実行系列をXMLで表現した動的プログラムスライサの実装と評価

武田 洋佑 (110073)

北陸先端科学技術大学院大学 情報科学研究科

2003 年 2 月 14 日

キーワード: 動的解析, データスキーマ, XML, 動的スライシング.

1 背景

動的スライシングは、工学的に有用な技術で現在も多くの研究が行なわれている。動的スライスは、デバッグやテスト、メンテナンス、プログラム理解に役立つ。最初の動的スライシングは、Korel と Laski が提案した。その後、Agrawal が、動的スライシングのいくつかのアプローチを提案した。また、手法の研究のほかに実行系列のサイズを軽減することを目的とした研究も多い。さらに、精度向上を目的とした研究として、プログラムツリーにマークする手法もある。動的スライシングをより実用的にすることを目指し、関数呼び出しやポインタ解析をサポートする手法もある。しかし、これらは全て開発効率に関する研究ではない。

動的スライシングの研究は、実行系列のサイズが大きくなり扱いが難しい問題、実行環境が必要だという問題、データ表現の難しさの問題がある。これにより、活発な研究が行われているにもかかわらず、我々の知る限り小さな言語を対象とする動的スライサしか存在しない。また、手法ごとに始めから実装すると、評価に至るまでに多大なコストがかかり、評価の精度も低くなる。そこで、動的スライサにデータスキーマを導入し、実行系列の抽出と解析を分離する。これにより、その手法の本質的な部分の開発に専念することができる。よって、各々の動的スライサの開発コスト削減を計ることが重要になる。

一般的に動的スライサには細粒度の動的情報を持ったデータスキーマが必要である。プログラム情報のモデル化として、ACML や Sapid の I-model、JavaML が存在する。JavaML は、粗粒度の表現形式であるため、スライシングに必要な情報が不足している。また、ACML と I-model は、細粒度であるが静的情報しか持たない。よって、動的情報を持ったデータスキーマが必要である。

2 目的

本研究は、実行系列に XML を導入し、データスキーマを定義し、開発効率を向上させることが目的である。XML を用いて実行系列を表現することで、実行系列抽出部分と依存解析部分が明確に分離できるため、抽出部と解析部の開発が同時に行える。これにより、手作業でのテストケース作成を行えるため、実行環境が無くても動的スライサの評価ができる。動的スライサの開発効率の向上は、動的スラシングに関する研究の促進に大きく貢献できる。

我々のゴールは、フルセットの ANSI C に対応した動的スライサを実現することである。第一歩として、ANSI C プログラムのサブセットを用いた動的スライサの実現を目指す。実現には、以下の 3 つのステップが必要である。

1. 実行系列のための適切なデータスキーマの設計
2. データスキーマに基づいた動的スライサの実装
3. 上記の実装に対し、データスキーマの妥当性や XML 関連技術の利便性、XML による共通フォーマット導入による有用性の確認

3 動的 ACML の提案と実現

本研究では、動的スライサ用のデータスキーマを提案する。以下にデータスキーマの条件を示す。

- コンパクトな設計
- 情報抽出者と解析者が共に扱い易い構造
- 理解しやすい単純な構造

これらの条件を満たすための設計要因として、必要な情報の取捨択一 (構文情報、変数名、変数値、アドレス値など)、抽象度の設定、粒度の設定が考えられる。

まず、Korel と Laski の手法を基に動的 ACML を実現した。これは、実行時点、変数の値、制御命令の真偽値といった動的情報を持つ。さらに、行番号、変数のシンボル名、型情報、定義、使用、制御命令の制御範囲などの構文情報を残した。次に、Agrawal の手法に対応した動的 ACML を拡張した。これにより、変数のアドレスとサイズから、シンボル名ではなくアドレス値での依存関係の抽出ができるようになった。さらに、変数に構造体、配列を加え、明確に判別できるようにした。これは、構造体、配列がそれぞれオフセットをもつためで、この情報を保持することにより、精度の高いスライスが求められるようになる。例えば、配列`a` が存在し、`i` が 1 であれば、`a[i]` は`a` 全体への依存ではなく`a[1]` への依存だとわかる。

4 依存解析ツールの実装

本研究では、前項で述べた 2 種類の動的 ACML に基づき、Korel と Laski の手法、Agrawal の手法の 2 種類の依存解析ツールを実装した。この 2 つを選択した理由は以下の通りである。

