ソフトウェア工学6－1
（1978. 5. 31）

スタンフォード大学人工知能研究所

玉井 哲左雄（三菱・総合研究所）
吉村 鉄太郎（管理工学研究所）

訪問先概要

スタンフォード大学の人工知能研究所（A.I.ラボラトリ）は，人工知能を中心とするコンピュータ・サイエンスの領域で先進的な業績をあげてきている。とくに，今回の調査テーマの一つであるプログラムの検証系の研究に関しては，D.C. Luckham，F.W. v. Henke，N. Suzuki などによって精力的な研究がなされ，また，現在動いている数少ない自動検証システムの一つを，作成したところでもある。

A.I. ラボラトリは，スタンフォード大学のキャンパスからはかなり離れた静かな見晴しのよい高台にある。建物は，木造平屋で扇状の独特の形状をしている。内部は，一人または二人が在室する研究室と計算機室などからなる。多くの研究室ではドアが開けはなされたままで，きわめて開放的な雰囲気である。いうまでもなくここには優秀な研究者がそろっているが，とくにプログラムの検証に関しては，Luckham等のほかに，現在，Z. Manna（我々が行ったときは，たまたま外出していて会えなかった）や，IBMのワトソン研究所からサバティカルで来ている J. King（彼には，別に会ったので，その時の模様については別項で述べる）などがいる。J. Mac Carthy にも会ったが挨拶程度であった。また，R. Weyhrauch が，訪問の予定外ではあったが，廣瀬先生の知己ということで，議論にも加わったし，ラボの中の案内等をしてくれた。

調査内容

David C. Luckham との話し合いの内容は，次のようである。Luckhamにはあらかじめ質問状を送ってあったので，我々が今回の調査の経緯，目的等を簡単に説明したあと，その質問状の，とくに検証システムの将来の展望について，Luckhamは自分の意見を述べた。

我々も質問で述べたが，将来の自動検証システムに関し，次のような問題点があげられるだろう。

(1) どのような使われ方が考えられるか
(2) どんなプログラムを対象として扱えるか
(3) 使うのがどの程度難しいか

(1)については，とくにプログラムが間違っていた場合によい情報が得られることが，重要だろう。このためには，symbolic executionの方

がよいという意見もあるが，検証系で検証条件を生成する時にはsymbolic executionと実質的に同じことを行っているわけで，証明不可能な検証条件から対応するプログラム部分を検出することができるはずである。しかしその際に，会話型で利用者が誤まりの個所を発見しやすいような工夫が必要となろう。我々の，反例を示すというような方向はどうかとの質問に対し，それも一つの考えられるやり方だが，技術的に難しい点が大分あろうし，どの程度役に立つかはっきりしないのではないかとの意見だった。

(2)については，対象言語の問題として考えると，言語の設計の問題と関連するだろう。たとえば，ポインタの存在は検証の上で大きな障害である。PASCALのポインタの機能は，たとえばPL/Iのそれに比べるとずい分制限されたものである。それでもPASCALのポインタを含むプログラムは検証するのが多くの場合困難である。しかしもし，ポインタを持たない高級言語を設計し，ポインタのようなものは，もっと下位のレベルで，たとえばコンパイラが知っていても，利用者は知らなくてもよいという形にすれば，コンパイラの正しさの証明が別に必要であるにしても，一つの解決にはなるかもしれない。

(3)については，やはり難しいといわざるをえない。一つの解決の方向は，仕様の標準化である。この標準化は，問題領域の概念に対して行うことにより，仕様のかきやすさと検証のしやすさを導く。標準化はおそらく分野別に行う必要があり，たとえばsortに関てはどういう標準化をすべきかについてかなりのことが分っている。この辺の考え方については，[7] IFIPのLuckhamの招待論文に，やや詳しい話がある。

別の方向として，仕様の必要ないような事柄の検証，たとえば配列の添字に使われている変数の値が，その配列の上限を越えないとか，NILポインタをポインタとして参照しないとかいうような実行時のチェックに属することは，技術的な可能性も高く，利用者も使いやすいので，近い将来実用化されると思われる。このような方向で解決できる問題がどの程度あるのかを見きわめることが，現在の一つの課題である。

また，我々の，人手による検証についてはどう考えるかという質問については，人手であるプログラムを検証したといってきた学生の検証と，自動検証系の結果とを比較し，学生の証明に誤りを発見したので，それを学生に伝えたところ，そんなはずはないと主張するので，実際に違っている（ぬけている）点を示したら，“ああ，それはtrivialだ”といったという例を示し，人間は，やはり考えおとしが多いということを，いっていた。

ゼロックス・パロアルト・リサーチ・センター

訪問先概要

Xerox社のPalo Alto Research Centerは，いくつかの研究部門を有するが，その中のコンピュータ・サイエンス関係も，多くの優秀な研究者を擁するものとして名高い。Stanford大学でかなりの能力をもったverifierをつくってPh.Dをとられ，Carnegie-Mellon大学に2年ほど勤められた鈴木氏が，11月初めからこのXerox Palo Alto に移られたので，訪問した。

鈴木氏によれば，コンピュータ・サイエンス部門には，約30名ほどの人間がいるとのことである。意外に数が少ないのに驚いた。ハードウェアをやる人とソフトウェアをやる人と，どの位の割合かとの我々の質問に，大体皆，ハードウェアもソフトウェアもやるといわれて，また驚かされた。格別優秀な人が，数少なく集まると，よい仕事ができるということであろう。

