システムソフトウェアと
オペレーティング・システム 67－15
(1994. 12. 9)

プログラム従属性理論に基づく並行処理ソフトウェアの統合的開発支援環境の構築

笠原義晃　　程京徳　　牛島和夫

九州大学工学部情報工学科

〒812 福岡市東区箱崎6-10-1

E-mail: {kasahara,cheng,ushijima}@csce.kyushu-u.ac.jp

あらまし

プログラム従属性の明示的表現は、プログラムの内容理解、テスト、デバッグ、保守、複雑さ計測といったソフトウェア開発支援におけるさまざまな応用がある。我々は既に並行プログラムにおけるプログラム従属性を明示的に表現できるプロセス従属ネットを提案した。これを基に、各種逐次・並列手続き型言語で書かれたプログラムにおけるプログラム従属性を統一的に表現することができる。この統一表現を用いて、逐次・並行処理ソフトウェア開発の各種支援を行うツールを開発し、それらを一つの環境として統合することにより、逐次・並行処理ソフトウェア開発のさまざまな場面を支援する統合的開発支援環境を構築することができる。

和文キーワード　プログラム従属性　並行処理ソフトウェア　プログラム従属グラフ
プロセス従属ネット　ソフトウェア開発　統合的支援環境

Constructing an Integrated Development Supporting Environment for Concurrent Software based on Program Dependence Theory

Yoshiaki KASAHARA　Jingde CHENG　Kazuo USHIJIMA

Department of Computer Science and Communication Engineering, Kyushu University

Hakozaki 6-10-1, Higashi-ku, Fukuoka 812, JAPAN

Abstract

Explicit representations of program dependences have many applications in software development activities including program understanding, testing, debugging, maintenance, and complexity measurement. We proposed the Process Dependence Net to explicitly represent basic program dependences in concurrent programs. It can be used as a unified representation of program dependences in programs written in various sequential/concurrent programming languages. Based on this representation we can implement various tools to support software development, and construct an integrated development supporting environment for concurrent software as well as sequential software.

英文 key words　program dependences　concurrent software　program dependence graph
process dependence net　software development　integrated supporting environment

1 はじめに

近年、マルチプロセッサシステムやLANの普及に伴なって、並行処理ソフトウェアに対する需要が増加している。並行処理システムにおいて中心的な役割を果たす並行プログラムの信頼性を高めることは重要な課題である。しかし、並行プログラムには多重の制御の流れとデータの流れが同時に存在しうるため、その実行時の振舞いには非予測性と非決定性がある。このため、並行プログラムの理解、テスト、デバッグ、保守などは非常に困難である。

現在、並行プログラムの信頼性を向上させるために、並行プログラムの開発を支援するための研究が行われている。これらの研究は開発過程のある一部に焦点を絞ったものが多く、並行プログラムの開発過程全体を一貫して支援するような統合的開発支援環境の構築に関する研究はほとんど行われていない。

プログラム従属性は、プログラム中の制御の流れとデータの流れによって決定される、プログラムの各文間に存在する従属関係である。逐次プログラムのプログラム従属性を明示的に表現したモデルとしてプログラム従属グラフがある[9][11]。これはもともと逐次プログラムの並列化のために考案されたものだが、逐次プログラム開発のさまざまな場面で有用な情報を持ち、逐次プログラムの並列化のみならず、テスト、デバッグ、保守などの様々な応用が考えられ、研究されている[11][14][15][16][17]。

並行プログラムのプログラム従属性を明示的に表現したモデルとしては、我々が提案したプロセス従属ネットがある[3][4][5]。逐次プログラム開発に対するプログラム従属グラフの応用を考えると、並行プログラムの開発の際にプロセス従属ネットを利用して同様の開発支援が行えると考えられる。しかし、並行プログラムの開発支援にプログラム従属性理論を適用する研究はまだあまり進んでいない。

本研究は、さまざまな手続き型言語で記述された逐次・並行プログラムを、統一的なプログラム従属性表現を用いて表現することにより、プログラム開発過程のさまざまな場面を支援する統合的な開発支援環境を開発することを目的とする。本論文では本研究の基礎となる逐次・並行プログラムのプログラム従属性とその明示的表現を紹介し、本研究の方向性と目標、及び実施計画について述べる。

以下、第2章では、プログラム従属性とプロセス従属ネットについて述べる。第3章では、現在我々が開発しているプログラム従属性理論に基づく統合的開発支援環境の開発計画とその有用性について述べる。第4章では、まとめと今後の課題について述べる。

