ソフトウェア工学 96−27
（1994．1．21）

認知科学的プログラム理解の考え方と方法

上野晴樹

東京電機大学理工学部経営工学科

梗概　プログラム理解は、従来は主にプログラミング教育を対象としたバグのある不完全なプログラムの理解に焦点が当てられていたが、CASEの登場によって最近業務処理プログラムを対象とするリバースエンジニアリングのための技術としても注目されるようになった。本報告では、認知科学的なプログラム理解の考え方と方法を考察し、これからの研究の方向について議論する。認知科学的プログラム理解とは、人がプログラムを理解するやり方を参考にしてコンピュータにプログラムを理解させるシステムを実現することである。そのためには、プログラムにおける５つのレベルの意味とそれを表現する枠組みを準備し、マッチング処理とモデル生成によって理解を行う。今後、意図の理解まで行うにはプログラム作成過程の認知科学的モデリングの研究が必要である。また、リバースエンジニアリングの研究においては、現場の理解が不可欠であり、そのためにも産学共同研究が必須である。

Concepts and Methodologies for Coginitive Science Based Proguram Understanding

Haruki Ueno

Department of Systems Engineering, Tkyo Denki University
Ishizaka, Hatoyama-cho, Saitama-ken 350-03, Japan
ueno@k.dendai.ac.jp

Summary　Research on program understanding has been tried in the field of programming education in the last decade, where novice's buggy programs were targets of understanding. Since CASE technologies had appered this technology has become interesting candidate for Reverse Engineering. In this report, concepts and methodologies of Cognitive Science-based program understanding are discussed and future research directions are proposed as well. Development of shared knowledge base for maintaining programming knowledge is needed to facilitate R&D on useful technologies by Universities-Industries collaborative works.

1　まえがき

ここで論ずるプログラム理解とは、与えられたソースプログラムの中から各種の意味的情報を抽出し、かつそれらに基づいてプログラマの意図を推定するといったようなことである。また、認知科学的プログラム理解とは、人がプログラムを理解するようにして、あるいは、人のプログラム理解のやり方を参考にして、プログラムをコンピュータに理解させることであると考えてよい。

最近になってプログラム理解に関するニーズが高くなっている。これまでは主にプログラミング教育のための知的CAIシステムの一環として研究されてきたが、CASEの出現によって既存の膨大なソフトウエア資産を有効に活用してCASE環境の中に取り込むための技術としてReverseEngineering(以下REと略す)の概念がソフトウエアに適用されるようになり、そのためにプログラム理解技術が注目されるようになった。両者の間には類似点はあるが、当然相違点もある。これらの違いを明確にすることはこれからのこの分野の技術発展にとって重要なことである。

我々は過去約１０年にわたって教育向きの知的プログラミング環境INTELLITUTORを開発し、そのサブシステムとしてプログラム理解システムALPUSを開発してきた[1-7]。最近、ALPUSで利用されているプログラム理解技術をREに応用する研究に着手した。また、知的CAIシステムのためのプログラム理解の研究も以前に比べるとかなり活発になってきた。このような状況を前提として、この機会に、認知科学的プログラム理解の考え方と方法について若干の考察を行ってみたいというのが、この報告の趣旨である。

2　プログラム理解研究の背景

プログラム理解の研究は８０年前後から行われるようになったが、８０年代はそれほど活発ではなかった。この種の研究が活発になったのは９０年前後からである。活性化した理由は、研究成果が目に触れるようになって興味を持つ若手の研究者が増えたこと、周囲の環境がこの種の研究を必要とするようになったことなどであろう。特にCASEが与えたインパクトは大きなものがある。つまり、CASEと呼ばれる統合型プログラム開発環境の普及にともなってすでに存在するソフトウエア資産のCASE環境への取り込みの問題がクローズアップされ、その実現技術としてプログラム理解の研究が注目されるようになったわけである。これは前述のようにReverseEngineering(RE)と呼ばれるようになり、ソフトウエア産業において極めて熱い期待がもたれており、９０年代に入って活発に研究が行われるようになった。類似の研究として、オブジェクトコードを記号語表現に変換したり、さらにはフローチャートを生成するといったようなことが７０年代から行われていたが、CASEはより抽象度の高い記述情報を必要としているので、プログラムからの意味の理解が注目されるようになったしだいである。