調査内容

我々の質問状は，前もって鈴木氏に送ってあった。初めに，我々が現在従事しているプロジェクト，および，今回の調査旅行の趣旨を簡単に説明した。以下，鈴木氏の語った内容は，おおよそ次の通りである。

ソフトウェア・ハウスの技術力の蓄積のためなら，verifierをつくるよりも，

(1) 言語の設計

(2) そのコンパイラの作成

(3) エディタの作成

というように，ソフトウェア開発のシステムをつくりあげるのが先決であろう。たとえばXeroxでは，MESAという言語を開発し，それを使用するソフトウェア開発システムをつくって，現在，ほとんどすべてのソフトウェアは，MESAで開発されている。アメリカでは，使われている機械が，ほとんどIBMかDECであることもあって，ポータビリティということは余り問題にならない（と彼は，断定した。ポータビリティについてはともかく，このようなソフトウェア開発の環境をまず作らなければならないというのは，もっともである。しかし，もっと商業的なソフトウェア

の生産を行っているソフトウェア・ハウスでは事情はどうなのだろうという疑問は残る。これに関しては，今回の調査の他のグループが，確かUCCで，ドキュメントやプログラミングの標準化についてきいたところ，IBMの基準に従っていればよいから，標準化というのは余り問題にならないといったという話が，逆の意味で鈴木氏のいうことの正しさを，裏書きしているようにも思えた。）

さて，プログラムの検証については，現在complexityの問題が大きな壁となっている。だから，実用性という意味では，たとえばformulaのmanipulaterだとか，pretty printerだとかが，現在考えられる有用な道具であるといえるだろう。

complexity の問題は，たとえば配列を扱うと，すぐにでてきてしまう。配列を対象とする簡単な問題で，もうNP完全なものがあるから，いわゆる inductive assertion methodをとると，手間はexponentialに増大してしまう。1つの例として，Knuthの本にある多項式の演算を処理するプログラムに検証を試みたところ，紙に2枚のプログラムに対し，検証条件が12～3枚できてしまい，とても証明できそうもない結果となった。この検証条件を減らして短くするには，非常な苦労がいった。

そこで，検証システムの方向で，現在のところ，全く自動的なものとして何ができるかというと，たとえば配列の境界の検査というような，利用者の仕様を必要としないような問題で，多項式の手間で（exponentialでなく）できる部分の多いものがある。これは，利用者にとっても，仕様をかく必要がないという意味で，使いやすいものであろう。

ポインタがあると，これは配列以上に，indexingが重なるという意味でcomplexityの壁が大きく，おそらく，inductive assertion のような手法では，扱えないのではないかと思う。

並行処理プログラムの検証については，まずO.S.のプログラムでも，実際に並行処理になるプログラムの部分というのは，きわめて少ないのではないか。現在，数学的な問題として，informalな証明を与えた例はある。しかし実際上は，weak correctnessは余り問題ではないだろう。

順路式（path expression）に関しては，これで表現できない問題があるという意味で，限界がはっきりしている。従って，順路式に関する関心は一般に小さくなっているが，実際の問題で順路式で表現できないのがどの位でてくるのか，などはまだ分っていない。

ＩＢＭパロアルト・科学センター

訪問先概要

ＩＢＭの数多いScientific Centerの一つである。Palo Alto 周辺はいろいろな研究所の類が集まっている。J.C.Kingは，本来ＩＢＭ Thomas J.Watson Research Centerの所属であるが，現在サバティカルという制度で，１年間スタンフォード大学のA.I.ラボラトリに来ており，Palo Alto の Scientific Centerにも officeをもっている。

我々の最初の予定では，Thomas J.Watson Research Center を訪ねる予定であったが，それが都合で出来なくなったところへ，Kingがスタンフォードの方にいるということが分って，こちらの方で会え，当初の目的をある意味で果たせたことは幸いであった。なお，Kingは，Stanford A.I.Lab.では，やはり世話になったR.Weyhranchと同室であった。

調査内容

J.C.King に会う予定は，我々がスタンフォードの方へ来てから初めてたてたものなので，我々の用意した質問状は，その場で見せることになった。Kingは，その質問状に目を通したあと，大体質問の趣旨に沿って以下のような意見を述べた。

(1) 検証システムのニーズについて

プログラムの検査（checking）として使われているというのが，一つのみちであろう。しかし，先日，数値解析の誤差の分析で有名なWilkinsonと話をしたが，彼の扱っているような浮動小数の世界では，このcheckingには無理がありそうである。また，問題の明確でない，たとえば，天気予報とか医療診断などの分野にも，検証は向かないであろう。

検証を適用するのが適しているのは，プログラムの背後にある概念が簡単で明解であるようなものに対してである。たとえば，事務処理がそうである。また，システム・プログラムは，事務処理と比べれば，データ構造などがやや複雑ではあるものの，適用の見こみのある分野といえる。

最近では，マイクロ・プログラミングに検証を適用しようという試みが注目されており，たとえばＩＢＭ Thomas J.Watson Research Centerでは，この研究をすすめている。

検証を，プログラム開発のどの段階で適用すべきかについては，どの段階（設計，コーディング，テスト）でも使えるということになるが，コストの考慮とともに決めなければならない。現在のところ検証システムの使用は高くつくから，使用の効果が充分ないと使う意味がない。