2 プロセス従属ネットとその明示的表現

あるプログラムのソースを読み、その内容を理解し、デバッグや改良を行うときには、「この変数はどこで定義されているのか」「この変数は何に使われるのか」「この文はどういう条件で実行されるのか」といった情報を把握することが不可欠である。これらの情報は暗黙的にプログラムに含まれているものであり、プログラムを注意深く読み進めて行けば把握することが可能である。

変数の役割に関する情報は、プログラム中のデータの流れに関係している。また、ある文が実行されるかどうかといった情報は、プログラム中の制御の流れに関係している。

ある変数の値を定義している文は、そこでその変数の値を決定するために参照している他の変数の値を定義している文から影響を受ける。ある条件分岐文の選択枝となっている文はその条件分岐文における制御述語の評価に影響を受ける。このように、プログラム内のある文の実行によってその後に実行される別の文の実行がなんらかの影響を受けるとき、後者は前者に従属していると言う。プログラムにおける制御の流れとデータの流れによって決定される、プログラムの各文間に存在する従属関係のことをプログラム従属性と呼ぶ。

逐次プログラムにおける基本的なプログラム従属性には、制御の流れによって起こる制御従属性と、データの流れによって起こるデータ従属性がある。これら2種類の従属関係を使って逐次プログラムのプログラム従属性を明示的に表現するモデルとしてプログラム従属グラフが提案されている[9][11]。

一方、並行プログラムは複数の逐次プログラム(プロセス)が相互に同期・通信することによって一つのシステムを作っている。プログ

```
        1: with TEXT_IO, INTEGER_IO;
        2: use TEXT_IO, INTEGER_IO;
        3:
        4: procedure test is
        5:
        6:   task T;
        7:
  t1    8:   task body T is
  t2    9:     R:INTEGER := 1;
       10:     task FACT is
       11:       entry E1(I: in INTEGER);
       12:       entry E2(O: out INTEGER);
       13:     end FACT;
       14:     task PERCENT;
       15:
  p1   16:     task body PERCENT is
       17:       X, Y, P : INTEGER;
       18:     begin
  p2   19:       X := 5;
  p3   20:       Y := 10;
  p4   21:       if X <= 0 or Y <= 0 then
  p5   22:         PUT("MINUS_OPERATOR");
  p6   23:       elsif Y = 0 then
  p7   24:         PUT("ZERO_DIVIDE");
       25:       else
  p8   26:         P := (X/Y)*100;
  p9   27:         FACT.E1(P);
       28:       end if;
  p10  29:     end PERCENT;
       30:
  f1   31:     task body FACT is
       32:       N, F : INTEGER;
       33:     begin
  f2   34:       N := 10;
  f3   35:       F := 1;
       36:       while N /= 0
  f4   37:       loop
  f5   38:         F := F*N;
  f6   39:         N := N-1;
  f7   40:       end loop;
  f8   41:       select
  f9   42:         accept E1(I: in INTEGER) do
  f10  43:           F := F*I;
  f11  44:         end E1;
       45:       or
  f12  46:         accept E2(O: out INTEGER) do
  f13  47:           O := F;
  f14  48:         end E2;
       49:       end select;
  f15  50:     end FACT;
       51:
       52:   begin
  t3   53:     FACT.E2(R);
  t4   54:   end T;
       55:
       56: begin
  m1   57:   null;
  m2   58: end test;
```

図 1: プログラム例

ラム従属グラフでは、このようなプロセス間の相互作用を表現することができない。我々は、並行プログラムにおけるプログラム従属性を表現するために、3種類の基本的なプログラム従属性を新たに提案した。すなわち、プロセス間における制御の流れの相互作用による同期従属性、プロセス間のデータの流れによる通信従属性、そして並行プログラムの非決定的な動作による選択従属性である。そして、並行プログラムにおけるプログラム従

