開かれた環境における実行時プログラム変換

浅井　健一

■ 研究のねらい

インターネットが普及するにつれて、データばかりでなくプログラムもインターネット上を移動して実行されるようになってきている。とある場所で作られたプログラムは、別の場所に移動して、そこにあるデータを使いながら計算を進め、さらにまた別の場所に移動して、そこにあるプリンタに結果を出力したりする。このような計算が従来の計算と違うのは、プログラムの実行される環境があらかじめわかっておらず、動的に変化する、という点である。実行環境があらかじめわかっていれば、その情報をもとにプログラム変換（例えばコンパイル）を行い、プログラムをより効率的なものに変換することができる。しかし、実行環境が予測不可能な出来事（例えばプリンタの故障）によって変化する可能性がある場合には、従来のようにあらかじめプログラム変換をかけておく、ということができず、従って効率的な実行が難しくなっている。

さきがけにおける研究の目的は、このように動的に変化する環境で、どうやってプログラムを効率化したら良いのか、その手法を確立することにある。あらかじめプログラム変換をかけるのではなく、そのときの状況に応じてプログラム変換をかけるにはどうすればよいか。そして、そこにはどのような問題があるのか。これらの問題を解決することを目指して３年間、研究を行った。その背景、内容、技術的課題を、研究課題名「開かれた環境における実行時プログラム変換」に出てくる単語を使ってもう少し詳しく説明してみよう。

「プログラム変換」

プログラム変換というのは、プログラムを入力として受け取り、それを別のプログラムに変換するもので、その代表はコンパイラである。変換の目的は普通、効率の向上で、コンパイラはまさにそのために存在しているのだが、この研究のように動的に変化する環境を扱っているときには、プログラム変換にはもうひとつ別の重要な役割がある。それは「一般性と効率の両立」である。動的に変化する環境にも対応できるプログラムを組もうとすれば、そのプログラムは必然的にとても一般性の高いどこでも使うことのできるプログラムとなる。しかし、一般性の高いプログラムは全ての状況に対応できるように作られているため、効率が悪い。一方、特定の環境を仮定してプログラムを作れば、その環境の情報を使って効率的なプログラムを作ることができるが、そのようなプログラムは仮定した特定の環境でしか実行することができない。このトレードオフを解決するのがプログラム変換である。つまり、一度、一般的なプログラムを作成しておき、どの環境で実行するかが判明した時点でプログラム変換をかけ、その環境に適応した効率的なプログラムに変換するのである。

本研究では、このようなことを念頭において、プログラム変換の中でも部分評価という手法を取り上げた。部分評価と言うのは、部分的にわかっている情報を最大限に利用してより効率的なプログラムに変換する手法である。これは、ちょうど、実行環境が判明した時点で、その情報を使って効率化する、という状況に対応している。

「開かれた環境」

従来、部分評価と言えば、プログラム全体が与えられているときに、それをどうやって効率化するか、というのが普通であった。しかし、本研究で扱いたいのは、プログラムの実行環境が自分だけで閉じてはおらず、外に開かれており、外的要因によって変化するような状況である。このような状況をここでは「開かれた環境」と呼んでいる。

開かれた環境は、技術的には大きく次のふたつで特徴づけることができる。

- 部分評価するプログラムがあらかじめ全て与えられているわけではない。

これは、例えば移動先のプリンタの機種がわからず、従って、プリンタ出力の命令がまだ手元にない、という状況に対応している。

- プログラムの一部が、実行の途中で変化する。

これは、例えばプリンタが新しくなったため、そのドライバプログラムを入れ換えた、という状況に対応している。

従って、これらふたつの状況に部分評価をどう対応させるか、というのが本研究の主要な課題となる。

「実行時」

研究課題に出てくる最後のキーワードは「実行時」である。本研究ではこれにふたつの意味を込めている。ひとつは（あらかじめプログラム変換をかけるのではなく）実行時に「その場の状況に応じて」プログラム変換をかける、という意味である。部分評価を含む従来のプログラム変換は、実行前に変換を行うものがほとんどであった。しかし、開かれた環境では環境がどのように変化するかわからないため実行前に変換を行うことはできない。従って、実行時に実行環境が明らかになった時点でプログラム変換を行う必要がある。