(2) 検証システムの将来の見通しについて

いつごろ実用的な検証システムが作られるかという見通しについては，現場で使われるようなものができるのは，7年後だといった人がいる。たとえば現在のコンパイラのような使われ方をするようになるのは，自分の感じでは，20－30年はかかると思う。

将来の検証システムが，対象言語としてどのようなものまで扱えるようになるかについて，たとえばFORTRANが扱えるかどうかについては，次のように考える。FORTRANは，たとえばポインタがないといった，むしろ検証には便利な特徴もある。数学的にすっきりしていない要素を含んでいることは確かであるが，実用的な方法で解決できると思われ，FORTRANを対象とするから検証系が難しくなるという点は，比較的少ないと思う。ここで，我々の，"確かLondonの論文に，Torontoでのプログラムの信頼性に関するワークショップでの結果として，FORTRANプログラムの検証にとって障害となるのは，おそらく，

(a) COMMONの存在

(b) EQUIVALENCEの存在

(c) GOTO文の多用

の3点ではないかということになった，とあるのを読んだことがあるが"という質問に対しても，Kingとしては，それらはそれほど大きな障害とは思えないと語った。

検証システムを使いこなすために，プログラマにどの程度の知識，訓練が要求されるかについては，プログラマの数は今後とも増大するだろうが，検証をとり扱うのは，やはり"良いプログラマ"でなければだめであろう。良いプログラマは，ある意味で検証にあたることを，自分なりに実行してきているはずである。

プログラマにとって，仕様（specification）や表明（assertion）を書くという負担の増大があり，また検証システムを動かすのにコストがかかっても，それが2倍程度の手間やコストの増大であれば，テストの短縮によって充分payするはずである。

実用的な検証システム完成のための大きな障害としては，specification の問題があげられよう。よい specification language がつくられるには，多くの時間がかかろう。Luckham のいうように，分野別の知識による仕様の標準化ということは，重要な考え方となるだろう。また，配列・ポインタはやっかいなものである。これらは，もしかすると，specification の問題として解決される道があるかもしれない。

(3) 検証技法について

Floyd 流の inductive assertion method が今後も検証システムの中で重要な役割でありつづけるかどうかについては，assertion というのは人間にとって自然な考え方であるから，イエスという答えになる。Wegbreit 等の subgoal induction というのを，最近読みなおしてみてやっと理解したように思うが，inductive assertion と本質的には同じことではないかと思う。問題によって，一方の方法のほうが具合がいいというようなことは，あるだろう。

このあと，抽象データ型，順路式，Alphard や Euclid などの新しい言語などに関する質問事項があったが，この辺の問題については King はそれほど興味をもっていないため，簡単におわった。

(4) 検証系以外の自動化ツール類について

ツールの例として，King 自身の EFFIGY の話があった。EFFIGY については，論文もでているので我々も大体内容を知っていた。このシステムは PL／I に似たきわめて簡単な言語を対象として，プログラムのテストを助ける機能をもつものである。基本的な手法は，いわゆる symbolic execution と呼ばれるものである。King によれば，EFFIGY の一番の特徴は，利用者の要求の程度によって，いくつかの段階に応じた使い方ができる点にあるという。簡単なのは，データを与えて実行結果を示す，その次の段階では，実際のデータでなくシンボルの実行による結果を示す，さらに上のレベルとしては，表明を入れることにより，検証も可能となるというわけである。

全体の印象としては，やはり大学の関係者と違い IBM の人間ということで，Luckham 等の考え方と違うという感じをうけた点があったと思う。たとえば，すでにかいたが，FORTRAN に対する考え方で，FORTRAN の使用は，今後多少減ることはあっても，無視できるほどなくなることはないとして，FORTRAN を対象とする検証システムの意義をかなり認めている点などである。

カリフォルニア大学（ロサンゼルス）

訪問先概要 （Professor Jerry Popek）

Popek教授は，Computer Science Dept.のassociate professorである。このDept.はUCLAのSchool of Engineering and Applied Scienceに属しており，２０人以上の教授を有している。UCLAはその名の示すように州立大学であるUniversity of Californiaのロサンゼルス地区の分校であり，カルフォルニア大学全体としてはこのほかに８ケ所のキャンパスをもっている。この中で計算機科学の分野で知られているのはUCLA，UC at Berkeley，UC at Irvineと云った所であろう。BerkeleyにはUCの本部があり，計算機科学の分野ではソフトウェア・エンジニアリングの分野で有名なRamamoothy教授，データベース（INGRES）の専門家であるStonebraker教授あるいは，医療データベースの開発やソフトウェア・エンジニアリングの教授で知られたWasserman教授らがいる。

さて，UCLAにおけるプログラムの検証分野の主要な研究者は，今回訪問したPopek教授の他に，D.F.Martin教授である。またこれに関連した分野として，J.A.Goguen教授が人工知能の研究を行っている。同教授は現在，英国Edinburgh大に滞在中で，Burstall教授とともにデータ抽象化の研究を進めている由であるから，結局この方面の主要な研究者は少なくとも３人いることになろう。

調査内容

Popek教授のグループにおける主要な研究テーマの一つは，いわゆるComputer Security問題である。すなわち情報処理システムの安全性と機密保護能力の向上に関する諸問題が研究対象とされており，このうち特に下記の４分野について研究が行われている。

(1) オペレーティング・システム（OS）
(2) データ・マネジメント
(3) コンピュータ・ネットワーク
(4) 分散型システム（Distributed System）