しかし、同じプログラム理解といっても、プログラミング教育を対象としたものとREを対象とするものとは基本的な違いがある。これについては次章で論じるが、要約すると、初心者によって書かれた不完全で稚拙なプログラムから論理的バグとその原因や意図を同定するための理解と、専門家によって書かれた正しい業務処理プログラムから仕様情報を抽出するための理解の違いである。更に、プログラミング教育を対象としたプログラム理解においても、より精密に、より深く理解しようとする試みが行われつつある。このためには、人（プログラマ）のプログラム作成過程に踏み込んだモデル化が必要とされ、認知科学的な側面の研究がこれまでにもまして重要となってきた。つまり、人がプログラムを理解しようとするときはプログラム作成者のプログラミング時における意図を思い描くことが必要となるが、コンピュータにプログラムを理解させるときも当然この機構が要求される。バグを含む稚拙なプログラムにおいては特にそうである。

3　プログラミング教育とリバースエンジニアリングにおける理解の違い

先ず、プログラミング教育においては、事前に行われるアルゴリズムやプログラミング技法等に関する教

育の効果を評価すること、プログラム中のバグの検出と訂正の助言を行うこと、更にはこれらに基づいて個別に適切な指導を行うこと、等のためにプログラム理解が必要となる。したがって、対象となるプログラムはバグを含む不完全なものであり、バグの種類、プログラマの意図、バグの生じた原因等を推定し、それをプログラマ（学生）に説明し、訂正を助言し、正しい知識を教授することが必要となる。教授の部分が知的CAIシステムであり、適切な個別指導のためにユーザモデルの生成が必要となる。また、アルゴリズムは、一般に典型的なものが選ばれ、理解において問題分野の専門知識は特に必要ではない。

また、教育の対象となる学生あるいは生徒のレベルによって、理解の内容やアプローチが異なってくる。すなわち、高校１年生にプログラムの初歩を教育する場合には、プログラミング言語と基本的なデータ処理の概念の習得が中心となるので、練習問題は入力データ列の中の特定の値を持つデータのカウントや平均値の計算程度のものであり、特にアルゴリズムという意識は必要ではない。このようなプログラムに対しては、「プラン(plan)と呼ばれるプログラミング技法の構成単位を組み合わせてプログラムが構成される」という考え方がSoloway等によって提案され、PROUSTという実験システムを通してうまく働くことが実証された[8]。このアプローチは初心者プログラムの理解の基本的枠組みとして広く用いられている。これとは対照的に、我々のプログラム理解システムALPUSは専門過程の大学生が作成するプログラムを対象としており、アルゴリズムが中心的な役割を担っている。専門課程のプログラミング教育では、Quicksortのような典型的なデータ処理アルゴリズムを解説し、プログラミング演習を通してプログラミング・スキルを獲得させるという手法が採られているからである。更に、意味の理解と意図の理解を明確に区別する必要があることが、教育向きのプログラム理解の特徴といえる。つまり、バグの検出に加えて、発生の原因を推定するためにプログラマが意図したことは何であるかを推理することが必要となるからである。これについては別に論ずる。

一方、REにおいては、プログラムは専門家によって書かれた正しいものであり、特定の業務を処理するための一連のプログラム群の一つである。したがって、バグの検出は必要ではない。その代わりに、仕様情報の抽出が必要となる。そのためには、対象となる業務との関係づけを行い、処理プロセスのどの部分がどんな処理を担っているかを明らかにすることが求められる。これを可能とするためには、ある程度の仕様情報が必要となる。つまり、与えられた仕様情報とプログラムから、詳細な仕様記述と処理手続きを生成することが、REにおけるプログラム理解の基本機能であると考えられる。すなわち、処理手続きの図的表現への変換、仕様記述言語による定義的表現の生成や、日本語による説明文の生成等が必要となる。このような情報はCASE環境の中に取り込められて再利用されることになる。また、一般に業務処理プログラムはエラー処理等の例外処理を含んでおり、この部分はプログラムの基本仕様ではないことが多く、保守性をを疎外している。したがって、例外処理の部分を読み飛ばして、重要な部分だけをまとめる機能が要求される。

