プログラムの部分的変更にともなう依存関係グラフの更新手法

An Update Method of Dependence Graph with Program Modification

高田 智規　鍛治 武志[†] 佐藤 慎一[††] 井上 克郎
Tomonori Takada　Takeshi Kaji　Shin'ichi Sato　Katsuro Inoue

大阪大学大学院 基礎工学研究科
Graduate School of Engineering Science, Osaka University
†現在，奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science,
Nara Institute of Science and Technology
††現在，NTT データ通信株式会社
NTT DATA Communications Systems Corporation

概 要

依存関係グラフ（依存グラフ）とは，プログラム内の文の間の依存関係を表す有向グラフであり，依存グラフを利用した様々なシステムが作られている．これまでのシステムでは，プログラムの部分的な変更に伴い依存グラフの一部のみが変更されるような場合でも，変更されたプログラム全体を解析し依存グラフを作り直していた，

プログラムの部分的な変更が行われた時に，関連する部分のみについて依存グラフを効率良く再構築する手法を提案する．提案した手法を実際に我々が作成しているデバッグ支援システムに組み込んで有効性を確認した．

1 まえがき

プログラム依存グラフ（Program Dependence Graph, PDG）[4] は，プログラムの各文における変数間の依存関係を表す有向グラフである．PDG の各節点はプログラム中の各文・条件式を表し，有向辺は依存関係（データ依存・制御依存）を表す．PDG 上の有向辺を辿ることにより，ある節点上の変数と依存関係のある節点の集合（プログラムスライス [2]）を抽出することができる．

これまでに，プログラムスライスを抽出しデバッグを効率良く進めるためのシステム [3] を開発した．このシステムでは，プログラムに変更を加えるたびに PDG を作り直しており，それに多くの時間を費していた．しかし，PDG のうちプログラムに変更のあった箇所に関する部分だけを更新することができれば，PDG の作り直しに要する時間が短縮され，大幅な作業効率の向上が期待できる．

プログラム内の部分的な変更がその他の文に与える影響を調べるアルゴリズムは既に提案されているが [1]，このアルゴリズムは PDG の利用を考えていない，関数間にまたがる文の依存関係について考慮されていない，などの問題がある．

そこで，プログラムの部分的変更が行われた時に効率良く PDG を再計算する手法を提案し，実際に

既存のシステムに実装しその有効性を確かめた．

2 依存グラフ

2.1 到達定義

文 s における変数 x（の値）の定義が文 t に到達するとは，文 s が変数 x を定義し，かつ，s から t に至るパス $s, u_1, u_2, \ldots, u_k, t$ が存在して，$u_1, u_2, \ldots, u_k$ では変数 x を定義しない場合を言う．

文 n において定義された変数 vが文 s に到達する場合，文 s には到達定義 $\langle n, v\rangle$ が存在すると言う．到達定義集合とは，各節点における $\langle n, v\rangle$ の組で表される到達定義の集合である．

2.2 依存関係

依存関係として以下の二種類を考える．

1. **データ依存関係 (Data Dependence, DD)**
   文 s において変数 wが定義されており，その定義が変数 wを使用している文 t に到達している場合，文 s から文 t に対してデータ依存関係があると言う．
2. **制御依存関係 (Control Dependence, CD)**
   文 s が分岐文または繰り返し文であり，文 s の条件判定の結果が文 t の実行に直接影響を与える時，文 s から文 t への制御依存関係があると言う．

2.3 プログラム依存グラフ

プログラム依存グラフ（Program Dependence Graph，PDG）とは，プログラム内の各命令間の依存関係を表す有向グラフである．節点には，プログラム内の各文・または条件式（分岐文・繰り返し文の条件判定部分）を表すもの，及び手続き境界をこえる依存関係の解析などに用いる特殊節点がある．

辺は 2 つの節点の間の依存関係を表し，その種類には依存関係に応じて以下の二種類がある．

- データ依存辺（ $s \xrightarrow{w} t$ と表す）
- 制御依存辺（ $s \dashrightarrow t$ と表す）

2.4 フローグラフ

フローグラフとは，プログラムにおける実行可能な命令の実行順序を表した有向グラフである．節点は，プログラム中の各文または条件式に対応しているもの，及び二分された制御を合流させるための合流節点がある．辺（他のグラフの辺と区別するため，フロー辺と呼ぶ事がある）は文間の制御の流れを示す．文 s から文 t に実行が移る可能性がある時，節点 s から節点 t の辺 $s \rightarrow t$ と表す．