このうち特に(1)および(2)の分野で，より信頼度の高いシステムを実現するための手段としてプログラム検証系に注目している。Popekらはオペレーティング・システムとデータ・マネジメントにおけるsecurity問題解決の一手段として，いわゆるsecurity kernelのアプローチを採用して

いる。これはデータ・マネジメント・システムの運転基盤としての，極めて高い機密保護能力と信頼度をもつOS（ secure operating system ）を実現しようとする試みである。そのためにはOS自体が仕様を忠実に守って作成されていることを，何らかの方法で証明（ certify ）しなければならないが，OSそのものがかなり大型のプログラムであるから，プログラム検証技術をそのまま適用することはできない。

この問題を解決するためにPopekのグループがとったアプローチは，大略下記のようである。OSの中で，機密保護機能に関する部分がごく少数のモジュールに集約されるように設計を行う。このようなモジュールをsecurity kernelと呼ぶ。そしてこのsecurity kernelを中心とする極めて少数のモジュールに対して，プログラム検証を行いその正当性を確認する。Popekらは強化版のUNIXをベースとして，上記の方法を採用したOS及びデータ・マネジメント・システムを実現した。

Popekらのグループにおける検証技術の位置づけは大略上記のようであり，機密保護と云う実際的な立場からプログラム検証技術を見ている点が注目に値すると思われる。

次に検証方法自体の構成は，全体として下記の諸段階から成る。

(1) まずプログラムの抽象モデル（ abstract model ）を作り
(2) これに対する仕様記述（ specification ）を行い
(3) 次に抽象モデルから具体モデル（ concrete model ）への書き換えを行い
(4) 具体モデルに関する仕様記述を行う
(5) 具体モデルにもとずくプログラミングを行い
(6) 検証を行う

実際に仕様記述を行ってみると，仕様部分がプログラム本体の数倍の分量に達することが明らかになった。

Popek教授の見解によれば，使いものになる検証系の実現に関する最大の障害は理論面よりもむしろ技術的な面に存在しており，この方面の解決に十分な努力をはらうならば，実用化は時間の問題であるとのことであった。他の訪問先における意見にくらべ，この見方は一見極めて楽観的であるように思えるが，その前提条件として検証の対象となるプログラムをごく限られたものにしぼる（たとえばsecurity kernel）と云うことであれば，それなりに意味のある考えと云えよう。

一方hand verificationの有用性に関しては否定的であって，hand compilationが無価値なのと同様であるとの意見であった。もっともこれは第三者が他人のプログラムをhand verifyする事を指して云った

もので，知的努力を集中するに値する重要なアルゴリズムであればともかく，日常作業として書かれる他人の平凡なプログラムを対象にした場合に，正確な検証作業をルーチン・ワークとして期待することは，事実上不可能だと云う趣旨である。したがって自分自身の書いたプログラムの信頼性を向上させるために行うのであれば，それなりの意味をもつだろうと附言している。

テキサス大学（オースチン）

訪問先概要

テキサス大学は州立であって，州内に数ケ所のキャンパスをもつが，計算機科学の分野では州の首都オースチンのキャンパスが国際的にも有名である。ここの計算機科学科は近年めきめき頭角をあらわしており，特にデータベースおよびソフトウェア・エンジニアリングの面ですぐれた業績をあげている。

Raymond T. Yeh教授はソフトウェア・エンジニアリングでは世界的に有名である一方，計算機科学科の主任教授でもあり名実ともにこの学科の中心人物である。

プログラム検証技術に関しては，同教授のグループに属するスタッフが注目すべき多数の論文を学会誌上に発表しており，テキサス大学（オースチン）はこの分野における米国内の一つの中心として大きな影響力をもっている。

現在同教授のグループがとりあげている研究テーマの一つに，Decision Support System（意思決定支援系）があり，そのサブテーマの一つにsemantic networkと呼ばれる高水準データベース・システムがある。定理証明やプログラム検証等の技術は，このsemantic networkへの応用と云う立場から，特に研究が進められているようである。なお，訪問中第6回テキサス計算機科学会議が学内で開かれていたが，招待講演者として参加したIBMのHarlan Mills博士にも紹介され，意見交換を行う機会を得たのは予想外の幸運であった。

調査概要

Decision Support System (DSS)

計算機科学科における総合研究の一つとして，将来のDecision Support System (FDSS)に関する基礎的な研究がYeh教授のもとに行われている。FDSSのねらいとするところは，大規模かつ分散型のDSSであるばかりでなく，多種類の問題分野に対して有効であり，しかも各種環境の変動に対する適応力をもつシステムである。このようなFDSSに対しては，適応能力，知能および信頼性が要求される。現在計算機学科で行われている各種の研究，たとえばデータベース，コンピュータ・ネットワーク，プログラム検証などの成果はこのFDSSプロジェクトにおいて重要な役割をもつことになると予想されている。

FDSS実現に関して現在指摘されている問題点には下記がある。

(1) システム・パラメータ相互間の不連続性

(2) 分散型データ

(3) 知識データベース

(4) 推論機能

これらの諸問題へのアプローチとして，ソフトウェア・エンジニアリング，人工知能，性能評価モデル，データマネジメントおよびコンピュータ・アーキテクチャ等各分野の知識を総合的に利用しようとするのがテキサス大学における基本的な研究方針であると考えられる。そしてプログラム検証技術も，たとえばLarry Flonによる並行処理プログラムの検証のように，FDSSプロジェクトの一環として位置づけられているようである。

