B-017

デバッグ支援を目的とした大域的動的依存性解析の効率化
Toward Practical Implementation of Wide-Range Dynamic Dependency Analyses for Debugging Support

楠和馬 † 久米出 ‡ 波多野賢治 †
Kazuma Kusu Izuru Kume Kenji Hatano

1. はじめに

ソフトウェア開発において，ライブラリやアプリケーションフレームワークを用いた開発は開発効率の向上やプログラムの品質の維持につながるが，プログラムの挙動が複雑化する問題が生じる．また，ライブラリやアプリケーションフレームワークの利用方法に関する説明を記述したマニュアルの品質は保障されていないため，それらを用いた実用的なプログラムに対するデバッグは困難である．そのため，デバッグ手法の一つである動的依存性解析は，プログラム実行時のプログラムの挙動を解析可能であるため，プログラムに生じたエラー箇所の特定やソフトウェアアーキテクチャのようなプログラムに対する理解の向上に大きく貢献するとされている．

動的依存性解析手法には動的スライス解析やトレース解析が存在し，それぞれ様々な解析目的を持ったプログラムが提案されている．動的スライス解析では一般的に解析の対象となる変数を設定し，その変数が依存しているプログラムのステートメントを導出するため，動的スライス解析の解析対象となるプログラム領域は局所的である [1][4]．したがって，動的スライス解析は解析に多大な時間を要することなく解析を行うことができるため，より効率的かつ正確な依存関係の導出が求められている．そのため，動的スライス解析に関する研究では解析の処理効率に関する評価が行われている [2]．

一方で，トレース解析はプログラムの実行過程を全て記録したデータであるトレースを参照することでプログラムを大域的に解析することが可能なため，前述したエラー箇所の特定やプログラムに対する理解への貢献が動的スライス解析よりも大きい．しかし，実用的なプログラムの実行によって生成されるトレースは膨大かつ複雑なデータになるため効率的な解析が困難になる．その上，そのようなトレースに対して大域的な解析を行う場合に多大な解析時間を要することになるため，トレース解析に関する研究においては解析の効率性を考慮したトレースの処理方法に関しては言及されていない．

ソフトウェア開発においてトレース解析は一連のデバッグ作業の中で利用されることを想定すると，トレース解析のような大域的な動的依存性解析に対しても処理の効率性を考慮する必要があると考えられる．

そこで，本研究ではトレース解析のような大域的な動的依存性解析をより実用的にするために，動的依存性解析処理の効率化を考慮した解析基盤を構築する．また，構築した解析基盤において処理の効率性を損なう要因を明確化し，その要因を解消するための改善策を考案しその効果に関する考察を行う．

†同志社大学大学院
‡奈良先端科学技術大学院大学

2. 動的依存性解析環境の構築

トレースは実行過程を記録するプログラムが実用的であるほどデータ量は膨大かつ複雑なデータとなる．したがって，効率的な動的依存性解析を実現するためには膨大で複雑なトレースにたいする解析も効率的に解析可能にする環境が要求される．

本節では，どのようなトレース解析手法でも適用可能な動的依存性解析環境の構築に必要なトレース生成の手法と，トレースのデータ構造およびトレース解析手法の特徴に着目したトレース解析基盤の構築について説明する．

2.1. トレース生成

トレース解析はトレースに記録されている内容に従って解析を行うため，トレースに含まれるデータの量が少なければそのトレースに対して解析手法が限られる．そのため，本研究ではプログラムの実行過程が詳細に記録されており，各種命令や配列やクラス変数，クラスインスタンス変数のことを指すオブジェクト間の依存関係も記録可能なトレース生成手法を用いる [3]．

文献 [3] の手法ではプログラム実行時の挙動を追跡可能にするために，命令の呼出しや値・オブジェクトの生成などが詳細に記録されている．以下に文献 [3] の手法で生成されるトレースに記録されているデータを提示する．

