2V-04 実行コンテクストのテキスト化と，その後戻り可能なデバッガへの適用

下川僚子, 梅村恭司, 中島浩

豊橋技術科学大学情報工学系

1 はじめに

我々は [2] で, トランスレータを用いて, 実行コンテクストを機種独立なテキスト形式で保存し, さらにその保存した実行コンテクストを用いて実行を再開する方式を提案した. 本報告では, この実行コンテクストのテキスト化手法の後戻り可能なデバッガへの適用について述べる.

デバッガを用いてデバッグを行う際には, まずバグが存在すると予測される箇所の少し手前にブレークポイントを設定し, そこまで実行を進め, そこからはシングルステップで実行の状態を確認しながら実行を進め, バグの場所を絞っていくのが一般的である. しかし, 実行を進めてしまってから少しだけ戻りたい場合などは, ブレークポイントを新たに設定し直すなどをして, 再び実行を最初からやり直さなければならない. 実行に長い時間要するプログラムでは, ブレークポイントの箇所によっては再実行にもやはり長い時間を必要とするため, デバッグの手間が極めて大きなものとなる. このような場合に, デバッガの後戻りの機能は有用であると考えられる.

また, プログラムの一部を変更した場合, 普通はリコンパイルと再実行が必要であるが, これも変更した箇所までの実行に同様の手間を要する. このような場合に, あらかじめ実行途中状態を保存しておき, リコンパイル後に途中から実行を再開できる機能は有用であると考えられる.

我々はこれらの機能を, 実行コンテクストのテキスト化手法を用いて実現する方式を提案する. 我々の実現した方式では, デバッグ対象のプログラムを, トランスレータを用いて実行コンテクストを保存できるようなプログラムに翻訳する. 翻訳結果のプログラムは, プログラムの実行コンテクストを保存し, その保存した実行途中状態を復帰することができるため後戻りが可能となる. 我々は, このようにして後戻りの機能を実現し, さらにこの後戻りの機能をもつデバッガのプロトタイプを作成した.

2 実行コンテクストのテキスト化手法

我々は, 実行コンテクストをテキストとして保存することができるようなプログラムの変換方式を考案した.

実行コンテクストとはある時点でのプログラムの状態すべてのことであり, これを用いることによってプログラムを途中から実行することができるようなものである. 実行コンテクストは, 主としてリターンアドレスのスタックと変数の値から成るが, テキスト化の際に特に問題となるのはリターンアドレスの表現方法である. 我々の提案する方式では, このリターンアドレスに機種独立の名前をつけることが出来るようにプログラムの構造を変更する. 実行コンテクストには, リターンアドレスをこの機種独立な名前に変換してダンプする. これによって, リターンアドレスをテキスト形式に変換することが出来る.

このように変換されたプログラムは, 実行途中状態を機種独立に保存し, さらにその保存した実行途中状態を用いて実行を再開することができる. さらに, プログラムの呼出し関係に影響を及ぼさない関数の細部の変更 (たとえば定数の変更など) をおこなっても, 保存しておいた実行途中状態を用いて実行を再開することが出来る.

3 デバッガへの適用

我々は, 前章で述べた実行コンテクストのテキスト化手法を後戻り可能なデバッガに適用した. この手法を適用するにあたって特に問題となるのは, 実行コンテクストを取り出すタイミング, ソースコードとの対応付け, 実行経過の管理である.

Ryoko Shimokawa, Kyoji Umemura and Hiroshi Nakashima
Texturization of Execution Context and its Application to Backtraking Debugger
Department of Information and Computer Sciences, Toyohashi University of Technology

3.1 実現したデバッガ

我々の実現したデバッガのプロトタイプは, ブレークポイント実行とシングルステップ実行の機能と, 簡単な変数参照の機能を持つ. さらに, ブレークポイントあるいはシングルステップの後戻りの機能を持つ. 我々は, これらの機能をGUIで実装した.

3.2 トランスレータ部分

我々が実現したデバッガのトランスレータ部分は, デバッグ対象のプログラム[1]を2節で述べた方式によりプログラム変換する. このため, 生成されたコードは実行コンテクストをテキスト形式で保存し, 復帰させることができるようなプログラムになっている.

さらに, 毎ステートメント（セミコロン）ごとに, 実行コンテクストを保存するようなコードを追加し, そのために毎ステートメント毎に関数を分けている. つまり, 実行を操作できる単位は, ソースプログラム中のセミコロンの場所となっている. ここで, 文献[2]のようにすべての情報を保存すると, 大量の中間状態を生成することになるので, 差分情報のみを保存するようにしている. これにより, オーバーヘッドを減少させた. また, ブレークポイントのチェックなどの実行を制御するためのコードも追加している.

翻訳結果のプログラムを実行すると, プログラムは毎ステートメント毎に実行コンテクストを保存する. そして, 実行を後戻りさせたい場合には, 保存しておいた実行コンテクストの内容を読み込むことにより, 後戻りすることが出来る.

3.3 リコンパイル後の再実行

我々は, 実行コンテクストがテキストとして保存されるというこの方式の特徴を利用して, リコンパイル後にも途中から再実行できる機能を実現した. これは, プログラムの一部を変更してプログラムをリコンパイルしても, 実行を途中から行なうことができるというものである.

通常の場合, プログラムを少し変更しただけでもコードの場所は移動してしまう. したがって, 例えばコアダンプを用いてプログラムの実行を再開しようとしても, 関数スタックの中のリターンアドレスが不正となり, 正しくプログラムを続行できない. しかし, 我々の方式では保存された実行コンテクスト中の戻り場所はアドレスに依存しない名前になっているので, コードの移動は問題にならなくなっている.

たとえば, 関数呼び出しを含まない代入文を追加したり, 削除したりする操作は安全にできる. もちろん, すでに実行したプログラムの部分のプログラムを変更をした場合は, その変更を反映したような実行コンテクストにならないが, 実行していない部分にどのような複雑な代入文を追加したとしても, 変更する前と変更の後の実行コンテクストは最初から計算しても同一である. したがってこのような変更において, 変更前のプログラムで生成した実行コンテキストを, 変更したプログラムに引き次ぐ操作が可能となっている. また, 実際のシステムではさらに大きな変更が可能となっている.

通常の方法では, 定数の変更程度の変更しか許されないのに比べて, 可能なプログラム変更の範囲が大きく増加している. これが, 実行コンテキストをコストをかけて機種非依存にする効果となっている.

4 考察

我々は, 実行コンテクストをテキスト化する手法を用いて後戻り可能なデバッガを実現する方法について提案し, 実際にそのプロトタイプを作成した. 我々の実現したデバッガは, 実行時間についてはまだ課題が残るものの, リコンパイル後の再実行が可能になるというメリットがある. 今後は, プログラムを翻訳する際に最適化を行い実行時間を短くすることと, 後戻りの難しいプログラムについての考察を行うことを課題として考えている.

5 まとめ

我々は本論文で, 実行コンテクストをテキスト化する手法を延べ, さらにこの手法を後戻り可能なデバッガへ適用する方法について述べた. 我々の実現したデバッガは, 後戻りが可能であり, さらにリコンパイル後の再実行が可能であるという特徴がある.

参考文献

[1] Jonathan B. Rosenberg 著, 吉川邦夫訳, How Debuggers Work 「デバッガの理論と実装」, アスキー出版局,1998.

[2] 下川僚子, 梅村恭司, トランスレータを利用した機種非依存な実行移送方式, 情報処理学会論文誌, Vol.40, No.6, pp2553-2562, 1999.

[1] 現時点でのデバッグ対象のプログラムは,C のサブセットである