プログラム正当性証明の方法論

テキサス大学は，verifierの研究では米国内で重要な地位を占める大学の一つである。特にinteractive verifierに関しては，W.W. Bledsoe, D. Goodらの成果が有名である。また最近カーネギー・メロン大から移ったLarry Flonによる並行処理プログラムに関する検証技術の研究も興味深いものがあった。この研究でFlonはDijkstra流のweakest precondition formalismを，並行処理における最大の問題であるデッドロック／資源競合問題に拡張した。このことから並行処理プログラムにおいても，プログラミングの前に正確な仕様を記述し，またプログラミングと並行して検証を進めることの可能性がより大きくなったと云える。

一方C. Reynolds による研究は，プログラムの構造を数学的証明の構造に対応させることによって，数学における証明理論への成果をプログラム検証技術に応用しようとするものである。Reynoldsは現存する5種のinduction methods（inductive assertion, subgoal induction, structural induction etc）がある意味で等価であることを近々論文として発表するとのことで，この点は我々にとっても大きな興味がある。

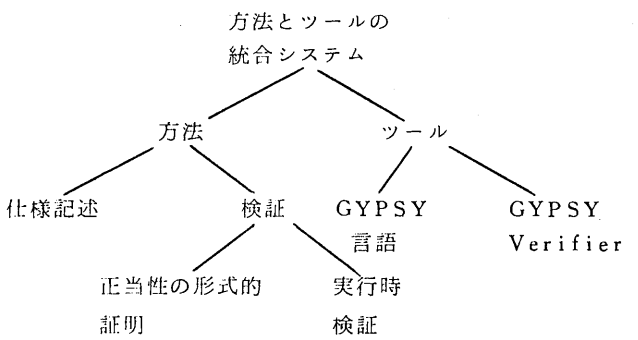
方法論とツールの組織化

D. Goodは，ミニコンピュータ・ネットワークの信頼性向上の一手段としてのプログラム検証技術の実用化を研究中である。メッセージ処理のための分散型システムにおけるソフトウェアの信頼性を強化するための手段として，Goodは下記のような総合的なアプローチを採用している。

(1) 正確な仕様記述
(2) プログラミングと検証の同時進行
(3) プログラムの分割
(4) 検証可能性の実現ののちに能率化を行う
(5) 自動化された各種ツールの整備

このためのツールの一つとしてGoodらはプログラミング言語GYPSYを発表している。Goodらは，これらの方法とツールを総合したシステム（図7－1）を考え，これによってソフトウェア開発を行おうとしている。

図7－1　方法とツールの統合システム



Goodはまた検証技術の実際的なあり方として，formalな証明と実行時のvalidationの調和のとれた組合せが重要であるとの意見をもっており，この点もうなずけるものがあった。

Harlan Mills

我々の訪問中たまたまTexas Computer Conference（第6回）が開かれていたが，冒頭にソフトウェア・エンジニアリングの国際的権威の一人であるIBMのDr. Harlan Millsが基調講演を行った。

Conferenceの主催者Yeh教授の招きにより我々もこの講演を聞くことができたが，ソフトウェア・エンジニアリングの意味に関して傾聴すべき内容を含んでいたのでその要点を記す。

Millsによれば現在ソフトウェア・エンジニアリングにおいて最も重要な問題の一つはソフトウェアの数学的な性質とその正しさに関する研究であるとしている。

ソフトウェア工学におけるより具体的なレベルでの課題としては

ソフトウェア品質をいかに測るか
仕様記述／設計のための言語
ソフトウェア開発におけるオートメーション
ソフトウェア開発におけるhuman factor
マイクロコンピュータの影響

をあげている。

数学的なアプローチが効果をあげると見られる分野のなかには，下記が含まれよう。すなわち

仕様記述
設　計
正しさの証明
文書化
ハードウェアの定義記述

今までの数学的アプローチによってなされた（ソフトウェア・エンジニアリングにおける）発見のうち主なもの三つは；

順次プロセスに関する信頼度の高い設計法
順次プロセスに関する信頼度の高い同期方法
ソフトウェア／ハードウェアの統一的な取扱いによる抽象機械概念

である。上記の課題や成果を前提として「ソフトウェアとは何か」を改めて考えてみるとそれは

人間と機械の調和のとれた協力関係のための，論理的な指導原理

と云ってもよいだろう。

カーネギー・メロン大学　計算機学科

訪問先概要

カーネギー財団とメロン財団の名を冠したこの大学は，U.S.スチールの町ピッツバーグ市の中にある。ここは，Simon,Newellといった人工知能の分野で先駆的な仕事をした人達をはじめとして，コンピュータ・サイエンスの分野に多くの研究者を集めている。

計算機学科で現在活躍している人達は，B.Wulf,M.Shaw,A.Jonesなど数多い。また近年，日本人研究者も多くここで研究をされてきている。SOSP－6で発表された益田氏が，1969年に来られたのが，本人によればおそらく最も古く，その後たとえば今回の調査に参加された土居範久助教授（今回のカーネギー・メロン大学訪問の手配は，先生による）も1年間研究され，また現在，木村泉助教授が籍をおかれている。

