ソフトウェア工学の新しい潮流

―― 徳田研究室～情報工学科 ――

コンピュータのソフトウェアは、通常、比較的人間が理解し易い高級言語によって記述され、実行に先立ち、機械語プログラムに翻訳されて動作する。この翻訳をするためのソフトウェアはコンパイラと呼ばれており、コンピュータの発展とともにソフトウェアの中心課題として盛んに研究されてきた。そしてその中で字句解析や構文解析、データ構造などに関する様々な言語処理技術が確立されてきた。

今日の情報処理技術、コンピュータ技術は急速に発展している。そしてインタープリタ*、コンパイラ*などソフトウェアの開発技術は研究され尽くしたわけではなく、新しい言語形態やハードウェアの開発にともなって、ソフトウェア技術が開発され続けてきている。

徳田研究室では、この日進月歩のソフトウェア工学を研究されている。今回の研究室訪問では徳田助教授の研究テーマのうち、言語処理系、ソフトウェア生成系、ソフトウェアの開発環境についてお話を伺った。

徳田　雄洋 助教授

コンパイラ：ソースプログラムを一括して機械語コードに変換するソフトウェア。

インタプリタ：ソースプログラムを一命令ごとに解釈・実行するソフトウェア。

コンパイラと言語処理系

言語処理系には、コンパイラ、インタプリタなどがある。以前はこれらを一つ一つ手作業で作成してきたが、開発過程がだんだん整理されてくるに連れて、コンパイラの生成系というものが可能となってきた。

しかし、コンパイラの自動生成や半自動生成を考える場合、どのように構文規則や意味規則を与えるかという問題がある。構文規則とは、記号列の正しい並びを記述する規則であり、意味規則とは構文規則にしたがって記述された記号列がどのような意味を持つのか、プログラムで言えば、それによってコンピュータに行なわせる処理の対応を示した規則である。

通常、コンパイラの生成系などでは構文規則を一種の書換え規則として書いている。そして構文規則をもとにして、プログラムの構文解析が行なわれるのである。構文解析とは与えられた記号列がどのような構文構造を持っているかを決定する作業で、基本的には構文構造を決定して、対応する意味の動作を起こすために行うものである。構文解析によって得られる文法構造は解析木（図１）と呼ばれている。

ところが、ある種の構文規則には対応する動作がない場合がある。この時には構文の認識をせずに、通過（バイパス）出来ないかと言う問題が生じる。そこで徳田先生は、バイパス型の構文解析、及びその方法について研究をされた。

構文解析を大別すると、下降型解析と上昇型解析がある。前者では解析木を根（文法の出発記号）から葉（字句）へと生成して行き、後者では逆に葉から根へと作成していく。

上昇型解析の代表的なものにD.クヌスによって提案されたＬＲ(k)解析がある。これは記号列を左から右へ一度だけ見て解析する方法としては、最も広い範囲の文法を解析で

文法規則 G:={E,T,F,I}
E:=(E+T) or T
T:=(T*F) or F
F:=(E) OR I
I:=a,b,c,…

以上のような文法規則による
a*(b+c)
の解析木は、以下のようになる。

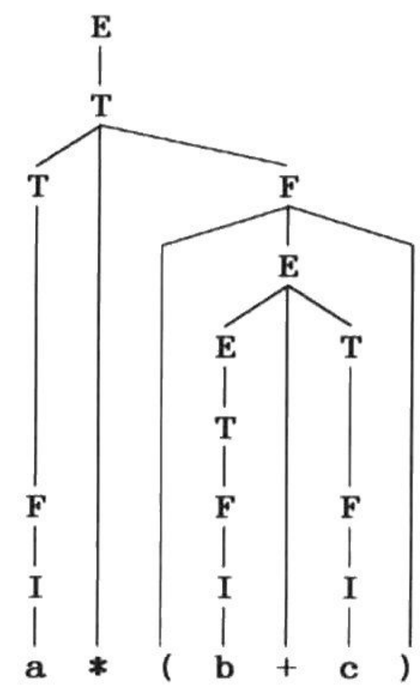

図１　解析木

きるという特徴がある。人間が文章を読むときには、日本語でも英語でも左から右へ向かって読むが、その時、分からない部分があってもそこですぐ左に戻らずに、普通はそのままいくつか字句を先読みして理解をしていく。このように、人間が構文解析するときの比較的自然なモデルが、ＬＲ(k)構文解析法では使われている。