なお、教育用のプログラム理解の対象言語は、PASCALやLISPが一般的であり、最近Cも取り扱われるようになりつつある。これに対して、REにおいては圧倒的にCOBOLが多い。これは、従来の業務処理プログラムが主にCOBOLで書かれていた事実から、当然である。今後はREでもCが取り扱われるようになるであろう。対象言語の選択にはこれとは別の理由もある。それは、この分野の研究者の趣向や親近感の問題である。プログラム理解は知識処理つまりAIと関係が深いので、AI向きプログラミング言語であるLISPが選択されることは、研究の本来の意義を考えるとはなはだ疑問を感ずるが、研究者にとっては取り扱いになれていて、コンパクトな問題設定ができ、かつ論文として投稿したとき査読者にも分かりやすい、等の利点があることなどから必然性が高い。しかし、LISPはCOBOLとあまりにも異なるので、取り扱われる例題も異なり、したがって研究成果の応用はあまり期待できない。CやPASCALも研究者にとって比較的取り扱いやすく査読者にもなじみがあるという点で選択されるわけであるが、こちらの方は実用への応用という観点からはLISPよりもはるかに意義があると言える。なお、PrologにもLISPと似たような側面がある。

4　認知科学的プログラム理解について

広辞苑によると、理解とは、1）物事の道理をさとりしること、2）物事の意味をのみこむこと、3）物事がわかること、とある。この説明で明らかなように、理解の主体は人間であるということである。また、認知科学(CognitiveScience)は人の知能の原理やメカニズムの究明を目的とする研究分野であり、心理学の一分野である。したがって、「認知科学的プログラム理解とは、人がプログラムを理解するようにしてコンピュータにプログラムを理解させる」ことであるといえる。しかし、このことを忠実に実現することは事実上出来ないので、その代わりに、「人のプログラム理解のやり方を参考にしてコンピュータによるプログラム理解システムを実現する」試みが行われている。プログラム理解は必然的に認知科学的であるわけであるが、ことさらに「認知科学的」という修飾詞を使うのは、人の理解のメカニズムにとらわれないが効果としてプログラム理解を行うというアプローチも採られているからである。

また、認知科学はAIの基礎でもある。AI研究が認知科学研究とことなるところは、ソフトウエアとして問題解決システムを作成して能力を検証するところまでを行うという点である。具体的には、知識ベースと推論機構から構成される、知識ベースシステムとして実現される。これは、人による問題解決が、対象問題に関する専門知識を利用して行われるという事実に基づいている。一般のプログラムが、知識とそれを利用する推論とがアルゴリズムとして融合されているのに対して、知識ベースシステムでは、知識と推論とを明確に分離しているところに、最大の違いがある。分離することによって知識の探索という非効率な処理が必要となるが、汎用性と柔軟性が飛躍的に高まるという利点が得られる。したがって、認知科学的プログラム理解（以後、わずらわしいのでプログラム理解と呼ぶ）システムは、知識ベースシステムとして実現されることが自然である。問題分野の専門知識を利用するという点で、エキスパートシステムでもある。

さて、プログラム理解とは与えられたプログラムから意味(semantics,meaning)を汲み取る作業である。ここでいうプログラムの意味は、いわゆるプログラム言語の意味論ではなく、プログラム言語を通してプログラマが表現している内容をいう。これについては別に議論しているので省略するが、プログラム言語の意味を最下位レベルの意味とし、その上位に、基本データ捜査レベル、データ処理技法レベル、アルゴリズムレベル、および問題解決概念レベルという、合わせて５段階の意味レベルがある[1]。

例えばデータ処理技法とは、カウント技法、捜査技法、交換技法等であり、先に述べたプランがこれに相当する。経験を持つプログラマは各種のデータ処理技法を知識をして持っており、プログラミングにおいてこれらの中から適切なものを選択して利用すると考えられる。これらはプログラムの中で（したがってデータ処理において）それぞれ特定の役割を果たすわけである。データ処理技法レベルの意味理解とは、プログラムの中で用いられている技法とその役割を特定する作業である。また、アルゴリズムレベルの意味理解とは、プログラムからアルゴリズムを抽出する作業である。なお、プログラムを設計し書く時に用いられるプログラミング知識は、これらの意味と対応関係を持っており、プログラム理解とは、プログラムの中で利用されているプログラミング知識を特定する作業であるということもできる[1]。意味を理解するには、予めコンピュータの中で意味要素をデータ構造として表現し、テンプレート化しておく必要がある。たとえば、データ処理技法はプランと呼ばれる方法で、アルゴリズムはHPGグラフと呼ぶ方法でそれぞれテンプレート化することができる。理解は、次章で述べるように、テンプレート・マッチングによって実現できる。

