と実践
  型理論，意味論，言語設計，標準化，評価
- 新しい可能性
  並列・分散，視覚言語，例示プログラミング，programmingless-programming

「学びて思わざれば暗く，
思いて学ばざれば，即ち危うし」

論的な取り扱いをされていないんじゃないかということです．

コンパイラに関してはずいぶんいろいろ出ているんでいまさら言うまでもないんですけれども，残っているとすれば支援環境関係．この勉強は少し理論的枠組みでどんなおもしろいことが出るかと．

ほかに何か新しいことといったら，いま非常に盛んなのは並列・分散・並行，この辺ですよね．視覚言語とか例示によるプログラミング，それからプログラミングレスプログラミング．つまりプログラミングをしないでもプログラミングができてしまうとか，およそいままでの範疇のプログラミングにないような形でプログラミングができるということがもし考えられたら非常におもしろいんじゃないかと思います．

最後に，こういう場合の常套的な結論は，「学びて思わざれば暗く，思いて学ばざれば即ちあやうし」ということになりますが，ここでは，もちろん「学ぶ」=「理論」，「思う」=「実践」という意味ですけれども，これはあまり当たり前すぎます．もし皆さん理論と実践を共存させようと思う場合に限り，それをお考えくださいということで，すべての人にお薦めするわけではありません．

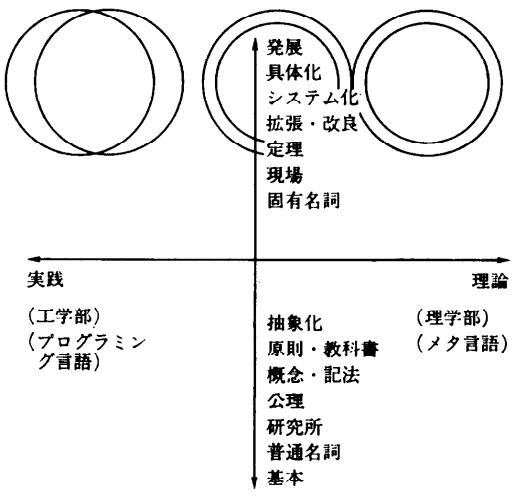
つまり唯理論派，唯実践派といって，これはそれぞれ非常にいいと思うんですね，私は．つまり理論で本当に楽しくていい仕事をしている人は，それはそれでやってください．あんまり実践の人が口をはさまないほうがいいと思うんですね．逆に非常に役に立つプログラムを書く人は，あんまり理論的なこととか悩まなくていいです．そのかわり非常に世の中に役に立つ，いろいろいまわれわれがお世話になっているプログラムがありますよね．そういったものをぜひおつくりくださいということを言いたい．

われわれはどうも共生派みたいなところにいるんで，そうだとすると，ときどきスイッチしてほかがどうなっているかというのを伺わなければいけない．この研究会はというと，この3種類の人が多分それぞれいるんで，あの人はどうかなということで，あんまりこっちの人とこっちの人によけいなことをいわないように．あと全体が仲良くやりましょうということです．

村井　さっき安村さんがおっしゃったことにも関連しますが，もしそういう何とか派というのが本当に分かれているなら，たとえば実践派は理論派ですら分かるような言葉で実践の結果を語らなければいけない．さっきオーバーヘッドといいましたけれども，そういうような努力をお互いにしあうのが研究会という場で一緒に勉強しているいいことだと思います．いまあえて理論派にも分かるような言葉で実践派が語るなんていいましたけれども，もちろん逆もそのとおりです．そういうような努力が必要だろうなというのがその研究会に関して．

それからもう一つ．まさかと思いますけれども，プログラミングの理論を考えて，そしてプログラムのコンパイラをつくったら，理論と実践がコンプリートしたなんて思っている人はいないでしょうね，この中に．どうもそういうことを考え

（理論 vs. 実践）vs.（基本 vs. 発展）
- 主張：「理論的」と「基本的」は独立．



ている人がいそうで，そのできたプログラムがどうなるかということは一切考えていない人が何人かいるような匂いがしたので，ちょっと最後にそれだけ．それがあったら怖いなと．