ところが、このＬＲ(k)構文解析を通常のプログラミング言語に対して適用しようとすると、解析した状態数が数千という単位に及び、一般には実用的ではない。そのため、できた構文解析の表を小型化するアルゴリズムが必要となるが、これを用いた場合、バイパス型の構文解析を行うことは非常に難しい。従来の方法では、適用できる文法規則に制限が加えられてしまう。具体的には、単位規則を１対１対応として扱わなければならない。これでは、複雑な分岐をもつ構造を記述することができなくなってしまう。また、単位規則の形の制限をなくすようにすると、今度は正しさが壊れてしまう。徳田先生は、この両方を解決するアルゴリズムを考え出した。

これは、バイパス型ＬＲ(k)構文解析法と呼ばれている。徳田先生はバイパス型構文解析を、解析可能な文法の範囲を落とさずに小型化するアルゴリズムを適用できる形で、完成させるのに成功した。

意味記述の２つの方法

またコンパイラの自動生成の時には、意味をどのように記述するかという問題がある。これには属性文法という書き方と動作ルーチンという書き方がある。前者は文法規則の持っている文法記号に属性をもたせ、属性の間の値の依存関係を定義する方法である。この方法では文法規則を認識する順番を意識せずに依存性だけを書くために、プログラムの記述は楽になるが、実行速度は能率的ではない。後者の動作ルーチンとは、具体的な文法規則を認識する順番を踏まえて、実際に起こす意味動作をプログラムで書いてしまう方法である。これを用いると、実行速度は早くなるのだが、プログラムを書くときに文法規則を認識する順番を考えなければならないので、記述は大変になる。

そこで徳田先生は、属性文法のある種のクラスのものをなるべく効率のよい動作ルーチンに変換する、一連の方法を開発された。

ソフトウェアの生成系

なるべく少ない労力で、正しく動くソフトウェアを効率的に作るために、ソフトウェアの生成系というものがある。字句解析系や構文解析系などを自動生成して効率を上げられるようになったのも、その一例である。

これをさらに発展させて行くための一つの方法として、徳田先生は構造エディタの生成系というものを研究されている。

通常、プログラムを組む時は、テキストエディタ（入力された文字をそのまま編集するエディタ）を使うが、テキストエディタではいま作成しているものが、論文なのか手紙なのか、あるいはどういう言語のプログラムなのか把握していない。これに対し構造エディタは、テキストエディタのような汎用性はないが、一度作成すると構文的なチェックや意味的なチェックなど、非常に知的な判断を提供してくれるものになる。この構造エディタを、構文規則と意味規則の記述から用途に応じて自動的に作成するために、生成系もいろいろと研究されてきた。

構造エディタは内部形式に木構造を持っていて、人間が理解し易い２次元的な形で表示する。この構造エディタそのものは、方式などがほとんど決まっている。そこで徳田研究室では２次元的なグラフ構造や、そのグラフに基づく図形にこれを適用

することを考え、グラフ構造エディタの生成系というものに取り組んでいる。(図２～４)このエディタは、構文規則、意味規則、表示規則から成立ち、規則を持った図形（例えば地下鉄の路線図など）を扱うことができる。もちろん、グラフィックエディタのように一枚の絵としてではなく、その構造や意味も把握することが出来る。

この生成系が最終的にできると、電気回路やプログラムに基づく図形の上で、ある種の意味計算ができるようなプログラムが、比較的効率的に生成できるようになると思われる。

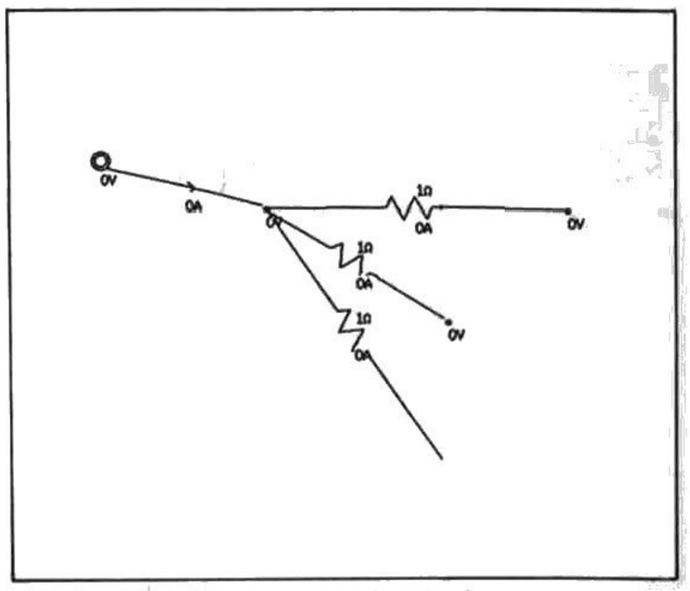

図２　編集途中

ソフトウェア開発環境

データベースというと、いままでおもに銀行の口座や飛行機の座席枠の管理などにおいて、いろいろな方法が発達してきた。これらは事務用のものについては完成されてきたが、ソフトウェアの開発過程を助けるデータベースにこれを適用しようとすると、必ずしもうまくは行かない。その理由は、事務用のデータベースは個々の量が非常に小さい情報を大量に高速に処理するというタイプであるのに対し、ソフトウェア開発用のものでは、一つの作業がかなり長い時間に渡って行われていて、一人が扱うデータ量も大きくなり、データが複雑な構造を持つことになるからである。

