データフロー解析に基づくプログラム保守支援

3K－2

四野見 秀明　藤井 邦和　高橋 真由美
日本アイ・ビー・エム株式会社 アジア・パシフィック・プロダクツ

1 はじめに

筆者らは、PL/I、COBOL対象の保守支援ツールRE/Cycleの研究開発を行って来た[1][2]。以前の報告後、対象解析範囲を1コンパイル単位内から、プログラム群へと拡張を行い、外部ルーチン呼び出しを経由する他プログラムへの影響波及解析をも可能にし、名前も'悟' Satoru (Source code Analysis TOol for pRogram Understanding) とした。本稿では、'悟'の機能について、保守支援におけるデータフロー解析結果の有用性という観点で報告する。技術的には、プログラム・スライシング[3]の実用レベルでの応用である。これらの機能は、西暦2000年対応のプログラム修正時にも強力な支援となる。

2 保守時のプログラム理解

プログラム、特に第三者の書いたプログラムを保守する場合、まずその内容を正確に理解しなければならない。ところが、現実の保守現場では、対象となるプログラムに関する仕様書が無くなっていたり、元々存在していなかったり、あってもプログラムに対する長年の修正の繰り返しを仕様書に反映することを怠ったため、仕様書に書いてある内容と実際のプログラムの実装が異なっていることも多い。その時、担当者が唯一頼りになるプログラムを詳細に読んで、その内容を理解しなければならない。

我々はRE/Cycle[1]をプロトタイプとして、社内外の保守現場との共同研究を行ってきた。それを通して、既存のPL/I、COBOLのプログラムの保守を困難にしているのは、GOTOの多用やルーチンの呼び出し関係の複雑さ等の制御の追跡の難しさもあるが、データがどのように利用され、流れていくかを追跡することの難しさが大きいことが明らかになった。

保守時の多くの部分を占めている作業は、ある部分を修正する場合の影響箇所を発見し、作業の見積りをしたり、実際の修正作業をすることである。プログラム中の修正による影響が、データの流れに沿って代入文やサブルーチン呼び出しにより、プログラム内の他の箇所もしくは、他のプログラムへと波及していくのを見落としなく正確に把握する必要がある。また、プログラムを理解しようとする場合、注目する入力データが、プログラムの中でどのように受け渡されているかを追跡して行く必要がある。データの流れを正確に追跡することが、正確なプログラム理解へと繋がるのである。

しかし、PL/I、COBOLの保守現場でのプログラムは、データの受渡しが明示的な代入であることは少ない。以下のようなデータの階層関係やメモリー上での配置を考慮しながら把握する必要がある:

- 同じメモリー領域を別のデータ構造として共用する複雑な再定義(COBOLのREDEFINES、PL/IのDEFINED、BASEDの利用)
- データ構造単位での代入や、COBOLのCORRESPONDING、PL/IのBY NAMEによる代入
- コンパイラが行うデータの境界合わせ

'悟'では、以上の点をツールが考慮し見落しの無い正確な影響波及分析、プログラム理解を実現する。

3 データ依存関係解析

'悟'の解析は、プログラム・スライシング技術[3]の静的スライスにおけるデータ依存関係 (Data Dependence)の利用に相当する。プログラム・スライスには、プログラムを静的に解析して得られる静的スライス(Static Slice)と、ある入力に基づいてプログラムを実行したパスに関して得られる動的スライス(Dynamic Slice)がある。動的スライスは実行時に動的に決定されることに関する解析も可能な反面、プログラム中に実行したパス上のことしか解析できない。静的スライスは、(入力データにより、ポインタが指す位置が移動するなど)実行時にしか決定されないことに関しては解析できないが、実行されなかったパスをも含む全てのパスに関してプログラムを解析できるという利点がある。プログラムのある箇所を修正した場合の影響波及分析という観点では、静的スライスが適している。ちなみに、COBOLではポインタは使用されることは無く、PL/I

Program Maintenance Support Based on Data Flow Analysis
Hideaki Shinomi, Kunikazu Fujii,
Mayumi Takahashi
Yamato Laboratory, IBM Japan, Ltd.