もうひとつは、効率良くプログラム変換を行う、という意味である。実行前に行うプログラム変換は、多少、時間がかかっても結果のプログラムが十分に速いものであれば構わなかった。しかし、実行時にプログラム変換を行うのであれば、プログラム変換自体の速度が全体の実行速度に影響してしまう。たとえ効率の良いプログラムを出力することができても、それにかかるプログラム変換の時間が長過ぎては全体として効率は下がってしまう。

■ 研究成果

以上のような点に考慮しながら３年間、研究を行ってきた。その研究成果は主に３つの部分にまとめることができる。以下、それらを順に説明していく。

１. Offline 部分評価器の能力の向上

第１部では、部分評価の効率に着目して、offline と呼ばれる部分評価法を採用し、開かれた環境で使用することを目指してその能力向上をはかった。以下、それについて説明する。

部分評価手法には offline の方法と online の方法と呼ばれるものがある。

- offline の方法というのは、束縛時解析と呼ばれる解析を使って、あらかじめプログラムを部分評価時に実行可能か不可能かを求めておく方法である。実際に部分評価を行う際には、解析の結果、

実行できる、とされた部分だけを実行し、それ以外は何もせずに返す。この方法は、あらかじめ解析で実行できる部分、できない部分を求めておくので、部分評価自体はとても高速に行うことができる、という利点を持つ。しかし、一方で解析結果が甘いと十分に部分評価できなくなってしまう。

- 一方、online の方法というのは、解析はせず、部分評価時に実行できるかできないかをいちいち判断しながら部分評価を行う。部分評価時に実行できる、と判断されたものは全て実行できるので、部分評価結果は（解析で近似が入ってしまう）offline の方法よりも普通、より良いものになる。しかし、いちいち実行できるかどうかの判断を部分評価中に行うので効率はあまり良くない。

第１部では、効率的に部分評価できる点に注目して前者の offline の方法を採用し、その能力向上をはかった。offline の方法の要点は、変換されるプログラムを（部分評価時に）実行可能な部分と不可能な部分とにうまく分離することである。うまく分離することができれば、十分な部分評価を行うことができるが、うまく分離できないと近似が入り、結果が悪くなってしまう。特に、開かれた環境では、将来、どのような部分評価が行われるかわからないので、うまく分離できない場合が多かった。そこで、うまく分離できない場合の一部に対して「実行できるが実行しなくても良い」という新しいカテゴリを導入し、「実行可能」、「実行不可能」とあわせて分類することで、今まで近似が入ってしまっていた部分をより正確に扱えるように拡張した。

この方法は、従来の offline の方法の良い点を失うことなく、解析を強力にするものとなっている。特に、解析がプログラムの長さに対してほぼ線形の時間で行える、という性質を受け継いだまま能力を向上させることに成功している。

２. Offline 部分評価器の動的な環境における限界

第１部で作成した部分評価器は、従来のものよりは強力になっているものの、開かれた環境で使用しようと思うとまだ力不足であった。第２部では、開かれた環境でも使用可能なところまで能力をあげるべく検討した。しかし、結論から言うと、それは難しい、ということがわかった。以下、それについてもう少し詳しく説明する。

offline 部分評価器の要点は「実行可能」な部分と「実行不可能」な部分に分離すること、と述べたが、開かれた環境においては、プログラムが将来、どのように使われるかわからないので、あらかじめ分離しておくのは不可能である。特に、プログラムの引数（パラメタ）についても何の仮定も置くことができない。これは、引数の分類が与えられた上で解析をしている従来の方法は使えないことを意味している。そこで、束縛時解析の中でも「多相」のものを考えることでこの問題を克服しようと試みた。

多相の束縛時解析というのは、引数の分類を具体的に与えるのではなく「変数」の形で与えるものである。つまり、引数が「実行可能」なのか「実行不可能」なのかを具体的に与えるのではなく、その引数の分類は x である、とし、x は「実行可能」か「実行不可能」のどちらかの値をとる、とするのである。このようにしておくと、その引数が実際に「実行可能」なのか「実行不可能」なのかわかった時点で x にそれを代入してやれば、解析をやり直すことなくプ
ログラム各部の分類が得られる、という仕組みである。

多相の束縛時解析と言うのはすでに以前に発表されているのだが、このアイディアを、開かれた環境における解析に用いてみようとしたところ、従来の方法では不十分であることがわかった。従来の方法は、let 型の多相を用いているのだが、let 型の多相では計算結果としてプログラムが返ってくる

