6L-1

コンパイラの中間表現を用いた実行系列取得システムの開発

松﨑　新司†　　太田　剛‡
静岡大学大学院情報学研究科†　静岡大学情報学部‡

1. 導入・目的

ソフトウェア開発の様々な場面で行われるプログラム解析は，静的なものと動的なものに分けられる．このうち，動的解析ではプログラムを実行して得られた情報をもとに解析を行う．そのため，プログラム実行を監視・記録する機能を実行環境に追加し，実行系列情報を取得する必要がある．

本研究では，次の条件で動的解析に必要な実行系列情報を取得するシステムの開発を目的とする．

I. 言語に依存しない手法で実行系列取得を実現することで，言語ごとに処理を変更・追加するコストを低くする．
II. 実行系列取得におけるユーザの手間や知識ができるだけ少なくなる手法で実現する．

2. 実行系列取得方法

実行系列情報取得については，Valgrind[1]などの仮想マシン上でバイナリコードを読み込み，動的に命令を追加する手法が盛んに行われている．この手法には，バイナリファイルやソースコードなどを一切書き換える必要がない利点がある．しかし，仮想マシンのオーバーヘッドにより，全体の実行時間が長くなる欠点がある．

他に，実行系列を出力できるように，コードを書き換える手法が挙げられる．この場合に書き換える対象は次の三つが考えられる．

- ソースコード
  各言語の構文に合わせて，個々にシステムを作る必要がある．あるいは，人が手作業で必要な位置に出力文を挿入することで行う．
- アセンブリコード
  アセンブリコードのターゲットマシンごとにシステムを作る必要がある．
- 中間表現
  複数言語で共通の中間表現を用いているシステムをベースにすれば，一度に複数言語に対応できる．また，複数のターゲットマシンに対応したコンパイラの中間表現を書き換えれば，複数のターゲットマシンに対応することができる．

本研究では，複数言語に対応したコンパイラである GCC[2]の中間表現を書き換えることで，言語に依存せず，オーバロードが少ないシステムを実現する．

2.1. GCC 中間表現

GCC version4.x では内部で三つの中間表現 GENERIC, GIMPLE, RTL [3]を利用し，言語依存性を排除したコード生成を行なっている．

本システムではこれらの中間表現のうち GENERIC の段階で，実行系列取得に必要なコードを追加する．GENERIC では言語依存性が排除されているため，言語によって処理を変える必要がない．

2.2. システムの概要

本システムの概要を図１に示す．現段階で C・Fortran・C++・Objective-C の４言語に対応しており，GCC 内で変換された GENERIC を解析し，実行系列情報出力対象ノードの前後に出力コードを追加する．コード追加を行うかどうかは，GCC のコンパイルオプションによって指定する．ユーザは，コンパイル時にコンパイルオプションを追加し，生成された実行ファイルを実行するだけで実行系列情報を取得する事ができ，追加の知識や手間を必要としない．

2.3. コード追加対象の GENERIC

本システムで実行系列情報出力用のコード追加を行う対象の GENERIC ノードを表１に示す．表中の定義変数は，代入文において値が代入される変数を表す．参照変数は式内で使用した変数を表す．オブジェクト番地は生成されたオブジェクト実体の番地，オブジェクト変数はオブジェクトのポインタ変数を表す．

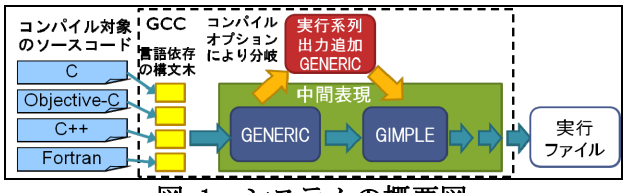

図 1　システムの概要図

A Multi-Lingual trace data gathering system by modifying Intermediate Representation of a Compiler.
† SHINJI MATSUZAKI, Graduate School of Informatics, Shizuoka University
‡ TSUYOSHI OHTA, Faculty of Informatics, Shizuoka University

表 1　実行系列情報出力対象 GENERIC ノード

ノードタイプ	抽出する要素	
代入文	左オペランド	右オペランド
	定義変数	参照変数，関数・メソッドの使用
関数呼び出し	呼び出し情報	引数オペランド
	関数番地	参照変数，関数・メソッドの使用
メソッド呼び出し	呼び出し情報	引数オペランド
	メソッド番地，オブジェクト番地，オブジェクト変数	参照変数，関数・メソッドの使用
インクリメント（デクリメント）	参照変数	
オブジェクト生成	オブジェクト番地，生成するクラス名	
オブジェクト削除	オブジェクト番地	
Return文	参照変数，関数・メソッドの使用	
条件分岐	条件判定結果，参照変数，関数・メソッドの使用	