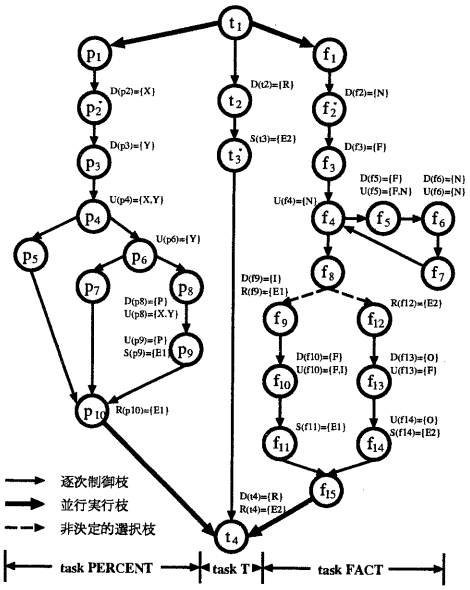

図 2: 定義使用ネット

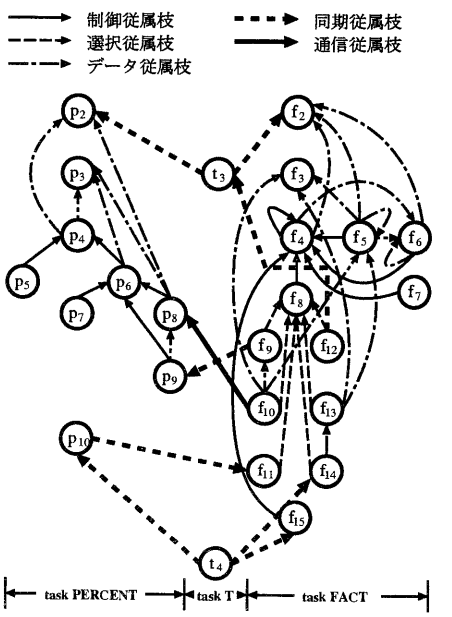

図 3: プロセス従属ネット

属性を明示的に表現するモデルとしてプロセス従属ネットを提案した[3][4][5]。

プロセス従属ネットは、並行プログラム中の

各単純文及び制御述語を節点とし、5種類の基本的な従属関係を表す5種類の異なる有向枝によって各文間に存在する各種従属関係を表現するラベル付き有向グラフである[3][4][5]。

プロセス従属ネットを対象プログラムから生成する際には、まず対象プログラムから定義使用ネットを取り出す。これは、プログラムの制御フローと変数の定義・使用の状況、および各プロセス間の同期・通信を表わした有向グラフである。プロセス従属ネットにおける5種類の従属関係は定義使用ネットに基づいて定義されており[5]、定義使用ネットを基にプロセス従属ネットを生成することができる[8][12]。

また、逐次プログラムのためのプログラム従属グラフは、プロセスが1つしかないプロセス従属ネットの特殊な形であるとみなすことができる。

図2に定義使用ネットの例を、図3にプロセス従属ネットの例を示す。これらの図は、図1に載せた3つのタスク、T、PERCENT、FACTを持つAdaプログラムに関するネットある。

3 統合的開発支援環境

プロセス従属ネットはソフトウェア開発のさまざまな場面を支援する応用が考えられ、これらをツールとして実装することにより、ソフトウェア開発支援ツール群を提供することができる。

プログラム従属性を基礎とした支援ツールは、対象プログラムのソースから従属性を抽出してモデルとして保持する必要がある。並行プログラムからのプロセス従属ネットの生成には$O(N^2)$(Nはプログラム中の文及び制御論理式の数)の計算量が必要である[12]。従って、1つの対象プログラムに対し複数のツールがそれぞれ個別にネットを生成するのでは効率が悪い。そこで、対象プログラムからプログラム従属性を抽出しプロセス従属ネットを生成する独立したツールを用意し、このツールの出力を開発支援ツール群が利用するように設計することにより、効率的に対象プログラムの従属性を利用することができる。また、ツールに対して命令を与えるインターフェイスを規定することにより、複数のツールをまとめて、ソフトウェア開発のさまざまな場面について統一的に支援を行える統合的開発支援環境を構築することができる。

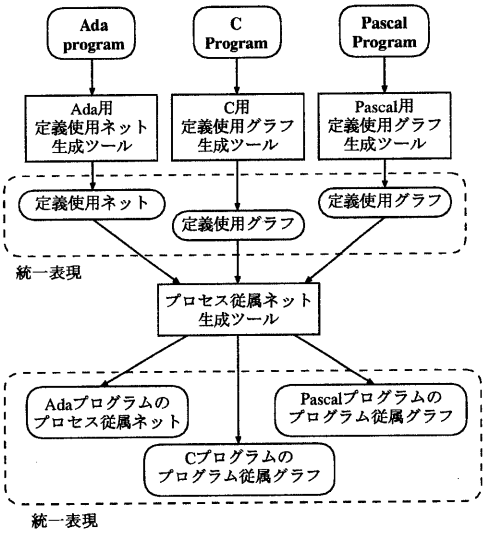

図 4: プロセス従属ネット生成ツール

手続き型言語で書かれたプログラムは、つきつめて考えれば、命令列と情報を保存する記憶領域の相互作用によって仕事をするプログラムである。逐次プログラムの場合はこれが一対だけだが、並行プログラムの場合には複数対あって、互いに同期のための信号や情報を交換して共同作業をしている。これはほとんどどのような手続き型プログラムでも同様である。従って、さまざまな手続き型言語で書かれたプログラムは、同じ表現形式の定義使用ネットに変換でき、この統一的な定義使用ネットからプロセス従属ネットを生成するツールを1つ用意すれば各種の手続き型プログラムからプロセス従属ネットを生成することができる[12]。