1. メソッド呼出し構造
2. 条件分岐
3. 局所変数の代入
4. 配列やフィールドへの値の代入や参照
5. 演算
6. instanceof 演算子
7. クラスインスタンスや配列の生成
8. Throw と Catch
9. 定数値の生成
10. 不可視な命令の呼出し

以上の他にもトレースの要素として存在しているが，それら要素にはプログラム行番号やスレッド番号，クラス名などといった属性情報が付随している．また，命令および値の間には参照関係や制御関係といったプログラム実行中に発生する依存関係もトレースに記録されている [3]．

一方，トレース解析の効率的な処理を実現するためにはトレースのデータ構造やトレース解析の方法について考慮する必要がある．

まず，トレースのデータ構造については，トレース解析手法に関する研究においてトレースは一般的に節点や辺から成るグラフ構造で扱われており，グラフ構造の節点はデータの中で実体を持つデータを表現することができ，辺は実体間の関係性を表現することが可能である．同様に本研究で用いるトレースにおいても，命令や値を節点，それらの間に依存関係を辺としたグラフ構造で表現することができるが，節点や辺には属性情報が付随しているためそれを保持できる必要がある．また，トレースにおいて依存関係には方向性を持つため，依存関係は有向辺で表現する必要がある．さらに，命令や値，依存関係は多種存在することを考慮し，グラフ上で判別することを可能にする必要がある．

したがって，トレースの解析を行う際には，オブジェクトの状態の変遷やプログラムのエラーを追跡など，命令や値の依存関係を辿っていくことになるため，トレース解析では依存関係を辿る処理が頻繁に行われることが想定され，グラフの関係性を辿る処理の効率化が要求される．

2.2. グラフデータベース Neo4j

本研究ではグラフの関係性を辿ることによりトレース解析を行うことに着目し，グラフの辺を走査する処理に最適化されたグラフデータベースをトレースの解析基盤として用いることにした．グラフデータベースには既に様々なプログラムが存在するが，本研究ではトレースを格納するのに適しているプロパティグラフモデルを採用していることや，学術的な利用が多い点や開発の活発度が高い点から Neo4j [§] を採用することにした．

グラフデータベース Neo4j が採用しているプロパティグラフモデルは次のような構成要素となっており，図 1 のような構造をしている．

節点: グラフデータの実体を表現する要素で，プロパティが付随している．

有向辺: グラフデータの関係性を表現する要素で，プロパティが付随している．

プロパティ: key-value 形式で属性情報を格納することができる．

以上のように，節点および有効辺には属性情報を登録することができるプロパティが付随しているため，命令や値に付随しているプログラムに関する属性情報をそのままプロパティに登録することができる．また，Neo4j では節点に対してラベルという節点をカテゴライズするための機能が備わっている．さらに，一つの有向辺に対しては一つのタイプと呼ばれる関係性を設定することが可能であり，グラフデータの中に複数の関係性を定義することが可能である．

プロパティグラフモデルは基本的にリレーショナルデータベースのようなスキーマを設計する必要がないスキーマレスであるため，トレースにさらなる属性情報が加わる場合にも柔軟に対応することができるためトレースが持つ拡張性を損なわずにトレースをグラフとして表現することが可能である．