でも殆どの場合、ポインタを動的に移動することは無く、主に同じメモリー領域を異なる名前で共有するために(DEFINEDの代わりに)使用される。

悟'では、プログラムの構文解析後に、ステートメント間の制御を表現するコントロール・フロー・グラフを内部的に生成し、その上でステートメント上の変数間のデータ依存関係を生成する。その依存関係を利用して次節で述べる機能を実現する。

4 データフロー解析による保守支援

ここでは、'悟'の機能の中で、データ依存関係を辿ることにより実現している機能について述べる。

4.1 プロセス間のデータフロー表示

'悟'では、プログラム間の呼び出し関係や、プログラム内での内部ルーチンの呼び出し関係を表す図を自動生成する(図1)。その呼び出し関係を表現する矢印の横に♂の印としてプロセス間を流れるデータを表現する(データカップル)。呼び出し関係の矢印を選択することにより、呼び出しにより渡される/返される変数名がリストされる。データ構造中の下位構造のみが渡されるときは、その修飾された下位構造名がリストされる。呼び出しによる入出力インターフェースが明確になる。リストされるのは、呼び出しパラメータにより渡されるものだけではなく、(EXTERNAL属性を含む)グローバル変数を経由して渡される/返されるデータも含む。

4.2 影響波及分析

データ依存関係を辿ることで、ステートメントや、データ構造中のフィールドの意味を変えた時の影響箇所を把握できる。'悟'では、ステートメント中の変数を選択し、その値が代入/参照さる箇所(代入/参照検索)、また、データの流れの追跡による、その値に影響を与える/が影響を与える全ての箇所(代入/参照連鎖検索)を画面上で表示できる。前者はデータの流れを追跡しながらのプログラム理解に有効であり、後者はプログラム修正時の影響波及の把握に有効である。既に述べたデータの再定義やデータ階層を考慮した正確で見落としのない解析を実現する。図1はファイルから読み込むレコードの1フィールドに関して意味的変更を行う場合の参照連鎖検索の結果である。影響波及するステートメント、内部ルーチン、プログラムが明示されている。

5 西暦2000年問題への適用

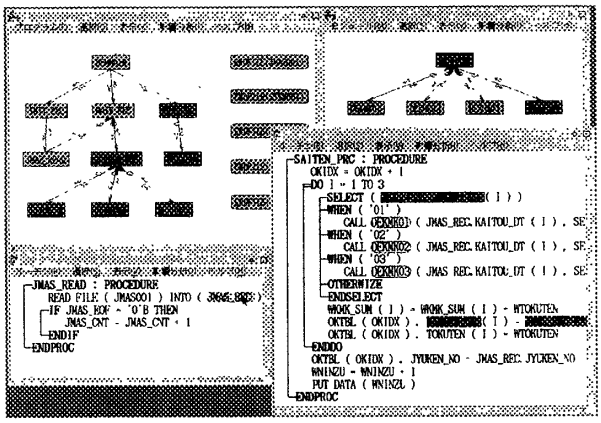

図1 影響波及分析結果表示（参照連鎖検索）

年数を表現する際に西暦の下2桁のみを使用しているプログラムが、西暦2000年を迎えて修正を余儀無くされている。'悟'は、その修正を見落としなく、効率良く行う手段を提供する。データカップルを参照することで、プログラム、ルーチンの呼び出しインターフェースの理解をすることが出来、仕様書のないプログラムに関しても、影響波及分析を起動すべき変数の決定を支援する。修正が必要とされる年数の変数から影響波及分析を起動することにより、影響が及ぶ箇所を正確かつ見落としなく発見することが可能である。

6 おわりに

'悟'におけるデータフロー解析の結果として表示される機能と保守におけるその意義を述べた。その他、ソースコード表示のアクション・ダイヤグラムや、データ構造のメモリー上での配置や、階層関係を視覚化したデータ構造図等の機能については別稿で述べる。また、'悟'の静的スライスの解析結果としてのデータ依存関係と制御依存関係の抽出機能は、ある会社で、システム再構築のための既存プログラム理解に実際に使用されている。

参考文献

[1] 四野見秀明, 藤井邦和, 牧野正士, 津田和幸, "構造化分析/設計の方法論に基づいたプログラム理解支援ツール", ソフトウェア工学研究会資料 SE-87-1, 情報処理学会, 1992.

[2] 四野見秀明, 藤井邦和, "プログラムの自動的な再構造化における限界とその解決法", 情報処理学会情報システム研究会 42-1, (1993).

[3] 下村 隆夫, "Program Slicing技術とテスト, デバッグ, 保守への応用", 情報処理, Vol.33, No.9, pp.1078-1086.