すでに我々は、Ada、C、及びPascalで書かれたプログラムから統一的に表現された定義使用ネットを生成するツールと、定義使用ネットからプロセス従属ネット(プログラム従属グラフ)を生成するツールのプロトタイプを作成している[12]。プロセス従属ネット生成ツールは共通であり、任意の手続き型プログラムで

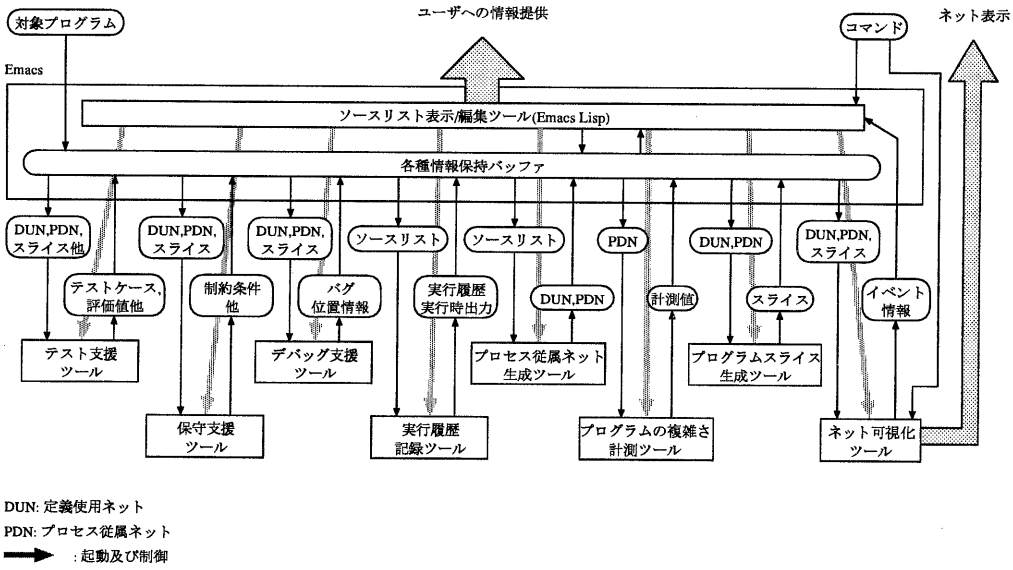

図 5: 統合的支援環境の構成

書かれたプログラムから定義使用ネットを生成するツールを追加することにより、そのプログラムのプロセス従属ネットを生成することができる。このツールの構成を図4に示す。

同様に、複数の異なる手続き型言語で書かれたプログラムに関するプログラム従属性を統一的に扱える従属性表現を設計することにより、複数の言語を単一のツール群で扱うことができる (動的解析を除く)。逐次プログラムは並行プログラムの特殊な形と考えられるため、逐次プログラムに関する開発支援も同じツール群で行える。

以上のような方針で、手続き型逐次・並列プログラムの統合的開発支援環境の設計・開発を行う。

現在開発予定のツールには、統一的ユーザインターフェイスの機能を持つソースリスト表示ツール、静的スライス生成ツール、可視化ツール、テスト・デバッグ支援ツール、保守支援ツール、プログラムの複雑さ計測ツールがある。また、各言語ごとに実行履歴記録ツールを開発することにより、動的スライス生成ツールを作ることができる。以下それぞれのツールに実装する予定の機能、及びその応用について説明する。

3.1 ソースリスト表示 / 編集ツール

このツールは、対象プログラムの定義使用ネット、プロセス従属ネット、スライス生成ツールの出力、及び実行履歴記録ツールの出力を利用し、プログラムの制御の流れの追跡、各種従属性の追跡、実行履歴のトレース、スライスの表示などを行うユーザインターフェイスを提供する。

このツールは統合的支援環境の「顔」(ユーザインターフェイス) となる。このツールから、対象プログラムの従属性解析など、各種ツールの呼び出し、出力の表示などを行う形にする予定である。また、対象プログラムの編集、コンパイル、実行などもこの上から行いたいということから、Emacs 上で実装するのが適当だと考えている。ユーザは、Emacs 上でプログラムを組み、コンパイルし、コンパイルエラーが発生しない状態ならいつでもコマンド一つで各種ツールを呼び出してその支援を受けることができる。