ここの計算機は，DEC11,DEC10が各1台あり，またPDP10が，数多くあるというように，大変ぜい沢である。

調査内容

今回は，N.Habermannを中心に，彼のもとにいる若手研究者達と話をした。

(1) Nico Habermann

彼は，とくに人手で行う検証について，および，検証システム以外の自動化ツール類について議論した。Habermannの主張は,大体次のようである。

現在試作されているような方向での検証系（verifier）が，かなり大きなプログラム，たとえば2,000行ぐらいのものを，近い将来に検証することができるようになるとは，思えない。仮にできるとしても，それを使うことのできる人は，高いレベルの知識をもった人でなくてはならず，それにしても，少なくとも10年はかかるだろう，従って，完全に自動的な検証系がつくられる可能性については，悲観的に思う。

そこで，人手によって検証することは重要になる。そのための，手引・マニュアルの類は，自分も作るべきであるという考えをもっている。検証をするために便利な規則がないのであれば，発明しなければ

ならない。たとえば，現在我々は，データの不変成分を見つけだすよいルールをもっていない。そのようなことで，プログラムのSpecificationを上手にかく方法が，確立されていない。

いま一つの方向は，検証システム以外の，プログラムの正当性をたしかめるための自動化ツールを開発していくというものである。この考えで，現在我々は，プログラムのデバッグの環境をつくるシステムを，試作中である。それは，次の3つの構想からなる。

(a) 仕様のためのツール

(b) テスト環境のツール

(c) テストのツール

(a)は，モジュールごとに，そこで使われるデータのタイプ，データの対象（変数，配列といったもの），関数（その仕様として，引数や関数値のタイプ，値域などを与える）などを仕様として標準化してもち，それらをファイルとしてとり扱うことができるようにする，というものである。したがって今の段階では，仕様といっても，モジュールの機能を記述することはできない。

(b)は，会話型でテストするためのツールで，モジュールごとに，そこで用いられている関数（演算）を，適当なデータを与えて実行させてみたり，変数の値を表示したりする機能をもつものである。これについては，デモンストレーション用であろうが，すでに試作ができており，後で実際にデモンストレーションをしてくれた。

(c)は，まだ手がつけられていないが，実際にプログラムを実行させてテストするものである。モジュールに変更があった場合なども，その履歴を管理するなどの機能を，もたせるはずである。

デモンストレーションで見せた例は，マルチ・プログラミングにおけるタスク管理をシミュレートする簡単なプログラムで，タスクの出入や優先度の変更に応じて，プログラムがどのように動くかを，表示する。なかなか面白いみものであった。

(2) Paul Hilfinger

P. HilfingerはPh.Dの学生で，プログラム言語ALPHARDのプロジェクトに参加しており，またプログラムの検証についても興味をもっている。彼は主として，検証技術の将来の展望，およびALPHARDなどの言語について議論した。

現在の検証技術の問題点は，実用的なspecification languageがないことと，定理証明系(Theorem prover)の能力が低いという2点だろう。検証方法としてのinductive assertion法は，今後も検証系の中で重要な役割をもちつづけると思う。

検証系の扱う言語として，FORTRANやCOBOLも扱いうるはずで，SRIでは，COBOLのverifierを作っているように聞いている。

ALPHARDのような言語が広まるかどうかについては，ドキュメントの問題が大きな要因となるだろう。たとえばAlgol 68も余りはやらなかったが，1つの原因はよみ易いドキュメントがなかったからである。最近では，たとえばPaganの本など，分りやすいのがでている，ALPHARDも今のところよいドキュメントがない。

現在，CMUのPQCCというプロジェクトでは，効率のいいALPHARDの処理系を作ろうとする予定でいる。

(3) Sten Andler

S.Andlerはスウェーデンからの留学生である。彼は，自分の研究である，Algol 68に抽象データ型をもちこむという仕事について，主に話した。Algol 68を選んだのは，CMUにいい処理系があって，割とよく使われているからしい。インプリメントの形式は，プリ・プロセッサが，抽象データ型の要素を付加したAlgol 68を普通のAlgol 68におとすというものである。Algol 68のmodeは，当然のことながら，この目的のために便利だったようである。

(4) David Jefferson

D.Jeffersonは，鈴木則久氏がCMUにいたころ面倒をみた学生だったということもあって，とくに検証について興味をもっていた。彼は，大体我々の質問状に沿って話をした。

- 検証システムのニーズについて
  よくフォーマライズされた分野に，検証はむいている。ヒューリスティックが入ったりするもの，人工知能などには向かないだろう。事務計算や，多くの科学計算の分野にむいている。
  プログラム開発のどの段階で使われるのがよいかについては，いわゆる段階的洗錬という設計場面で使われるのが理想だろうが，それが唯一の道とは限らない。
- 検証システムの将来の見通しについて
  たとえば，配列の境界のチェックなどは実用的だし，Suzukiの研究にもあるように，すぐできるだろう。

FORTRANやCOBOLを扱うようになるかは，難しいにしても不可能ではないと思う。

検証システムを使うと，次の3つの結果がでるはずである。

(a) すぐ正しいことがいえる。

(b) すぐ誤っていることがいえる。

(c) combinatorial explosionにぶつかり，そのことをいう。

(c)の場合，多くは表明が弱すぎる時であろう。

将来，検証システムの障害となるのは，verifierが数学を理解せねばならないという事実であろう。たとえば，sortのプログラムの正しさをいうために，結果が与えられた入力列の置換になっていることを示すのに，大変な数学がいる。

