1L-02

値への追跡子付与による動的プログラム解析手法

西谷 幸太郎 † 小宮 常康 ‡
†,‡ 電気通信大学 大学院情報理工学研究科

1 背景

プログラムを実行すると，様々な値が生成・参照されながら計算が進む．例えば以下の JavaScript プログラムを実行すると，1 行目の値 1 は変数 a に格納されたのち，2 行目で変数 a へのアクセスによって取り出される．また，2 行目の式 a + 2 の評価結果として生成された値 3 は，変数 b に格納されたのち，3 行目で変数 b へのアクセスによって取り出され，関数 console.log の実引数となる．このように，プログラム実行時の値の流れを理解することは，そのプログラムの意味を理解するために重要である．

```
1 let a = 1;
2 let b = a + 2;
3 console.log(b);
```

追跡子とは，流体の流れを追跡するために使用される物質のことであり，生体内での薬品の伝播の状態やその範囲の調査に使用される．本研究では，追跡子による調査のアイデアをプログラムの領域に活用した，新しい動的プログラム解析手法を提案する．この手法では，プログラム中の追跡子付与ポイントを通過した各値に追跡子を付与する．追跡子は，例えば付与ポイントの場所情報などを含んでおり，それらの情報を活用して，デバッグやプログラム解析に必要な操作を実現する．

関心のある値を直接的に特定し，その値に狙いを絞って追跡子を付けることができないこともある．しかしそのような場合でも，例えば，関心のない値に追跡子を付け，その追跡子を持たない値を探すことで関心のある値とその流れを見つけ出せることもある．そのためには，追跡子を持つ/持たない値の集合に対する集合演算等を施せればよい．そこで本研究では，追跡子が持つ情報をより有効に活用できるように，指定した追跡子を持つ/持たないなどの追跡子に関する条件を満たす値のみを絞り込むためのクエリ言語を提案する．この言語は集合演算をベースとしており，複数のクエリを組み合わせてより複雑なクエリを作成することが可能である．そのため，ユーザの関心事をクエリとして柔軟に表現できる．

Dynamic program analysis utilizing tracers attached to values
†Kotaro Nishitani
‡Tsuneyasu Komiya
†,‡Graduate School of Informatics and Engineering, The University of Electro-Communications

2 提案手法

2.1 値への追跡子の付与

本手法において，プログラム実行時の全ての値は，複数個の追跡子を自身に保持する能力を持つ．

値は，式を評価した結果として得られる．値への追跡子の付与もこのタイミングで行う（追跡子付与ポイント）．このとき，付与する追跡子には，式を評価した時刻を暗黙的に含める．また，各値は暗黙的に ID を持つ．

本手法では以下の 3 種類の追跡子を導入する．

- 値が通過したプログラム中の場所を記憶する追跡子
- 値の生成源の値（生成のために使用された値）を記憶する追跡子
- ユーザ指定の文字列を記憶する追跡子

値がプログラムを通過した場所について，1 節のプログラム中の値 1 を例にすると，図 1 のグレー背景部分ような追跡子が付与される．同プログラムの値 3 は，値 1，2 から生み出されているため，値 3 には，値の生成源を示す追跡子が付与される（図 1 破線部分）．ユーザ指定の文字列については，例えば同プログラムで変数 a にアクセスしている場所を通過した値に文字列 *refa* を記憶する追跡子を付与するようユーザが指示した場合，実行時にこの場所を通過する個々の値 1 にその追跡子が付与される．

最後に，1 節のプログラム実行後に得られる，実行時の全ての値及び付与された追跡子は図 1 全体のようなグラフ構造を形成する．

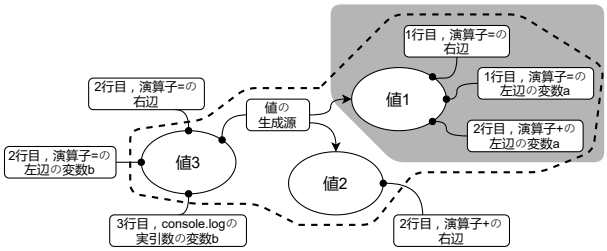

図 1　値とその追跡子のグラフ構造

2.2　クエリを用いた解析の例

前述のグラフ構造に対するクエリを用いてプログラムの解析（デバッグ）を行う例を示す．以下のプログラムは，数値の配列 a の要素の総和を求めるプログラムである．このプログラムを実行すると，結果として意図しない値 NaN が出力される．

```
1 let a = [3, 1, 4]; let sum = 0;
2 for (let i = 0; i <= a.length; ++i) {
3   sum = sum + a[i];
4 }
5 console.log(sum); // NaN
```

