イベント処理を考慮した正確かつ高速なデータフロー解析

高野 健太1　　荒堀 喜貴 1　　権藤 克彦 1

JavaScript のデータフロー解析を難しくする要因の 1 つに非同期処理として実行されるイベント処理が存在する。先行研究における非同期処理のモデル化は不正確であり、かつ全ての実行順序を考慮したモデル上では解析自体がスケールしないという課題が存在する。本研究ではこの非同期処理について、データフローの簡約を行い解析することで精度を保ったまま高速に解析できる手法を提案する。また、提案手法を利用しテイント解析を行うことで有用性を示す。

1. はじめに

近年、JavaScript はブラウザや Web サーバー等で広く使われるようになっている。一方で、JavaScript のコードは XSS や SQL インジェクションといった脆弱性を抱えることが多く、このようなバグを検出するための解析ツールが必要とされている。しかし JavaScript は動的言語であり、プロパティの参照・代入や動的型付け等の特徴により解析が難しい。特に静的解析においては、プログラムの振る舞いを正確に解析するのは難しく、実際に実行されない経路も含まれるため誤検出が多くなる、解析対象の実行経路が膨大で解析がスケールしなくなるという課題が存在する。

JavaScript の解析を難しくする要因の 1 つにイベント処理がある。イベント処理は非決定的な順序で実行される処理であるため、全ての実行経路を考慮すると解析がスケールせず、逆に実行フローを考慮せずに解析すると、実行されない経路が多く含まれてしまい解析精度が低下し解析時間が増大する。

本研究では、このイベント処理を考慮した上で解析を行うために、イベント順序ベースの簡約と操作類似ベースの簡約の 2 つの手法を提案する。これらの簡約により、既存手法に比べてより正確かつスケールする解析を行うことを目的とする。

2. 関連研究

既存の静的なデータフロー解析手法として WALA[1]や TAJS[2]、SAFE[3]が存在する。しかし、イベント処理に関しては不正確であり、イベントの発火順序を無視した完全に非決定的な制御フローを仮定して解析するため、実際に実行されない経路による誤検出が多い。

イベント処理の静的解析に関連する既存手法として Event-based Call Graph[4]が存在する。この手法は、イベントの発火順序をある程度考慮しているものの不正確であり誤検出が発生する。また、非同期処理の実行順序を非決定的に解析しているため解析自体がスケールしない。

3. Motivating Example

例えば、図 1 のプログラムを考える。このプログラムの write 関数では、条件式の writable の値によって片方のイベントが発火する。このプログラムを単純にデータフロー解析した場合、発火するイベントが判断できないため、両方のイベントが発火すると仮定して解析を行うことになる。その結果、実際には存在しないイベントの制御フローが含まれてしまい、解析が不正確でスケールしなくなる。この問題に対し、writable の値を解析し、その値によって発火するイベントを絞り込むことで発火するイベントをより正確に解析できると考えられる。

また、もう 1 つの例として図 2 のプログラムを考える。この例では、複数のイベント処理が同時に発火され、その callback がそれぞれ順番に実行される。このプログラムについてデータフロー解析した場合、図 2 中の A,B,C の発火順序を全て区別する解析ではスケールせず、発火順序を全てマージして

1 東京工業大学 情報理工学院

```
Stream.prototype.write = function(data) {
  if(!this.writable) {
    this.emit('error', err);
    return false;
  }
  this.emit('data', data);
  return !this.paused;
}
```

図 1: 条件式によるイベント処理の制御例

```
fs.readFile(..., (er, data)=> { // A
  transporter.sendMail(..., (er, info)=> { // B
  });
});
fs.readFile(..., (er, data)=> { // C
  res.end(data);
});
```

図 2: 複数のイベント処理が同時に発火する例

解析すると、図 2 中の B→C→A のデータフローなど、複数の制御フローの組合せによって解析が不正確になる。そのため、非同期処理の発火順序を保持した上でスケールする解析を行う必要がある。

4. 提案手法

本手法は Event-based Call Graph[4]をベースとした手法でデータフロー解析を行う。イベント処理の実行順序を考慮しつつ大規模な JavaScript コードにもスケールする解析の確立を目指す。この目的を達成するために、イベント順序ベースの簡約と類似操作ベースの簡約の 2 つの手法を提案する。

1 つ目のイベント順序ベースの簡約では、実行時には発生しないイベント処理の順序を枝刈りしてデータフローの候補を絞ることでより誤検出が少ないスケーラブルな解析を実現する。具体的には、イベント発火に影響する分岐の条件式について flow-sensitive に値解析を行い、発火し得るかを判断することで実行不可経路を削る。今回は複雑な条件式を考慮せず、単純な値解析が可能な式についてのみ考慮する。

2 つ目の類似操作ベースの簡約では、他の非同期処理を考慮した上で、類似したデータフローの計算を 1 つにまとめることで、よりスケールする解析を行う。全ての実行経路をそのままデータフロー解析を行うと解析がスケールせず、逆にフローを要約してから解析を行うと実際には起こり得ないデータフローの経路が多く含まれ解析が不正確になってしまう。そこで、非同期処理の順序が変化しても変化しない変数間の伝搬関係を考慮し、変数の伝搬関係で要約を行う。この解析を行う際は、既存の解析手法である Sparse Analysis[5]を利用する。

5. 実験計画

提案手法は、既存ツールの WALA[1]をベースに実装している。このツールを用いて実際の Node.js プログラムに対してテイント解析を行い、2 つの簡約を行わなかった場合の解析と比較することで有効性を示す。評価指標は、テイント解析における誤検出や検出漏れといった解析精度、解析に要した時間とする。

6. 現状

現状では Node.js 内のコードを簡単に調査し、変数によってイベント処理の制御を行っているパターンが複数存在することを確認している。そして、現在はこれらの変数による制御を解析することによる有効性を確認するための実装を進めている。同時に、提案手法のデータフロー解析の実装も進めており、これらの実装が終わり次第、評価実験を行う。

参 考 文 献

[1] IBM Research. T.J. Watson Libraries for Analysis (WALA). http://wala.sf.net. Accessed November, 25, 2017.

[2] Jensen, Simon Holm, Anders Møller, and Peter Thiemann. "Type Analysis for JavaScript." In *SAS*, vol. 9, pp. 238-255. 2009.

[3] Lee, Hongki, Sooncheol Won, Joonho Jin, Junhee Cho, and Sukyoung Ryu. "SAFE: Formal specification and implementation of a scalable analysis framework for ECMAScript." In *International Workshop on Foundations of Object-Oriented Languages (FOOL)*, vol. 10. 2012.

[4] Madsen, Magnus, Frank Tip, and Ondřej Lhoták. "Static analysis of event-driven Node. js JavaScript applications." In *ACM SIGPLAN Notices*, vol. 50, no. 10, pp. 505-519. ACM, 2015.

[5] Oh, Hakjoo, Kihong Heo, Wonchan Lee, Woosuk Lee, Daejun Park, Jeehoon Kang, and Kwangkeun Yi. "Global sparse analysis framework." *ACM Transactions on Programming Languages and Systems (TOPLAS)* 36, no. 3 (2014): 8.