神奈川大学理学部　情報科学科　馬谷研究室

データフロー解析フレームワークにおける不動点アルゴリズムの性能調査

鈴木 滉弥 201703360

1　はじめに

データフロー解析とは，ノードを持つ制御フローグラフ (CFG) に対して $Nodes = v_1, v_2, \cdots, v_n$ とし，L が抽象状態をモデル化する格子である L_n で作業を行う．ノード v_i がデータフロー方程式 $[\![v_i]\!] = f_i([\![v_1]\!], \cdots, [\![v_n]\!])$ を生成すると仮定し，$f(x_1, \cdots, x_n) = (f_1(x_1, \cdots, x_n), \cdots, f_n(x_1, \cdots, x_n))$ を定義することにより，結合関数 $f : L_n \to L_n$ を構築する．ここで，不動点アルゴリズムを適用すると，$[\![v_1]\!], \cdots, [\![v_n]\!]$ の解が得られる．

本研究では TIP と呼ばれる小さな命令型プログラミング言語を Scala で実装し，データフロー解析を行うものとする．

本研究の目的は以下の二つである．

- 不動点アルゴリズムが改良されていく過程で，その問題点と改善されている点を明確にする．
- 各不動点アルゴリズムの性能を比較し，効率の差を検証する．

2　検証方法

検証には，Static Program Analysis[1] にで紹介されている以下のアルゴリズム 4 つを用いる．

- Naive Fixed Point Solver(Naive)
- Worklist Solver(WL)
- Worklist Solver with Reachability(WLR)
- Worklist Solver with Reachability and Propagation(WLRP)

上記アルゴリズムを用いて符号解析を実行しその実行時間を比較して検証する．

以下，それぞれのアルゴリズムについて簡潔に説明する．

Naive Fixed Point Solver

最も簡単で基礎的なアルゴリズムである．

```
procedure NaiveFixedPointSolver(f)
    x := ⊥
    while x ≠ f(x) do
      x := f(x)
    return x
```

プログラムの各ノードのデータフロー方程式を x に代入していき，すべてのデータフロー方程式が変化しなくなるまで計算を繰り返すというものである．

Worklist Solver

再計算が必要な可能性のある式のみをワークリストに追加していき計算を行うので Naive の欠点である冗長な計算を減らすことができる．

```
procedure WorklistSolver(f1,...,fn)
    (x1,...,xn) := (⊥,..., ⊥)
    W = (v1,...,vn)
    while W ≠  ϕ do
      vi := W.removeNext()
      y := fi(x1,...,xn)
      if y ≠ xi then
        xi := y
        for each vj ∈  dep(vi) do
            W.add(vj)
    return (x1,...,xn)
```

Worklist Solver with Reachability

解析に用いるプログラムの中で到達することのない箇所を事前に除いて解析を行う手法である．この解析方法を行うことで，コード量が多いが実際に使う部分は限られているようなプログラムの解析時間を大幅に短縮することができる．

Worklist Solver with Reachability and Propagation

これまでの 3 つとは異なり，変更があった際ワークリストに追加されるノードは後続節ではなく，先行節が追加されるようになっている．これは，変化していなくても後続節というだけでワークリストに追加されてしまうことが多いのでより冗長な計算を省略することができる．

3　検証結果

検証には，Static Program Analysis[1] の配布コードに含まれているサンプルプログラム全 33 個のうち 5 つを抜粋して考察する．計測箇所は，プログラムの実行開始から終了までとする．4 つの不動点アルゴリズムを各 5 回ずつ計測し，その中央値を結果とする．

結果を図 1 に示す．グラフの縦軸は計測時間 (ミリ秒) とし，横軸は解析対象プログラム名とする．各項目の要素は左から，青色が Naive，橙色が WL，灰色が WLR，黄色が WLRP とする．

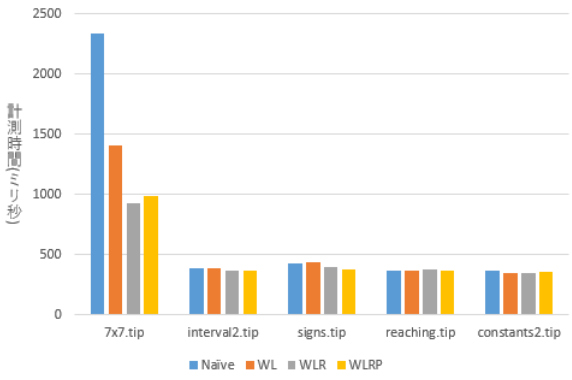

図 1　検証結果

4　考察

検証結果より，右肩下がりのコードと変化の差が少ないコードがあると判明した．

7x7.tip はソースコード自体は単純だが，ほかと比べるとコード量が多く，時間がかかっていると言えるだろう．また，コード量に対して変更するノードが少ないため，WL の効果が大きくみられる．

interval2.tip，signs.tip はコードが少し複雑な計算が多い傾向にあるため，右肩下がりになっている．これらのプログラムは変更するノードが多いため WLRP が有効に働くと考えられる．また，複雑なコードであればあるほど変化の差が大きくなるということが分かった．

検証前の予想ではどのプログラムでも右肩下がりになり，速度の改善が見られると予想していた．しかし実際には，単純な計算が多い傾向にある reaching.tip，constants2.tip は差が少なく横並びという結果になった．これはプログラムの構造によってそれぞれ最適な不動点アルゴリズムがあり，複雑なプログラムをより早く正確に解析するためにアルゴリズムが改善されたからではないかと考えられる．

5　まとめ

本研究ではデータフロー解析における不動点アルゴリズムの性能について調査した．各アルゴリズムの実行速度を測定することにより検証を行った．結果，必ずしも後続のアルゴリズムが速いというわけではなく，プログラムの構造によって適したアルゴリズムがあるということが分かった．これは，複雑なプログラムの解析をより効率よく進めるための改良が行われたためだと考えられる．不動点アルゴリズム改良の手法は今回使用したもの以外にも存在するのでそれらの調査をすることが今後の課題である．

参考文献

[1] Anders Møller and Michael I. Schwartzbach, “Static Program Analysis”December16,2019 https://cs.au.dk/~amoeller/spa/