徳田研究室では、ソフトウェア開発用のデータベースの作り方について現在模索中だそうである。その一つの考え方として、オブジェクト指向データベースというものを考えられている。

オブジェクト指向言語は、今までのプログラム言語が動詞の記述により構成されていたのと異なり、名詞と名詞に送るメッセージによって記述されている。この、名詞によって示されるものをオブジェクトといい、各オブジェクトは外部から直接見ることが出来ず、全ての情報の受け渡しはメッセージを通じて行うことになる。各オブジェクトは自分自身に関する独自の情報を持っていてこれを管理し、また与えられたメッセージに対応して必要な機能を実行することが出来る。このため、１つのオブジェクトに複雑な構造を記録させておいて、これをデータベース化して管理することで、強力なソフトウェア開発環境が実現されると思われる。ここで難しいのは、どのレベルで１つのオブジェクトを認識するかということである。現在のいくつかのオブジェクト指向言語のシステムでは、一部オペレーティングシステムの肩代わりをすることにより、ソフトウェア開発効率を高めている。オブジェクト指向データベースが完成されれば、よりよい開発環境が提供されることになるだろう。

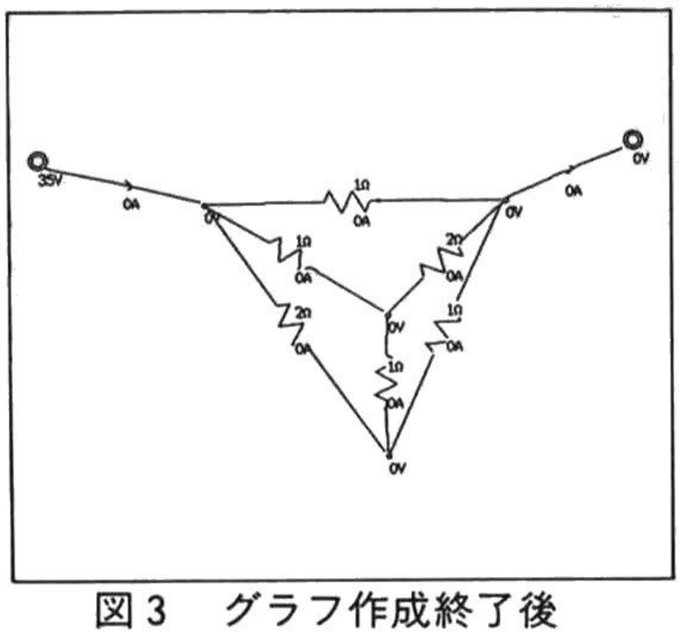

図３　グラフ作成終了後

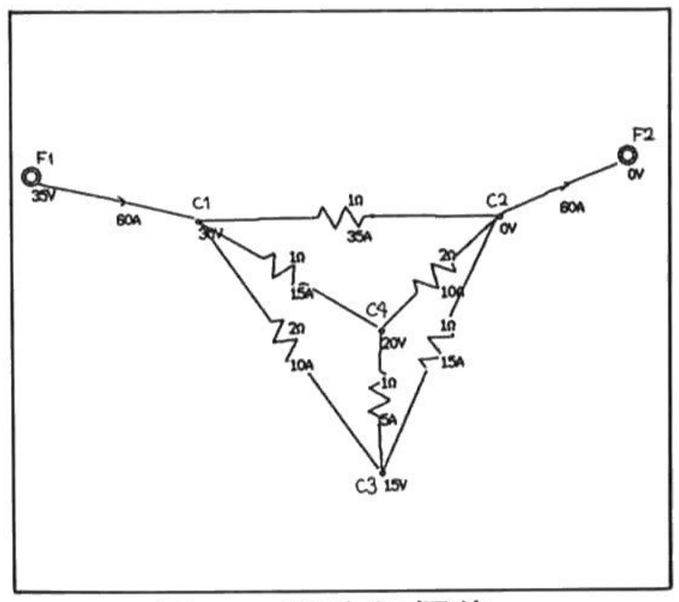

図４　意味評価後

徳田助教授はお忙しい中、こちらの不手際にも関わらず、ていねいに質問に答えて下さいました。感謝とともにこれからの御活躍に期待しています。

（内藤）