本手法を次のように用いることで，結果として得られた値 NaN を起点にデバッグを進めることができる．まず，以下のクエリを実行して，値と追跡子のグラフ構造の中から，5 行目の式 sum の場所を通過した値のみを取得する．

`findValue(〈5 行目の式 sum の場所〉)`

この結果として，値 NaN とその ID 情報 `{id:id24,value:NaN}` が得られる．次に，出力された値 NaN の生成源を調べる．そのために，値 NaN の ID id24 を使用して，クエリ `findGen(id24)` からその情報を取得する．この結果として，以下の値が得られる．

`{id:id18,value:8},{id:id23,value:undefined}`

これらの値の通過場所を調べるためには，クエリ `showTrace(id18)`，`showTrace(id23)` を実行する．すると図 2，3 のように，値の通過経路がプログラム字面上に描画される．これにより，値 8，undefined はそれぞれ，3 行目の右辺の式 sum，左辺の式 a[i] を通過した値であると分かる．つまり，値 NaN は，3 行目の演算 sum + a[i] として，8 + undefined が行われたことによって生成されたと分かる．同様に，クエリ `findGen(id23)` から，値 undefined の生成源として以下の値が得られる．

`{id:id3,value:[3,1,4]},{id:id20,value:3}`

これらの値の通過場所は，クエリ `showTrace(id3)`，`showTrace(id20)` によって，それぞれ，3 行目の配列 a の値，配列の添字の値 3 であると分かる．よって，値 undefined は，配列外参照 a[3] によって意図せず発生したものであると分かる．

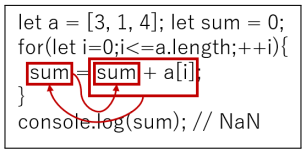

図 2　showTrace(id18) の実行結果

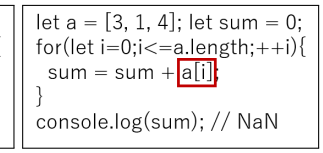

図 3　showTrace(id23) の実行結果

今回発見した値 undefined は意図せず発生したものだったが，ユーザが意図してプログラム中に undefined を記述する場合もある．それらの undefined を区別するために，ユーザ指定の追跡子とクエリを利用できる．ユーザは意図して記述した式 undefined を通過する値に追跡子文字列"intended"を付与するように追跡子付与ポイントを設定し，"intended"を持つ値を除外するクエリ `nhasMkr("intended")` を実行する．これにより，意図して発生させた値 undefined を解析対象から除外することができる．

3　関連研究

dynamic program slicing という動的解析手法がある [1]．この手法では，実行された命令と，その命令間の依存関係を表現したグラフから，指定した命令に依存関係のある部分グラフ（dynamic slice）を取得する．本研究でも，このようなグラフの再現が可能なのではないかと考えている．

追跡子のアイデアをデバッグ以外の用途として利用している研究として，権藤らの解析器 TBCppA [2] がある．この解析器では，C 言語のプリプロセッサ処理前後の字句の対応関係を取得するために XML 風の文字列を追跡子として利用している．

4　まとめ

値への追跡子の付与と，追跡子が持つ情報を有効に活用するためのクエリを組み合わせることで，プログラムの動的解析を実現する手法を提案した．

参考文献

[1] Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. *SIGPLAN Not.*, Vol. 25, No. 6, pp. 246–256, jun 1990.
[2] 権藤克彦, 川島勇人, 今泉貴史. TBCppA: 追跡子を用いた C 前処理系解析器. コンピュータ ソフトウェア, Vol. 25, No. 1, pp. 105–123, 2008.