- 前者は、最も基本的な手法で、動的スライサと XML の適用の有用性を確認する。
- 後者は、ポインタ解析ができ、より実用的な依存解析ができる。

我々の実装は比較的順調で、開発者 1 人で各々約 2 週間の時間で済んだ。これより、我々の小規模な実装実験に限るが、XML の開発コスト削減に対する有用性を確認した。これは、前節のデータスキーマ定義の条件がうまく実現できたからだと考える。

例えば、動的スライシングの重要な要素で、データ依存関係がある。この解析には、変数の定義、使用の探索が不可欠である。データスキーマに変数の定義と使用の情報を記述する場合、データの抽出者は、抽象構文木の評価順序で素直に抽出したい。一方、解析者は、定義と使用ごとにまとめると探索が容易である。ここで、解析者の利便性を考慮して定義と使用をまとめると、抽出者が被る開発コストの増加が大きいと考え、変数それぞれに定義と使用の情報を持たせ、評価順序に添った形で表現できるように自由度を持たせた。このため、解析者は、同じ情報でも評価順序が違うパターンを考慮しなければならなくなった。しかし、DOM には、指定のタグ名を持った子要素の集合を返すメソッド"`getElementByTagName`"があるため、データスキーマの自由度を吸収することができ、変数の評価順序を気にしないで実装することができた。

5 議論

開発コスト削減の要因は以下の項目と考える。

- 適切なデータスキーマの設計
  - 構文情報の削除や、必要最小限のデータチェックによりコンパクトな実行系列のサイズが実現できた。例えば、文字コードの最大値、最小値、合計を求めるプログラムで文字を 3 個入力したとき、XML 文書は、ソースコードに対して約 20 倍、CSV 形式に対して約 8 倍に収まった。
  - データ抽出者と解析者の間には、データの扱い易さについてトレードオフがある。この差は、データスキーマに自由度を持たせることによって解決した。
  - データスキーマの構造を複雑にすると、抽出者と解析者は、共に理解が困難になるため、理解するための時間がそのまま開発コストに上乗せされる。よって、データスキーマは単純な構造である方が良い。例えば、ある実行時点で取得できる情報は 1 つの木構造を持ってルートの子要素としている。ただし、制御依存が存在するときのみ、依存する制御命令の子要素とする。

- 中間データにテキストファイル形式の導入
  - 中間データを一度ファイルとして抽出するため、実行系列の抽出と解析の分離ができ、データ抽出と依存解析を並行して実装できる。
  - 中間データがテキストファイルであるため、テキストエディタ等でテストケースの作成が手作業できる。
  - テキスト形式のファイルであるため、実行系列を特別なビューア (テキストエディタや Web ブラウザ、ワープロソフト) を用いないで確認できる。
- 実行系列を XML で表現
  - DTD を使うことによって、EBNF 形式で容易にデータスキーマを定義することができる。また、データの妥当性検証を XML パーサを用いて容易にできる。
  - 字句解析、構文解析は、XML 技術である XML パーサを用いることによって補完できる。
  - DOM は、単純な操作が多い。そのため習得に時間を要しない。また、前述のメソッドを使えば、木構造の子要素を集合として扱うことができるため、集合演算も可能である。

6 まとめと今後の課題

我々は、XML を用いて実行系列を表現し、動的スライサに対するデータスキーマを提案し、動的 ACML を定義した。さらに、作成実験として、動的 ACML に基づいた 2 種類の動的スライシングの依存解析ツールの実装を行った。これらは、開発者 1 人で、各約 2 週間で実現した。これより、我々の小規模な実験に限るが、動的 ACML による開発コスト削減と、XML の有用性を確認した。
以下に今後の主な課題を示す。

- ポインタ演算の対応
  現在の動的 ACML では、ポインタへのポインタや、malloc 関数で領域を確保したポインタなどは考慮していない。アルゴリズムレベルでは対応しているので、実際に使用できるか検証が必要である。
- 関数呼び出しの対応
  関数呼び出しは、引数の参照渡し、値渡しの違いや、関数ポインタへのアクセスなど多くの問題を含んでいるため、現在の動的 ACML では対応できない。しかし、動的スライシングの大きな柱の 1 つで、ANSI C フルセットの対応には欠かせない項目である。