- 新しい検証技法の生まれる可能性について

  formalismがより明確になることはあっても，本質的に簡単になるような手法は出てくるまい。

- 検証と並行したプログラム作成という技術を，平均的プログラマが習得しうるかについて

  できると思う。プログラムを書ければ，それを証明できるはずだ。

- 人手による検証について

  段階的洗錬という設計段階では，人手によって証明した方がよい。少なくとも，証明の輪郭は描けるはずである。しかし，細かいレベルになると，長い式を書いて手で証明するのは面倒で大変である。

- 証明のこつのようなものについて

  たとえば，当り前のようだが，論理式をconjunctive normal formにしておいて，各項を別々に証明する，など，こつはあると思う。

(5) Peter Feiler

P.Feilerは西ドイツからの留学生である。彼は，PDP10を数多く使って，小さな計算機を集めて，超大型計算機に匹敵する機能をもたせる，という構想のプロジェクトにおいて，OSづくりを行っている。

IRIA

訪問先概要

冒頭 Bonnet氏より下記のごとくIRIAの概要が説明された。

IRIAはフランス政府によって運営される国立の情報処理研究および教育機関で，フランス国内に数ケ所の施設をもっている。IRIAはいくつかの主要な部門から構成されているが，その内主なものは LABORIA，SESORI，STI，およびSEFIと略称される諸機関である。

(1) LABORIA

IRIAにおける主要研究機関であり，検証系などソフトウェア工学に関する研究もここで行われている。

(2) SESORI

大学や計算機メーカーなど外部との共同研究を行うほか，実験的なテーマに関してパイロット・プロジェクトを組織し，SESORIの指導のもとに外部委託する。たとえばコンピュータ・ネットワークに関するCYCLADES計画とか，高度の能力をもつ義手義足の開発に関するSPARTACUS計画がこれらの代表的なものである。

(3) STI

情報科学（informatics）の成果を，産業界に普及させるための諸活動を行う。

(4) SEFI

この部門は技術情報の収集配布や教育活動を行う。ここには情報科学に関する総合図書館としてのドキュメンテーション・センター，出版部および教育部があり，研究報告の発行やセミナーの開催を行っている。

調査内容

検証系

IRIAにおけるプログラム検証系の研究は，検証系単独に行われているのではなく，総合的なプログラム開発環境（programming environment）の実現のための諸研究の一つとして行われている所にその特徴があるように見受けられた。ここで言うprogramming environmentとは，プログラミング言語，プログラム・データベースおよび各種のソフトウェア・ツールから成る統一的なソフトウェア・システムである。そし

てこのシステムの目的は，プログラムの研究，開発および教育をより能率的に行うために，プログラマに対してプログラムに関する各種の情報を提供し，またプログラムに対する各種の操作をより効果的に行えるようにすることである。

ＩＲＩＡではこの目的のための実験的なシステムを実現し，現在更に拡張中である。このシステムはMENTORと言う名で呼ばれ，ＩＲＩＡの他Toulouse大学で学生教育用に使用されている。MENTORは次の要素から構成されている。

テキスト・エディタ（およびそのコマンド系）

プログラミング言語（Pascal）

プログラム・データベース

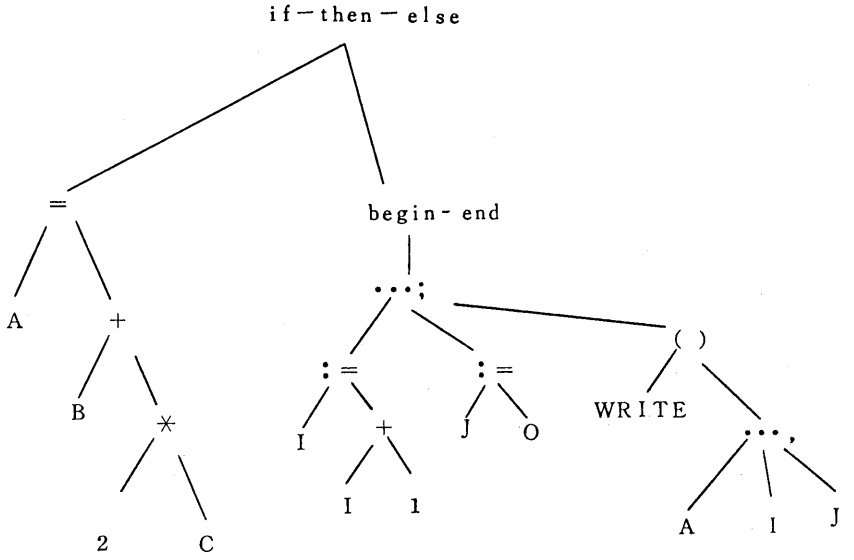
エバリュエータ（evaluators）　複数個

MENTORでは，ソースプログラムを抽象形式（abstract representation）の形でデータベースに格納する。この抽象形式は特定のプログラミング言語に依存しないので，MENTORは原理的にどんなプログラミング言語でも扱える。現在のMENTORがPascalを対象としている事は抽象形式に何等特別の制約を与えるものではない。抽象形式のプログラムは木構造で表現されている。したがってMENTORにおけるテキストエディタは，この木構造（あるいはその部分木構造）を対象とするマニピュレータであり，この点が通常の（linear character stringのための）テキスト・エディタとの相違点である。

MENTORでは抽象形式のプログラムをソース・プログラムに複元してCRT上に表示する機能をもっているが，この複元（unparsing）にはいろいろなレベルがある。たとえば下記のプログラム

```
if  A＝B＋2＊C  then
    begin  I:=I+1;  J:=0;  WRITE(A,I,J) end
```