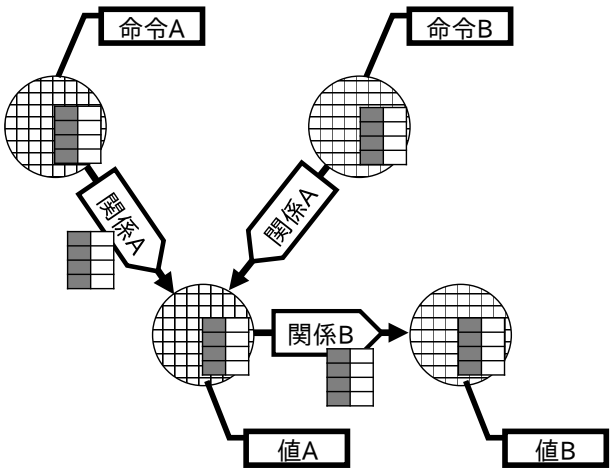

図 1: プロパティグラフモデルの構造

Neo4j ではグラフデータベースに格納されたグラフデータを問い合わせるための標準言語として Cypher が提供されている．Cypher は SQL ライクなクエリ言語であり，アスキー文字によってグラフパタンを表現することによって，データベースから指定されたパタンに合うサブグラフを問合せすることができる．

本研究では Neo4j 組み込みの Java API を用いて，Cypher によるクエリや Java API で提供されている機能を用いてグラフを走査することによってトレース解析を実装する．

3. 節点プロパティの分割による動的依存性解析の効率化

2.2 節では文献 [3] の手法により生成されたトレースの構造に手を加えることなく，そのままグラフにモデリングしグラフデータベースに格納する方法について説明した．しかし，この場合は解析時に不要なデータまで計算機の主記憶に読み込まれることになり，解析が非効率になる可能性がある．

そこで本節では，動的依存性解析の効率化に繋げるためにトレースの再モデリングを行い，計算機の主記憶に読み込まれるデータ量の削減法について説明する．

3.1. グラフ走査におけるグラフデータベースの仕様

グラフデータベースに格納されているデータに対して問い合わせを行う際には，まず，グラフ走査の開始となる節点データを問い合わせてから， それらが計算機の主記憶に読み込まれる．次に，グラフ走査の開始する節点データからクエリで指定されている関係性を辿ることによって次の節点データを取得し，計算機の主記憶にグラフを展開していく．つまり，既に計算機の主記憶に読み込まれている節点が走査の対象になった場合は，その節点は主記憶に読み込まれないが，未だ主記憶上に読み込まれていない節点は次々と主記憶に読み込まれ，グラフが展開されていく．

走査すべき全節点を計算機の主記憶に展開する際は，

[§] http://neo4j.com/ 2016 年 6 月 27 日閲覧

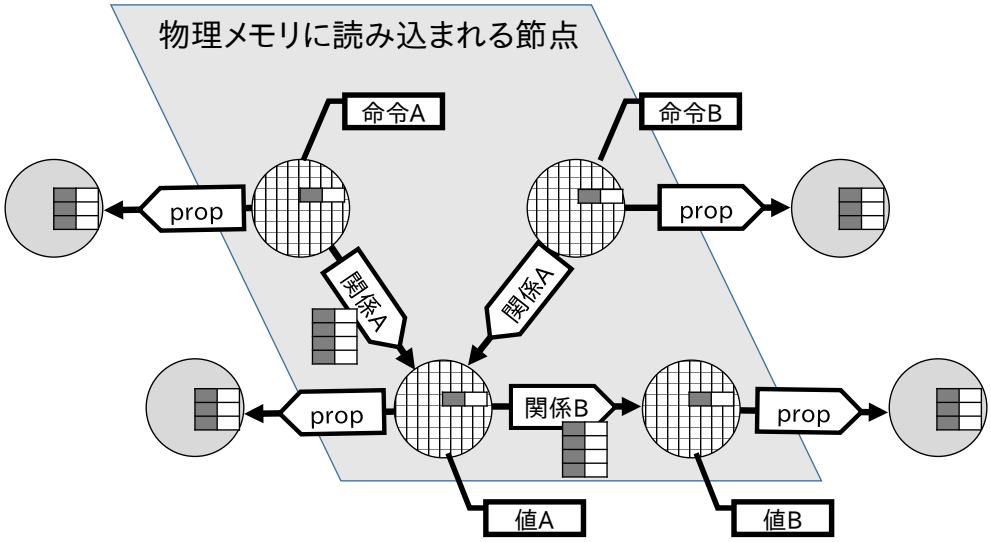

図 2: 節点プロパティの分割

当然，その節点に付随するプロパティも主記憶に読み込まれる．そのため，グラフ走査がデータベースに格納されている全データに対して行われる際には，ほとんどすべてのデータが計算機の主記憶に読み込まれることになるため，巨大なトレースを扱う場合には計算機の主記憶が枯渇する可能性があり，グラフ走査が効率的に行われなくなる懸念が生じる