3 グラフの更新手法

3.1 依存グラフ

本手法では 図 1 のような，PDG にフローグラフを組み合わせた依存グラフを用いる．この依存グラフは直観的には PDG とフローグラフの和集合であり，節点には文・条件式を表す節点，特殊節点，合流節点があり，辺にはフロー辺，データ依存辺，制御依存辺がある．

また，各節点で，実行直前・実行直後の到達定義集合を保持し，それぞれ ERD（Entry Reaching Definition），XRD（eXit Reaching Definition）と呼ぶ．

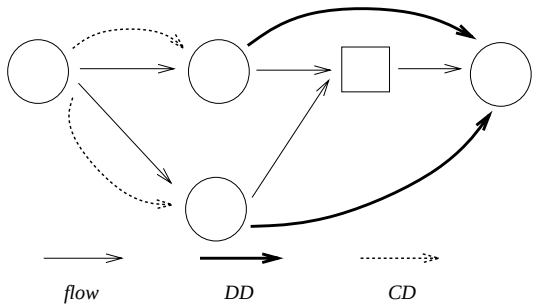

図 1　依存グラフの例

3.2 プログラムの部分的変更

プログラムの部分的変更とは，文の**削除**，または文の**挿入**を言う．文の変更は削除と挿入で実現できるため，ここでは考えない．

挿入，及び削除を行う対象としては，依存グラフの節点一つの場合だけを考える．

3.3 前節点，現節点，後節点

提案するアルゴリズムは依存グラフの節点ごとに処理を行う．アルゴリズムのある時点において節点 a に関して処理を行っており，フロー辺 $b \rightarrow a$ ，$a \rightarrow c$ が存在する時，節点 a, b, c のことをそれぞれ現節点・前節点・後節点と呼ぶ．

3.4 前定義節点

現節点 a において変数 wが定義されており，節点 b においても変数 wが定義されているとする．$\langle b, w\rangle$ が現節点 a の ERD に含まれている（節点 b における wの定義が節点 a の実行直前に到達している）場合，節点 b を現節点の変数 wに関する前定義節点と呼ぶ．

3.5 削除アルゴリズム

ある文を削除する事を考え，その文に対応する依存グラフ上の節点を便宜上 “削除節点” と呼ぶ．

3.5.1 概要

削除される文において変数が定義されていれば，それ以降の文，つまり削除節点からフロー辺を順方向に辿った先の節点における到達定義に影響を与える．そのため，フロー辺の先の節点での到達定義を順次再計算する．到達定義の変更が無くなった時点で終了する．

3.5.2 アルゴリズム

削除節点が分岐文あるいは繰り返し文の場合は，その節点から制御依存辺が出ている節点全てについて以下を繰り返す．

1. 削除節点において定義される変数がない場合，5. に進む．
2. 削除節点において定義される変数がある場合，削除節点の XRD を削除節点の ERD に置き換える．
3. 削除節点からフロー辺を順方向に辿る．
   (a) この時，現在辿っている節点を現節点とする．
   (b) 前節点の XRD を求める．この時，前節点が複数あれば，全ての前節点の XRD の和集合を求める．
   (c) 前節点の XRD（の和集合）と現節点の ERD を比較する．もし，その結果が等しければフロー辺を辿ることを終了し，異なれば前節点の XRD（の和集合）を現節点の ERD として，現節点の XRD を再計算する．
   (d) 現節点からフロー辺を順方向に辿り（たどるフロー辺がない場合 5. へ），3a. からまた繰り返す．現節点からフロー辺が複数本出ていれば，そのそれぞれについて，3a. からの処理を繰り返す．
4. 削除節点 s で変数 wを定義し $s \xrightarrow{w} t$ という関係の節点 t がある時，前定義節点 u を求め，データ依存辺 $u \xrightarrow{w} t$ を引く．
5. 削除節点の前節点と後節点をフロー辺で結ぶ．
6. 削除節点に関係する辺（データ依存辺，制御依存辺，フロー辺）を全て削除し，最後に削除節点自体を削除する．

ただし，関数（手続き）の境界を越えた依存関係は特殊節点を中継したデータ依存関係がある．そのため，関数（手続き）呼び出し文を含む文を削除する場合，削除する文に対応する依存グラフ上の節点に関係する辺（節点に入ってくるあるいは出ていく，データ依存，制御依存辺，フロー辺）だけでなく，関数（手続き）の特殊節点から直接出ていく辺あるいは特殊節点に直接入ってくる辺も削除する．

