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を用いた制御フロー解析の実装

佐藤 涼 201703405

1　はじめに

データフロー解析は，制御フローグラフ (CFG) と有限の束からなる．束は, CFG のそれぞれのノードについての抽象的な値が記述されている．また，各ノード v に，束の要素である制約変数 $[\![v]\!]$ を割り当てる．解析対象のプログラムに対して，制御フローグラフとそのノードにデータフロー制約を割り当てるルールを指定することで，フレームワークをインスタンス化することができる．制約に対する全ての解が，最小の解を計算できれば，可能な限り高い精度が得られる．既に実装されている符号解析は，プログラム内の変数や整数値の可能な式の抽象化するものである．

しかし，プログラム中の全てのコードを対象に制約を生成する方法ではすべての整数や式を解析しなければならない．それは使わない可能性のあるものも解析するということである．そこで，制御フロー解析と組み合わせることで抽象値を最小限にし，より精度の高い結果を返すことができる．特に，関数の呼び出しがあった場合，その関数に符号の値はついていない．そこで制御フローを解析すること関数呼び出しを値とすることを目的として本研究を進める．

2　基にした既存の制御フロー解析

関数を値 (つまり第一級関数) として導入したり，メソッドを持つオブジェクトを導入したりすると，制御フローとデータフローが急激に絡み合う．各呼び出し場所で，どのコードが呼び出されているかを確認することが重要になってくる．制御フロー解析は，このようなプログラムの手続き間の制御フローを保守的に近似することである．[1]

第一級関数については

$$E \to E(E_1, ..., E_n)$$

では，どの関数が呼び出されるかを構文から直接確認できない．正しい数の引数を持つ関数ならば，どんな関数でも呼び出せると仮定することで，粗くても健全な CFG を得ることができる．そこで，制御フロー解析を使うことで，より良い結果を得ることができる．また，束は，各関数名 X に対して X を含むトークンの集合の冪集合でであり，部分集合の包含で順序付けられている．構文木ノード v に対して，v が指す関数の集合を表す制約変数 $[\![v]\!]$ を導入する．v という名前の関数に対して，次のような制約条件が得られる．

$$f \in [\![f]\!]$$

また，代入文 $X = E$ に対しては，次のような制約がある．

$$[\![E]\!] \subseteq [\![X]\!]$$

最後に，関数呼び出し $E(E, ..., E_n)$ について定義を示す．引数 $a_f^1, ..., a_f^n$ を持つ関数 f を特定し，以下の制約条件を満たす式 E'_f を返す．

$$f \in [\![E]\!]$$

$$\Rightarrow ([\![E_1]\!] \subseteq [\![a_f^1]\!] \wedge ... \wedge [\![E_n]\!] \subseteq [\![a_f^n]\!]$$

$$\wedge [\![E'_f]\!] \subseteq [\![E(E_1, ..., E_n)]\!])$$

以上を使用することで，プログラムに対して制約条件が生成される．さらに，最小解が解析結果として得られる．本研究では，符号の型が正しいと思わ

れる呼び出しの関数 v に対してのみ制約を生成することによって，さらに精度の高い解析結果を得られることを目的とする．

3　データフロー解析を使用した方法

ここでは前節の制約に基づく手法を基にデータフロー方程式を定義していく．また，符号の解析をする際に生成する制約についても示す．ノード v の直前の抽象状態を結合する補助関数 *JOIN(v)* を以下で定義する [1].

$$JOIN(v) = \sqcup_{w \in pred(v)} [\![w]\!]$$

JOIN(v) は，プログラムのすべての制約変数

$$[\![v_1]\!], ..., [\![v_n]\!]$$

の関数である．次にいくつかのノード v に対する制約規則を示す．

- X = E:

  $$[\![v]\!] = JOIN(v)[X \mapsto eval(JOIN(v), E]$$

  eval 関数は，抽象状態 σ に対する式 E の抽象値の評価を行う．

  – $eval(\sigma, X) = \sigma\ (X)$

  – $eval(\sigma, I) = sign(I)$

  – $eval(\sigma, input) = \bot$

  – $eval(\sigma, E_1 op E_2) = op(eval(\sigma, E_1), eval(\sigma, E_2))$

- $var X_1, ..., X_n$:

  $$[\![v]\!] = JOIN(v)[X_1 \mapsto \mathrm{T}, ..., X_n \mapsto \mathrm{T}]$$

  新しく宣言されたローカル変数は初期化されていないので，どのような値を持つこともできることを示している．

- 他の種類の CFG ノードについては変数の値に影響しないので以下のようになる．

  $$[\![v]\!] = JOIN(v)$$

上記の代入の形式で関数の呼び出しを抽象化することに着目する．第 2 節で示した関数呼び出し

$$E(E, ..., E_n)$$

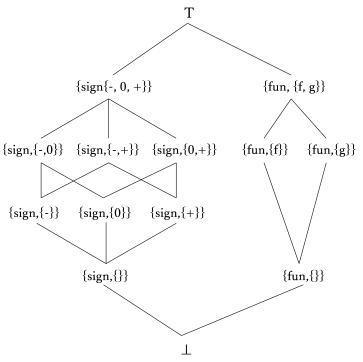

図 1　束

について定義された式をデータフロー方程式の形で表すと以下のようになる．

$$eval(\sigma, E(E, ..., E_n)) = fun(E(E, ..., E_n))$$

本研究では，符号解析と制御フロー解析の両立のために抽象値の束も変えなければならない．図 1 の左側は符号の抽象的な値，右側は関数を抽象的に表す値を記述する．ここで，呼び出される関数は対象のプログラムによって様々なのでここでは 2 個の関数であるとする．また，解析する値が符号か関数呼び出しかタグをそれぞれ $sign$，fun とつけることによって判別できるようにしておく．

4　まとめ

実装できれば，解析によって得られる解によって，解析する必要のある CFG ノードが限られてくるので，最小解に近づき，正確な解析結果が得られるようになるはずである．本研究では，データフロー解析手法で制御フロー解析をすることにより解を最小解にすることが目的であったが，これに加えて，ポインタやレコードを含んだものも解析することによってより精度の高い解析結果になるだろう．

参考文献

[1] Anders Moller and Michael I. Schwartzbach, “Static Program Analysis”, `https://cs.au.dk/~amoeller/spa/spa.pdf`, December 16, 2019.