場合に、それは多相にならない。開かれた環境では、計算結果をさらに部分評価することもあるが、それが多相になってくれないため、十分に部分評価されなくなってしまうのである。

そこで、let 型の多相を一般の多相へと拡張しようと試みた。すなわち、let 文のみではなく、計算結果のプログラムも多相になるように枠組みを拡張しようとしたのである。しかし、ここでわかったことは、一般の多相に拡張しようと思うと、解析が非常に複雑になり、一般には解析をしようとしているプログラムと同じ複雑度になってしまう、ということである。普通、解析をする時には、プログラムを抽象化することで、実際に実行するよりはずっと効率的に種々の情報を得るのだが、解析の複雑度がもとのプログラムと同じになってしまっては、解析をするのも実行をするのも同じになってしまい、解析をする意味がなくなってしまう。この結果は、何の仮定も置かない開かれた環境での束縛時解析は現実的ではない、ということを示している。

３. Online 部分評価器のインタプリタへの融合

第２部で、offline の方法を取るのは難しい、ということがわかったので、第３部では（offline の方法の特徴である部分評価の効率はあきらめ）部分評価の効率はいまひとつだが、より強力である online の方法を採用した。その上で、部分評価器をインタプリタに融合した。インタプリタ環境は、（ユーザが将来、どのような命令を実行するかわからないと言う意味で）開かれた環境と見ることができる。従って、この融合により開かれた環境における部分評価が実現している、と見ることができる。以下、融合にあたって考えなくてはいけない点をもう少し詳しく述べていく。

従来、部分評価する時には、部分評価されるプログラムが全体として与えられているのが普通だった。しかし、開かれた環境では、部分評価を行う時点ですでにプログラムは一部、実行されているのが普通であり、またプログラム自体も途中で変更されているかも知れない。従って、部分評価されるプログラムが何であるかは明らかではない。また、プログラムが一部、実行されてしまっている、ということは、データが計算機上に作られてしまっている、ということである。これらのことから、融合にあたっては主に以下のふたつのことを考えなくてはいけない、ということがわかってきた。

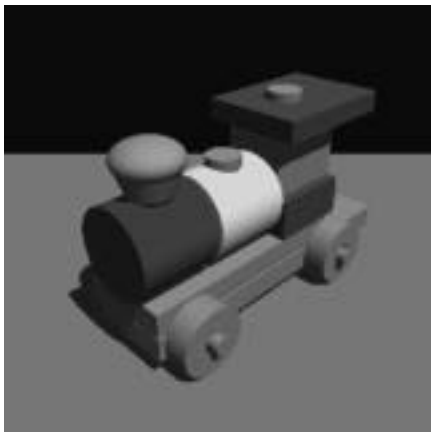
副作用との関係

まず最初はプログラムの変更、つまり副作用と部分評価との関係である。副作用は、部分評価を実行する前、実行中、実行後の３か所で起こりうる。このうち、部分評価前の副作用はあまり問題を引き起こさない。部分評価前にどのような変更が加わろうと、その結果、得られたプログラムに対して部分評価をかければ、問題はない。従来の、入力プログラムは全て与えられている、という仮定を捨て、現在のプログラムに対して部分評価をかける、という方針を採用するだけで対応できる。一方、部分評価中の副作用は、従来の副作用を含んだ部分評価と本質的に同じ問題である。ここには開かれた環境特有の問題は存在しないので、本研究では特に新しい提案は行っていない。最後に、部分評価後の副作用には、少し注意をする必要がある。というのは、部分評価というのは、現在、得られる情報をもとに、プログラムを効率化する、つまり現在、得られる情報を使ってしまうのである。ところが、もしその使ってしまった情報が将来、副作用によって変更されてしまったら、部分評価結果のプログラムは誤ったものになってしまう。つまり、部分評価後の副作用は、部分評価結果を無効なものにしてしまう可能性がある、ということである。