3.　動的スライサによる評価

本システムが出力する実行系列を利用した，複数言語対応の動的スライサを作成することで，生成された実行系列が入力言語に依存しないことと，動的解析が実現できることを示す．作成したスライサでは本システムにより得られた実行系列情報と実行ファイルより得られるシンボルテーブル情報のみを用いて，動的依存グラフ(DDG)[4]または，動的オブジェクト指向依存グラフ(DODG)[5]に相当するグラフを作成する．スライサで作成された依存グラフの例を図２に示す．右側が依存グラフであり，左側に各ノードに対応するソースコードを示している．ノード間の実線の矢印はデータ依存，破線の矢印は制御依存が存在していることを表している．矢印のラベルはデータ依存の変数名である．　本スライサでは，このグラフをたどることで任意の変数のスライスを求めることができる．C・Fortran・C++・Objective-C の各言語で記述された同様のプログラムにおいて，同様の依存グラフが生成され，スライシング操作が行えることを確認した．

4.　おわりに

本稿では，GCC の中間表現を利用した，多言語対応の実行系列取得システムの実現方法と評価について述べた．本システムは，GCC のコンパイル過程で生成される中間表現に対して実行系列情報出力コードの追加を行うことで，ソースコードに手を加えずに複数言語への対応を実現している．また，ユーザはコンパイルオプションを追加してコンパイルし，実行するだけで動的解析の実行環境構築ができる．しかし，元のプログラムにノードが追加されるため，プログラムの実行時間が長くなる欠点がある．例えばノードが五つ追加されたプログラムの場合，ノード追加なしの場合と比べて，実行時間は約３０倍になる．仮想マシン上で動的に命令を追加する手法でトレースを取得している研究[6]では，実行時間が平均 11.89 倍となっており，本システムの方が実行時間が長くなっている．

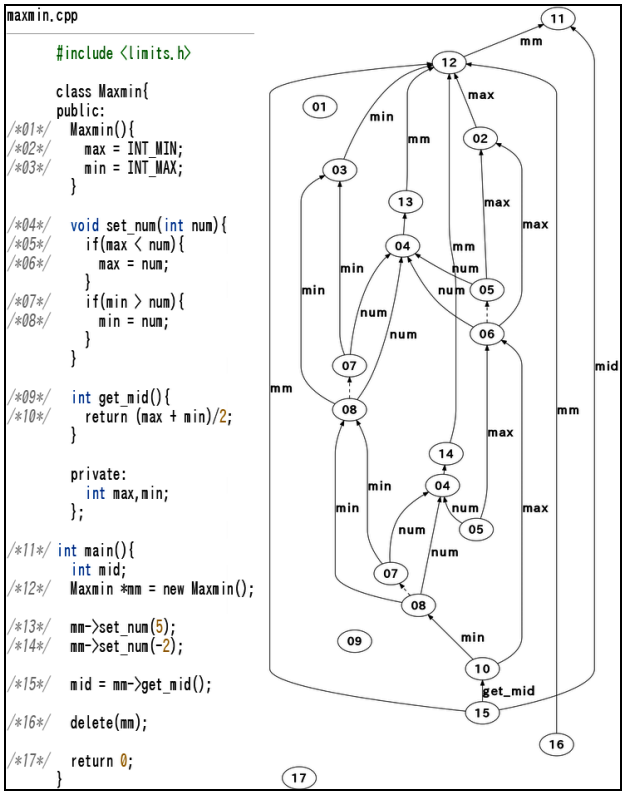

図 2　対象ソースコードと依存グラフ

今後は，実行時間の短縮機能を考案・実装することが求められる．例えば，注目する変数をユーザが指定できるようにすれば，ノード追加する範囲をその変数の値に関係する命令のみに減らすことができる．また，GCC が対応している他の言語についても，実行系列の取得ができるようにしていきたいと考えている．

参考文献

[1] Valgrind Home http://valgrind.org/
[2] GCC, the GNU Compiler Collection http://gcc.gnu.org/
[3] GNU Compiler Collection(GCC) Internals http://gcc.gnu.org/onlinedocs/gccint/
[4] H.Agrawal and J.Horgan, 1990, “Dynamic program slicing”, Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, pp.246-256
[5] Jianjun Zhao, 1998, “Dynamic Slicing of Object-Oriented Programs”, Technical report SE-98-119, Information Processing Society of Japan, pp.17-23
[6] S. Bhansali, W. Chen, S. De Jong, A. Edwards, and M. Drinic, 2006, “Framework for instruction-level tracing and analysis of programs” In Second International Conference on Virtual Execution Environments, pp.154-163