3.2. 節点プロパティの分割

3.1 節で挙げた問題では解析対象となる節点と一緒にその節点のプロパティも物理メモリに読み込まれることが問題であった．一方で，グラフ走査の対象とならない節点データは物理メモリに読み込まれておらず，解析に必要ない場合はグラフ走査が終了するまで読み込まれることはない．

したがって，節点データに付随しているプロパティを保持するためだけの別の節点を作成することによって，本体の節点データが主記憶に読み込まれたとしてもそのプロパティは読み込まれないように対処することができると考えられる (図 2)．以上の対処により，本体の節点データはどの種類の命令もしくは値であるか判別可能なラベルや仕様上最低限必要な節点 ID のみが解析対象となった際に読み込まれるだけで，その節点に付随していたプロパティのデータサイズ分だけ解析処理に使用する主記憶容量を節約することを可能にすることができる..

解析に不要な各節点のプロパティを別節点に退避させたことで，節点データのプロパティを利用する際にはプロパティを保持する節点を辺を辿ることによって取得する必要が生じるが，プロパティのデータをほとんど必要としない解析においてメモリ効率が良いことが考えられる．

4. 比較実験内容と考察

本節ではトレースに対して大域的に動的依存性の解析を行う手法と公開されているプログラムの実行過程を記録したトレースを用いて，グラフデータベースへのトレースの格納方法を変更したことによる節点プロパティのメモリへの読込み削減量を見積もり，それに対して考察を行う．

4.1. 大域的な動的依存性解析の実装

本研究で構築したトレース解析基盤において実行する大域的なトレース解析には文献 [3] で提案されている Outdated State 兆候解析を用いる．兆候解析とはトレース解析に包含される解析の一つとして提案されており，兆候解析を行うことによってプログラムにおけるアンチパタンとされているプログラムの記述の候補を取得することが可能である．Outdated State 兆候解析はオブジェクトの古くなった状態と更新されたオブジェクトの状態を二つ合わせて利用している命令を解析する．Outdated State 兆候の発生自体はプログラムによく内在しているパタンであり，必ずしもプログラムのエラーの原因となるわけではない．しかし，Outdated State 兆候が発生しているプログラム箇所が原因でのちにバグを引き起こす原因となる可能性があるため，Outdated State 兆候が皆無な状態の方が望ましいといえる．

Outdated State 解析ではトレースに記録されている命令を実行順に全て調査するようなアルゴリズムであるため，命令や値の節点は最低一通り物理メモリに読み込まれることが想定される．

4.2. 実験に用いるプログラム

4.1 節の解析に利用するトレースには Graph Editing Framework のデモプログラム ¶ (GEFDemo プログラム) の実行によって生成されたトレースを用いる．GEFDemo プログラムはアプリケーションフレームワークを用いた簡易な UML エディタプログラムであるが，GEFDemo プログラムにはプログラムの実装に欠陥があり，特定の操作を行うとプログラムが不正な挙動を

¶http://gefdemo.stage.tigris.org/ 2016 年 6 月 27 日閲覧

起こすことが知られている (図 3)[3].

