Sapidを使った動的解析ツールの実装

篠井 隆典

北陸先端科学技術大学院大学 情報科学研究科

2000年2月15日

キーワード： dynamic slicing, pointer analysis, Sapid, CASE tool platform.

1 背景

ソフトウエア開発保守において実行状態に関する情報は重要である。実行時に明らかになるメモリリークなどのバグのデバッグには、実行状態に関する情報は不可欠である。特に、ポインタの追跡に関しては静的な解析では困難であるとの指摘が多い。したがって、実行状態に関する情報が得られる動的解析もまた重要である。しかしその実装は、プログラムの規模が大きくなりやすく困難である。プログラムを実行するために、ソースコートの構文解析器、字句解析器をはじめとする様々な機能を実現する必要があるからである。しかも、これらの技術はすでに確立されており、その実装は本質的な問題ではないことが多い。それ故、これまでも様々な方法論が提案されているが、実装は多くない。

プログラム理解、デバッグ、テストなどのソフトウエア工学における問題に共通する解決法として、部分プログラムの抽出がある。部分プログラムの抽出法の代表的なものにスライシングがある。スライシングはスタティックスライシングと、ダイナミックスライシングに分けることができる。このうち、ダイナミックスライシングは、入力に制限を設けることにより、比較的簡単なアルゴリズムにもかかわらず、正確にスライシングをすることが可能という特徴がある。また、スタティックスライシングでは困難との指摘があるポインタ追跡においても、ダイナミックスライシングは有利である。実行パスが特定されるので、ポインタを追跡する範囲を著しく限定できるからである。しかし、動的な解析であるために実装が困難で、ある言語のサブセットや、特別に作られた簡易言語に対する実装が大部分である。

細粒度リポジトリとしてSapidが提案されている。これは、I-modelというモデルに基づいて作られたソフトウエアリポジトリである。I-modelに基づいたリポジトリは、これに基づいてソースコードを解析した結果を直接解釈実行することが可能な仮想機械を作

成できるなど、これまでのモジュールや関数単位のリポジトリに比べて、十分に細かい粒度を持っている。このことは、Sapidが動的解析に有効であることを示している。しかし、Sapidを動的解析に応用した例は多くない。さらに、SapidはCASEツールをのプラットホームになることを目的に設計されており、CASEツールを開発するときにしばしば必要になる構文解析器や字句解析器などの汎用的なモジュールを提供することで、開発の省力化やCASEツール間のデータ統合なども期待できる。

2 目的

以上の背景をふまえて、本研究では目的をSapidを用いてC言語のなるべくフルセットに対応できるダイナミックスライサーの実装を行うことに置く。そして、動的解析およびダイナミックスライシングを難しくしている要因を明確にし、それに対する回答を探る。また、その過程で、Sapidの有効性、特に動的解析に対する有効性を示すとともに、その問題点を探っていく。Sapidの細粒度という性質はは動的解析に対し有効なはずである。プログラムを実行するのに必要な情報は、構文に基づいてすべて保存されているからである。さらに、動的解析を行うためのAPIも用意されている。けれども、これを本格的に動的解析に応用した例はまだない。

3 実装

実装にはSapidに含まれる動的解析を目的としたSIP2というパッケージを用いた。これにはC言語の簡易インタプリタが含まれており、これを改造することで実装を行った。

実装ではポインタ解析を正確に行うために、1)malloc()の返り値以外のキャストは認めない,2)副作用のある演算は単独で使わなければならない,3)malloc()とfree() 以外のライブラリ関数とシステムコールの使用は認めない, という三つの制約を加えたC言語のサブセットを対象とした。

上記の制約のもとで、実行系列上のすべての実行時点で、すべてのポインタ変数に関するPoints-to集合を求め、Korelのダイナミックスライスのアルゴリズムにしたがってスライシングを行うことにした。

4 結論

ダイナミックスライシングにおいてもやはり、ポインタ解析は問題となる。C言語は、確保されていないメモリ領域もポインタを用いて制限なくアクセスできるなど、ポインタによる自由なメモリアクセスを許すので特に解析が難しい。また、システムや処理系に依存する部分が多くポータビリティーのある実装は困難である。その一方で、そのようなポインタの使い方や、処理系依存なコーディングは好ましくないとされており、システムプ

ログラミングでもない限りそのようなコーディングを必要としないことが多いのも事実である。

今回、実装に Sapid を用いた。Sapid を用いることにより実装の省力化が可能だったことは確認できた。今回改造した C 言語のインタプリタ sint は C 言語のプログラムで、その行数はコメント等すべてを入れて 104 行である。一方 Sapid を用いていない C 言語のインタプリタに CINT があるが、これのソースコードは主な部分だけで 85041 行ある。機能に大きな違いがあるので一概にはいえないがこの差は大きい。また、今回はデータ依存関係を追跡するルーチンだけ実装したが、ヘッダファイルもいれて 606 行である。このうちデータ依存関係を追跡しているのは 101 行で、制御依存関係、命令同一関係の追跡を行う部分も同様の規模で実装できると考えられる。したがって、Sapid は動的解析にも十分に応用がきくと考えられる。一方で、Sapid に関するドキュメントが十分に整備されておらず、習得に時間がかかったこと、sint のライブラリ関数の不整備などの実装が不完全な部分があることなど、有用なツールであるだけに悔やまれる部分も目についた。