の抽象形式は下図18－1のようである。



MENTORのテキスト・エディタ・コマンドによってCRT 上にこのプログラムを復元表示するとき，下記のように4種の表示を選択できる。

（選択１）

```
if  #  then#
```

（選択２）

```
if  A＝B＋2＊C  then
  begin ・・・ end
```

（選択３）

```
if  A＝B＋2＊C  then
  begin  #；  #；  #  end
```

（選択４）

```
if  A＝B＋2＊C  then
  begin I：＝I＋1；  J：＝0；  WRITE(A，I，J)  end
```

次にMENTORでは，抽象形式のプログラムを対象として各種の処理を行うサブシステムを組込むことができる。サブシステムの内容や種類には制限はないが，コンパイラ，オプチマイザなどとならんで検証系もサブシステムとして組込み可能である。現在J.King（IBM）流のsymbolic executorを組込むことを考えているようである。MENTORでは，これらサブシステムをevaluatorと総称している。

検証方法論自体について言えば，ＩＲＩＡの研究者達は一般に次のような立場をとっているように見受けられた。

まず，検証系の位置づけとしては，これを総合的なプログラム開発環境の一構成要素としてとらえる方針をとっており，抽象化されたプログラム構造を対象とした検証に努力を集中しているようである。

次に検証のための数学的方法論としては，inductive assertion でなくＳcott－Ｓtrachy semantics をベースとする方法を採用するとのことでこれはＬang氏から明確に表明された。

ＩＲＩＡにおける研究状況で最も注目すべきは，前述の総合的なプログラム開発環境の推進と言うアプローチであり，しかもそれを単なる方法論やdisciplineに終らせずに一つのソフトウェア・システムとして実現している点にある。環境の整備と言う点はCMUのＨabermann も別の立場から提唱し，その内容には並行処理のサポートと言う先進的な面を含んでいる。しかし，総合的なソフトウェア・システムの実現に成功したと言う点でＩＲＩＡのアプローチは極めて独自性に富むものである。

エジンバラ大学人工知能学科

訪問先概要

Ｅdinburgh大学のＤepartment of Artificial Intelligence は，Ｍeltzer，Ｍichie，Ｂurstall，Ｂarrow，Ｐopplestone といった，人工知能・計算理論などの分野ですぐれた業績のある研究者多数を擁し，ヨーロッパにあって異彩を放っている。とくに今回の調査対象の１つであるプログラムの検証の領域では，Ｂurstall を中心とするＴhe Ｔheory of Ｃomputation Ｇroupに，もといた人を含めて，Ｐlotkin，Ｄarlington，Ｔopor，Ｂoyer，Ｍoore，Ｇordon などがいて，活発な研究を行ってきている。

Ｅdinburgh はスコットランドの中心都市で，歴史的な城があり，街にも古い建物が多く残っている美しいところである。その市内にあるＥdinburgh大学は，アメリカの広大なキャンパスをもつ大学とは趣を異にし，やはり古い建物からなる校舎が，いくつか集まっているだけの規模の小さなものであるが，一種の風格がある。

調査内容

Burstallは，まず自分のところで，プログラムの検証あるいはそれに関係のある仕事で，過去にどのようなものがあったかを簡単に説明した。直接検証をとり扱ったものとしては，Boyer & MooreのLisp関数の性質を検証するシステムや，ToporのPh.D論文などがある。その後は，検証システムのimplementに関する仕事は，とくにない。また，プログラムの変換に関する研究は，BurstallとDarlingtonが行っている（Darlingtonは現在London大学にいる）。これについては，現在，学生でimplementを行っているものがいる。

これからは，すでに書かれたプログラムの検証よりも，仕様からプログラムをある程度自動的に作成していくという方向が，より実用的なのではないかと思う。プログラムの変換といった研究も，この方に役立つ話である。ただ検証に関しては，たとえばSchwartzがやっている，ある種のデータ構造をかきかえるようなプログラムの正しさを証明するというような研究が，割合面白いと思う。

Burstall自身は現在，データ抽象化の研究を，UCLAからきたGoguenとともにやっている。 specificationを構造化して書くために，従来の抽象データ型というような概念をより一般化して，問題の背後にある理論（theory）をかけるようにするという考え方である。理論に属するものは，たとえば，ベクトル空間といったものである。現在，そのための言語を設計中で，基本的なところは大体すんだが，implementについてはまだ考えていない。今，Osloにいる中島氏らが，似たような考え方から，"理論"をかけるような構造をPascalにもちこむ研究を行っているようである。

ここで，我々のほうはどんなことをやっているのかということで，我々としては，ソフトウェア・ハウスという立場から，検証という考え方やそれから出てくる道具にしても，現場で役立つものを考えたいと思っていること，そのために，現在いろいろな実際的なプログラムを，人手で検証する試みをして，問題点をひき出そうとしていることなどをいった。そして，その経験からいえることの一つとしては，仕様をかくことの難しさと重要性をいうと，Burstallは全くその通りだと，握手を求めた。

その他，Burstallの知っている検証に関する研究を，いくつか紹介してくれた。たとえば，Milnerは，とくに並行処理プログラムの証明を研究しており，Lamda calculasでかかれたものに対するある種のverifierをつくっていること。Reynoldsに，とくに配列についての仕様，表明のかき方について研究があること，などである。