計算機上にすでに作られてしまっているデータの扱い

もうひとつのポイントは、計算機上にすでに作られてしまっているデータの扱いである。従来の部分評価では、入力はまだ実行されていないプログラムだったので、計算機上にはまだ何のデータも作られてはいない。しかし、開かれた環境では、部分評価前にすでに計算機上にデータが作られてしまっている可能性があり、これらと矛盾しない形で部分評価をする必要がある。特に、部分評価結果のプログラムがこれらの値を参照する時には、それらを複製しないように気をつけなければならない。データの複製を避けるためにとった方法は、部分評価結果にそれらのデータへの直接参照を許す、というものである。つまり部分評価結果が一部、プログラムではなく、計算機上のデータへのポインタになってしまうのである。これは部分評価をプログラムからプログラムへの変換、ととらえていると奇異に感じられるが、考えてみると、部分評価の目標は効率的にプログラムを実行することであり、そのプログラムテキストを得ることではない。効率的に実行できさえすれば、別にプログラムテキストの形で結果が得られる必要はないわけである。

以上のような点を整理すると、比較的、簡単に部分評価器をインタプリタに融合することが可能である。でき上がったシステムは（副作用との相互作用に気をつけていれば）プログラムを実行し、環境を変化させても、その時の状況に応じて部分評価できるものとなっている。従来、ファイルに部分評価されるプログラムを完全な形で用意する必要があったのが、インタプリタ中で単に、この式を部分評価せよ、という命令を出すだけで簡単に部分評価をかけられるようになったのである。



応用として、このシステムを使ってレイトレーシングを実装した。そしてコアとなる部分に部分評価命令を挿入することで、約１０%から２倍程度の速度向上が得られることを確認した。右の図はこのレイトレーサーで作成した画像の例である。

■ 今後の展開

途中で多相の束縛時解析の研究が思い通りに進まなかったこともあって、３年間の研究で得られたものは、当初の予想に反して比較的、わかりやすいものとなった。つまり

- 副作用との関係が重要。特に部分評価後の副作用は部分評価結果を無効にすることがある。
- 部分評価結果には、計算機上のデータへの直接参照を許すべき。

の２点である。これらは、いずれも実行時のプログラム変換をしようと思えば自然なことである。しかし、それらを関数型言語の世界できちんと述べたのにはそれなりに意義があったかと思う。一方、これらをきちんと述べたために、新たな問題も明らかになってきている。例えば、部分評価結果が無効になる可能性を指摘したが、ではいつ無効になるのかを機械的に検出することは可能だろうか？ この問題は、単純に検出しようと思うと膨大なデータ（部分評価時に使用するデータ全て）を保存する必要があることがすでにわかっている。これをどこまで減らせるかは、何らかの仮定が必要そうだ、というのが現在の感触である。

多相の束縛時解析は、直接、一般的なケースを扱うのは難しい、というのが結論だが、では、その場で incremental に解析を行う、などのアプローチがとれるかどうか、というのは面白い研究課題である。インタプリタ上で、部分評価せよ、という命令を出すのと同じように、束縛時解析せよ、という命令を出して、その結果を使って部分評価する、などの方法は不可能ではないように感じられる。そうすれば、offline のアプローチによる効率の良い部分評価を実現できるかも知れない。

最後に自己反映言語のコンパイルについて触れよう。この研究の大きな動機のひとつに自己反映言語のコンパイルというのがあった。自己反映言語と言うのは、動的に変化する環境をモデル化したような言語で、そのコンパイルにはまさにこの研究でしてきたようなことが必要である。当初の予定では自己反映言語のコンパイルまで実現させるはずだったが、残念ながらひとつ技術的な課題が残っていて実現には至っていない。しかし、ここでの研究で主要な部分ができあがり、先が見えてきているので、是非、近い将来に実現したいと思っている。

謝辞

さきがけにおける３年間では、安西先生をはじめとする多くの先生方、仲間達に励ましの言葉を頂きました。ここで研究できたことを誇りに思うとともに、今後もそれに恥じぬよう頑張っていきたいと思いますので、今後ともよろしくお願い申し上げます。

■ 成果リスト

- Kenichi Asai "Binding-Time Analysis for Both Static and Dynamic Expressions," Static Analysis (LNCS 1694), pp. 117-133 (September 1999).
- 浅井 健一 「静的かつ動的な式を許すような部分評価器のための束縛時解析」 コンピュータソフトウェア、 Vol. 17, No. 3, pp. 20-37 (May 2000).
- Kenichi Asai "Integrating Partial Evaluators into Interpreters," Semantics, Applications, and Implementation of Program Generation (LNCS 2196), pp. 126-145 (September 2001).