竹内 私は理論と実践という区分けはもういいやという感じです．情報派という言葉が出ましたが，私はやはり芸術派，アート派をつくりたいですね．ただし，アートというのは職人肌という意味でのアートじゃありません．21世紀の計算機では，もう事務計算のプログラムを新たにつくる必要はなくなり，本来の意味での芸術プログラム，あるいは実用的に無価値なプログラムをつくることが新しいジャンルになると思っています．

並列分散は1990年代に大いに進むでしょうが，その先には遺伝子型，つまりDNAのようなプログラムを書く時代がくる．これが未来のソフトウェアです．そして発生系，つまり遺伝子を発現させるための発生系，それがアーキテクチャというかマシンに相当するわけです．そういうところに入っていく前に，なんらかの理論が絶対必要でしょう．それは生物学に学ぶ理論かもしれませんが，どういうものか分かりません．

もう一つ私が夢見ているメタな理論があります．自然言語の解析をアルゴリズム論でやると，文章の長さ N の肩に2とか3とかが乗っかる手間がかかりますね．でも人間にとってそんなはず絶対ないんですよね．並列だったら解決するというわけでもなさそうだから，アルゴリズム論と情報理論の合いの子みたいな理論ができそうな気がします．実は，アルゴリズム的情報理論という理論がすでにありますが，私の夢とは筋が違っています．

私が欲しいのは，現実の世の中の確率分布をうまく利用して，確率を使わない決定性のアルゴリズムなのに，確率的にほとんど線形時間で問題を解決してしまうアルゴリズムに関する一般論です．たとえば，ハッシュ法とか，キャッシュメモリとかはこの概念で説明できる工夫なんですね．うまく説明できませんが，「情報論的アルゴリズム理論」といった感じのものでしょうか．プログラムの理論じゃないですけれども．

上田 理論と実践の乖離があるかないかというと，いままでの両研究会の発表を聴いていても，実践の人は理論の発表で何を言っているのか分からないに違いないし，理論の人は実践の発表はつまらんと思って聞いているに違いないと勝手に推察しているんです．が，せっかく統一したんですから，できるだけお互いに分かる言葉で話すということを希望しておきたいと思います．

さて，実践と理論の区別を，プログラム言語でものを書くことと，プログラムについて議論をするメタ言語でものを書くことの違い，ととらえます．すると，その区別と直交して，物事の基本に関する研究と，基本的なものを発展させる研究とがあると思うんです．なぜこんなことを言うかというと，たとえばここに論理プログラミングの理論があったとする．それをみて，その本質を見極めることなしに，その理論を拡張しましたといって論文を書く．あるいは，ある計算モデルを，必然性もなく拡張して，全然使いものにならない計算モデルをつくって論文を書く．あるいは，アメリカのすぐれたソフトウェアをまね損なって日本で下手なソフトウェアをつくる．研究開発では発展ということも重要ですが，こういう，基本に立ち返ることを忘れた発展が多いのではないかと恐れているわけです．

基本的なことって，必ずしも（上で述べた意味で）理論的なことではなくて，プログラムをつくるときにモジュラにつくれとか，あるいは並列計算ではローカリティに気をつけろとか，数学の言葉じゃないけれども基本的なことというのが山ほどあるんですね．そういうものを蓄積していきたいし，この研究会の共通の文化にしていきたい．

も一つ注文を出しましょう．これは情報派の弊害かも知れませんが，いろんな場で話すときに固有名詞をやたら乱発するんで，お前には分からないだろうと言われているような気になってしまうんですね．具体的なシステムについての議論も重要ではありますが，計算機科学といえるためには，固有名詞は必要以上に乱発せず，普通名詞でものを語るようにしていきたいと思います．

難なくまとまってしまいましたが，終わりたいと思います．

司会 せっかく難なくまとめていただいたので余分なことは言わないようにします．長いことありがとうございました．パネリストの皆さんと参加していただいた方に拍手をして終わりたいと思います．（拍手）