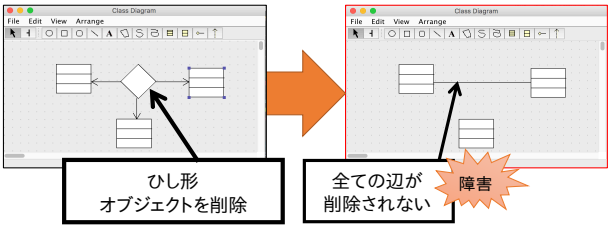

図 3: GEFDemo プログラムに発生する障害

本実験において利用するトレースは図 3 のように，以下の手順で故意に例外を発生させたプログラムの実行過程を記録している．

1. エディタ上にクラスを三つ作成する．
2. 一つのクラスから他のクラスに対して関連を作成する．
3. 関連を作成していないクラスから関連の線に対して関連を作成する．
4. 3 項関連を意味するひし形オブジェクトが発生する．
5. ひし形オブジェクトを削除する．

以上の手順を行うことにより生成された GEFDemo プログラムのトレースをグラフデータベースに格納した結果，節点や辺，プロパティなどの数は表 1 の通りである．

表 1: GEFDemo プログラムのトレース要素の数

要素	個数	データサイズ (MiB)
節点	503760	8.00
辺	4609781	152.00
プロパティ	1403937	56.00
タイプ	43	–
ラベル	45	–

4.3. 考察

表 1 において，通常時のプロパティのデータサイズは 56 MiB であり，最低限必要な節点 ID 以外のプロパティを分割することによって，Outdated State 解析時にはトータルとして約 50.45 MiB のプロパティを物理メモリへの読込みの削減することが可能になることが見積もることができる．上記の見積もりでは，命令と値の全ての節点を一通り読み込んだ場合の見積もりであるため，重複して読見込まれている節点については考慮に入れた見積もりになっていないため，実際はより多くのデータサイズの物理メモリへの読込みを削減できていることが考えられる．

5. おわりに

本研究ではトレース解析においてプログラムの命令や値の間に存在する依存関係を辿ることにより解析を行っていることに着目し，グラフ処理に最適化されたグラフデータベースを用いてトレース解析基盤を構築した．トレースをグラフデータベースで採用されているプロパティグラフモデルに適用することによって，トレース自体が持つ拡張性を損なうことなくデータベースに格納することができた．

また，グラフデータベースのグラフ走査の際に，走査候補である節点とそのプロパティも合わせて物理メモリに読み込まれることに着目し，節点プロパティの分割化を施すことによってトレース解析の効率化に貢献することを試みた．

GEFDemo プログラムの実行過程を記録したトレースに対して，Outdated State 兆候解析を実行した場合の節点プロパティの分割化によってプロパティのメモリへの読込みを軽減可能な見積もりとしては最低約 50.45 MiB 以上が期待できる．

謝辞

本研究は同志社大学ハリス理化学研究所研究助成事業および栢森情報科学振興財団研究研究助成事業，人工知能研究振興財団研究助成事業，日本学術振興会科学研究費助成事業 15K12009 の助成を受けて遂行された．

参考文献

[1] Kelly Androutsopoulos, David Clark, Mark Harman, Jens Krinke, and Laurence Tratt. State-based model slicing: A survey. *ACM Comput. Surv.*, Vol. 45, No. 4, pp. 53:1–53:36, August 2013.

[2] Yu Kashima, Takashi Ishio, and Katsuro Inoue. Comparison of backward slicing techniques for java. *IEICE Transactions on Information and Systems*, Vol. 98, No. 1, pp. 119–130, 2015.

[3] Izuru Kume, Masahide Nakamura, Naoya Nitta, and Etsuya Shibayama. A Case Study of Dynamic Analysis to Locate Unexpected Side Effects Inside of Frameworks. *International Journal of Software Innovation*, Vol. 3, No. 3, pp. 26–40, July 2015.

[4] Xiangyu Zhang, Haifeng He, Neelam Gupta, and Rajiv Gupta. Experimental evaluation of using dynamic slices for fault location. In *Proceedings of the Sixth International Symposium on Automated Analysis-driven Debugging*, AADEBUG'05, pp. 33–42, New York, NY, USA, 2005. ACM.