さて、認知科学的プログラム理解とは人によるプログラム理解のやり方を参考とするコンピュータによるプログラム理解処理であると述べたが、未だ決定版と言えるような人の理解プロセスのモデルは得られていない。これまでは、各研究者が色々と模索しながら、実現可能な方法を用いて理解システムを設計していると考えてよい。プログラミング教育を対象とするプログラム理解では、バグのあるプログラム、もしくは稚拙な技法が用いられているプログラムから、バグを検出して訂正を助言したり適切な知識を教授する必要があるので、プログラムそのものの意味理解に加えて、プログラマがどのような意図(intention)でプログラムを書いているかを推定する意図理解が不可欠である。

5　プログラム理解の方法について

プログラム理解システムの構成や具体的な理解の方法については、参考文献に挙げたPROUSTやALPUS等に関する報告を参照して頂くとして、ここでは考え方や課題について議論することにする。また、プログラミング教育を対象とするプログラム理解について議論する。

人（先生あるいはチュータ）は学生の（不完全な）プログラムを理解するとき、自分が持っているプログラミング知識を使って頭の中にイメージを生成するものと考えられる。このイメージは、与えられたプログラムの中に記述されている情報を手がかりに構築され、意味的に納得できるものが生成できたとき、つまりつじつまが合うような意味表現モデル（データ構造）が生成できたとき、理解できたと感じるはずである。この仕組は、自然言語理解や画像理解と何ら変わらない。このことは、歴史の長い自然言語理解や画像理解の研究成果を活用するか、少なくとも参考にすることが出来ることを示唆している。何が容易で何が困難か、どんな方法がどこに使えるかなど参考にしない手はない。しかし、そちらの領域も極めて問題が複雑であり、未だ未解決の課題が山積していることも事実である。また、勘違いの理解、つまり誤解もしくは”理解したつもり”も当然起こる。つじつまが合うモデル表現がユニークに決まらないことに起因している。

自然言語理解にしろ画像理解にしろ、基本的かつ代表的な方法は意味的パターンマッチング法(semantic patternmatchingmethod)である。これは、意味要素を予め準備しておき、与えられた状況にうまく適合するように、要素を選んで組み合わせることによって複雑な意味を表現することを行う方法である。意味ネットワークと呼ばれるデータ構造でモデル化し表現される。各構成要素に対し、それらがどんな条件で存在できるかとか、他の要素と結合するにはどんな条件を満たさなければならないかというような制約条件(constraints)を付加しておけば、探索とモデル構築が可能になる。つまり、制約を満たすような組み合わせが全てのデータを説明できるように生成できれば、理解が成功したことになる。これを効率良く行うには、探索空間を限定する必要があり、そのために記憶の構造化が行われることになる。なお、プログラム理解には５つのレベルがあると述べたが、各レベルの意味構成要素を分析・収集し、体系化し、知識ベース化することが必要となる。さもなければ、極めて限定されたプログラムしか取り扱うことが出来ない。このような知識の収集は未だ行われておらず、共同利用型の知識ベースとしての構築技術も未だ開発されていない。このような知識ベースはプログラミング支援、プログラム生成や再利用、したがってCASEにも有用なはずであるので、この問題に積極的に取り組む研究者や機関の現われることを期待する。

さて、バグの検出と理解は次のような考え方で説明できる。与えられたプログラムから正しい知識だけではうまく説明モデルが構築できないとき、バグが存在するものと仮定できる。この場合は、対応する要素に関連のあるバグ知識で置き換えを試みて見て、もしこれが成功したらバグの特定が出来たと考えることが出来る。これは、知的CAIの分野で学生モデルの表現法としてよく知られているバギーモデルである。バグ知識は、予め分析収集しておき、対応する知識（意味要素）に付加しておくことによって、比較的容易に管理でき、かつ探索し、利用できる。この方法の欠点は、バグ知識の網羅性を実現することが困難であり、したがって、汎用化に限界があることである。事例ベース推論の機構を応用することによって、知識ベースの(半)自動管理はある程度可能であろう。