3.6 挿入アルゴリズム

ある文を挿入する場合を考え，その文に対応する依存グラフ上の節点を便宜上 “ 挿入節点” と呼ぶ．

3.6.1 概要

挿入する文において変数が使用されていれば，その変数を定義した節点から挿入節点へのデータ依存関係が生じる．

また，挿入する文において変数が定義されていれば，それ以降の文，つまり挿入節点からフロー辺を順方向に辿った先の節点における到達定義に影響を与える．そのため，フロー辺の先の節点での到達定義を順次再計算し，それにデータ依存関係辺を更新する．到達定義への影響が無くなった時点で終了する．

3.6.2 アルゴリズム

1. 挿入節点自体を作成する．この節点の ERD は前節点の XRD と等しくする．
2. 前節点から挿入節点へ，また，挿入節点から後節点へフロー辺を引き，前節点と後節点の間のフロー辺を削除する．
3. 挿入節点において参照する変数があれば，挿入節点の到達定義集合を調べる．もし参照する変数に関する前定義節点があれば，その前定義節点から挿入節点に対してデータ依存辺を引く．
4. - 挿入節点において定義する変数がなければ，終了．
   - 挿入節点において定義する変数があれば，この節点の XRD を計算する．

5. 挿入節点からフロー辺を順方向に辿る.
   (a) この時，現在辿っている節点を現節点とする.
   (b) 前節点の XRD（の和集合）を求める.
   (c) 前節点の XRD（の和集合）と現節点の ERD を比較する．もし，等しければフロー辺を辿ることを終了する．異なれば現節点の ERD を前節点の XRD（の和集合）に置き換えて 5d. へ進む.
   (d) 現節点へのデータ依存辺を全て削除し，ERD 合をもとにして，現節点へのデータ依存辺を引き直す.
   (e) 現在の ERD をもとにして，現節点の XRD を再計算する.
   (f) 現節点からフロー辺を順方向に辿り（辿る辺がない場合終了する），5a. からまた繰り返す．現節点からフロー辺が複数本出ていれば，そのそれぞれについて，5a. からの処理を繰り返す.

ただし，分岐文・繰り返し文を挿入する場合は，まず，条件式部分を挿入し，その後実行部分を挿入する．条件式部分を挿入した時は，合流節点を条件式の後節点として挿入する．また，実行部分を挿入した時は条件式部分からの制御依存辺を引く.

4 実装

プログラムスライスを利用したデバッグ支援システム [3] に，本稿で提案した依存グラフ更新アルゴリズムを実際に組み込み，正しく動作することを確認した．このデバッグ支援システムが対象としている言語は Pascal のサブセット†1である.

プログラムの一部に変更があった場合を考える．プログラム全体を解析し PDG を作り直す従来の方法，変更された部分に関連する箇所のみを更新する今回提案した手法の実行時間を 表 1 に示す.

†1 入出力文，代入文，if 文，while 文，関数呼び出し文，複合文を持ち，スカラ型変数のみを扱う

表 1 PDG 作成・更新時間

行数	約 50 行	約 100 行	約 250 行
従来の方法	0.08 秒	0.26 秒	2.06 秒
今回の方法	0.01 秒	0.02 秒	0.02 秒

5 あとがき

プログラムの部分的変更が行なわれた際に，PDG を部分的に変更する手法を提案した．また，既存のシステムに組み込むことによりその有効性を確認した．本手法は，今回実際に組み入れたシステムに限らず，PDG を利用しその一部が頻繁に変更される可能性のあるシステム使用時の作業効率を向上させることが期待される.

しかし，変更の種類やその量によって PDG の再構築の手間が異なるため，多くの変更が行なわれる場合は最初から PDG を作り直した方が効率的な場合もあると考えられる．そのような場合を判断する，またはユーザがどちらかを選べるようにすることが実用化する際には必要であろう.

参考文献

[1] Barbara G. Ryder and Marvin C. Pall, Incremental Data-Flow Analysis Algorithm, *ACM Transactions on Programming Languages and Systems*, vol. 10, 1988, pp. 1–50.

[2] Mark Weiser, Program Slicing, In *Proceedings of the Fifth International Conference on Software Engineering*, pp. 439–449, San Diego, CA, September 1981.

[3] 佐藤 慎一, 飯田 元, 井上 克郎, プログラムの依存関係解析に基づくデバッグ支援システムの試作, 情報処理学会論文誌, vol. 37, April 1996,

[4] Susan Horwitz and Thomas Reps, The Use of Program Dependence Graphs in Software Engineering, In *Proceedings of the 14th International Conference on Software Engineering*, pp. 392–411, Dallas, Texas, 1992. Association for Computing Machinery.