もっと原理的に優れた理解機構を実現するには、学生によるプログラム作成過程のモデル化が必要である。もし、プログラム作成における思考作業が幾つかの基本的な知的作業の(generictasks)組み合わせとしてモデル化可能ならば、理解はより柔軟で説得力に富んだものとなるはずである。バグの発生もこのモデルで説明できるはずである。大槻らの摂動モデルは類似の考え方であるが、一般に、設計問題に対する説得力のある認知過程のモデル化は未だ進んでいないようである。この研究は、プログラミング教育にとっても極めて価値がある。現在の教育法は、前にも述べたように、演習によってプログラミング・スキルを自然に獲得させるというやり方であるが、個人差があることもあり、かならずしも教育効果が上がっていない。認知過程のモデルは、学生の学習状態の理解に役立ち、適切な教育指導を可能とするであろうと考えられる。

なお、ここで述べたような情報を獲得するには認知科学実験が不可欠である。単に考察するだけでは精密な情報を得ることはできない。これは、工学や理学において実験が不可欠であることと同じ理由である。

6　おわりに－産学協力のすすめ

プログラム理解は、認知科学、人工知能、知識ベース、知的CAI、プログラミング教育、ソフトウエア工学にわたる広範な研究領域であり、様々な背景の専門家の協力を必要とする。また、単に論文にするための研究なら適当な例題を選んで新しいアプローチと技術を主張し提案するだけですむが、実用的な技術として育てようという立場にたって考えると、実際の応用分野の中から意味のある問題を選んで評価を行うことが不可欠である。簡単な例題でうまくいったとしてもそれは研究者の我田引水にすぎない。特に、リバースエンジニアリングを対象とするプログラム理解の研究においては、業務プログラムの開発現場である企業と大学との産学協力研究が不可欠である。特に、プログラミング知識の収集、分析、体系化、および共同利用型知識ベースとしての構築と運用は社会資本としても極めて重要なものとなるであろう。

企業ではソフトウエア開発の重要なノウハウのリークを嫌い企業内研究所で研究開発をしたいのかもしれないし、日本の大学の技術開発力に期待できないから企業内研究を行うという選択を行っているのかもしれないが、ソフトウエア工学における独創的研究成果を出すには産学協力が必須であると思う。日本の大学からこの分野の新しい技術の提案が殆ど無いのは、現場が分からないために技術開発の目標が見えないからであると思う。適切な目標が設定できそれに向かって努力すれば、独創的な成果は自然の帰結であると考える。現在の大学の研究環境では、現実の問題を分析し将来の目標を設定しその実現に向かって研究開発を行うことは、荷が重いかもしれないが、避けて通れない道であることは疑いが無い。

これを実現するには、研究開発における縦割り行政の壁を取り払うことである。特に、通産省の管轄下である研究開発資金は大学から見ると２－３桁は額が異なる。アドバイザーとして協力するという形だけの共同研究ではなく、研究費の支援と成果の要求および評価をともなう産学共同研究の為の制度作りは緊急の課題であると思う。さもなければ、米国を中心とする欧米諸国から提案される新しい概念と技術に振り回され続けるという寂しい状況から、決して卒業は出来ないであろう。

参考文献

1）上野晴樹、知的プログラミング環境ープログラム理解を中心にー、情報処理、vol.28, no.1, pp.1280-1296, 1987

2）上野晴樹、PASCALプログラムを教える、コンピュータ科学、vol.2, no.5, pp.335-360, 1992

3）上野晴樹、プログラムの意味と理解、知的プログラミング（大須賀、上野、雨宮、奥野、原田共著、オーム社）、pp.151-187, 1993

4）Ueno,H., INTELLITUTOR: A Knowledge Based Programming Environment for Novice Programmers, Proc. COMPCON89 Spring, pp.390-395, 1989

5）Ueno,H., ALPUS: A Program Understanding System by Means of Algorithm-Based Programming Knowledge, Proc. PRICAI90, pp.693-698, 1990

6）Ueno,H., An Integrated Knowledge-Based Programming Environment for Novice Programmers, Proc. IEEE COMPSAC91, pp.124-129, 1991

7）Ueno,H., Integrated Intelligent Programming Environment for Learning Programming, IEICE Trans. on Information and Systems, vol.E77-D, no.1, 1994 (to appear)

8）Jonson, W.E. and Soloway, E., PROUST: Knowledge-Based Program Understanding, IEEE Trans. on Software Engineering, vol.SE-11, no.3, pp. 11-19, 1985