環境全体の構成を図5に示す。各ツールは統合的支援環境を離れて単独で動作することも

できるように設計する。したがって、統合的支援環境の「顔」であるソースリスト表示/編集ツール以外は互いに他のツールを直接呼び出すことはせず、外部から必要な情報をファイル(または標準入出力)として受け渡す形にする。

3.2 プログラムスライス生成ツール

プログラムスライス生成ツールは、対象プログラムのプロセス従属ネットから、指定された文に関する前方・後方スライスを生成する。また、指定された変数に関する分解スライス[10]を生成する。プログラムスライスは応用範囲が広く、逐次プログラムに関するスライスはかなり研究されているが、並行プログラムに関するスライスの研究はあまり進んでいない[1][7][13][17][18][19]。

プログラムのスライスとは、ある文の実行やそこで使われている変数の値に影響を与える可能性のある文のみをプログラム全体から抜き出したものである。プロセス従属ネットを用いることにより、並行プログラム中に存在する各種従属関係が明示的に示されるので、これを用いてプログラムのスライスを生成することができる。

スライスには静的スライスと動的スライスの2種類がある。静的スライスは、プロセス従属ネットにおいて注目したい文に対応する節点から到達可能な節点を全て求めることによって得られる。また、動的スライスは、プログラムのある実行履歴に含まれる節点のみを対象として、注目する節点から到達可能な節点を全て求めることによって得られる。

静的スライスは、対象プログラムから生成したプロセス従属ネットだけから計算して生成することができるが、動的スライスの生成には実際に対象プログラムを実行した履歴を必要とする。実行履歴の取得は対象プログラムの言語に依存しており、各言語ごとに動的実行監視ツールを用意する必要がある。我々はすでにPascal、C、Adaについて、対象プログラムを一定の規則で変換することにより実行履歴を記録するツールを開発しており、このツールを応用することによって、動的スライス生成に必要な実行履歴を取得することか可能である。

3.3 テスト支援ツール

このツールは、プロセス従属ネットやプログラムスライスを利用し、テストケース生成やテスト充分性の評価を行う。

逐次プログラムについて、プログラム従属グラフやプログラムスライスを使ったテストケース生成、テスト充分性評価に関する研究が報告されており、これらをプロセス従属ネットに応用することによって並行プログラムのテストを支援することができる。

3.4 デバッグ支援ツール

このツールは、プロセス従属ネットやプログラムスライスなどを利用して、プログラムのデバッグ支援を行う。

プロセス従属ネットに基づき、プログラム中の誤りが検出された文についてのスライスを求めることにより、誤りの原因となったバグが存在する可能性のある部分を取り出すことができる。バグの位置特定はプログラムのデバッグの際に最も手間のかかる作業であり、バグの位置特定を支援することはデバッギング作業の負担を大きく軽減できる。

すでにC言語について、スライスを用いたデバッギング支援ツールに関する研究が報告されている[2]。プロセス従属ネットを用いたスライス生成ツールの出力を利用することにより、並行プログラムについて同様のツールを作成することが可能である。

また、プログラムの静的・動的スライスを推論領域とし、プログラムの誤りとその原因の間の因果関係を推論対象とする因果関係推論法を開発することができれば、これを実装することにより、バグの位置を機械的に絞り込むことができる。

3.5 保守支援ツール

このツールは、主にスライス生成ツールの出力を利用することにより、プログラムの保守を支援する。

プログラムの分解スライスは、Gallagherらによって提案されている逐次プログラムの保

守法に用いられるスライスの一種である[10]。変更したい文(変数)に関する分解スライスを生成することにより、その変数に影響を受ける部分と受けない部分にプログラムを分解することができる。そして、影響を受ける部分に対してある一定の制約条件下で変更を加える。その後、その部分のみでテスト、デバッグを行って正しく動作することがわかれば、分解された2つの部分を一定の規則で合成することにより、プログラム全体も正常に動作することが保証される。従って、分解スライスを利用することにより、保守の際の手間(プログラム変更後のプログラム全体のテストの手間)が大幅に軽減できる。

また、プログラムの前方スライスと後方スライスはそれぞれある文が影響を与える文の集合、影響を受ける文の集合を示すことから、これをユーザに提示することによって、プログラム保守の際のコードの書き換え時に起こるプログラム全体への影響範囲を知らせることができる。

3.6 可視化ツール

このツールは、定義使用ネット、プロセス従属ネット及びプログラムスライスを図として表示する。

ソースリスト表示ツールがプログラムのソースリストを中心に従属性の追跡表示などを行うのに対して、可視化ツールはグラフ表示を中心に表示を行う。プロセス従属ネットを可視化してユーザに提示することにより、予期しなかった従属関係の発見などの効果が期待できる。

また、ソースリスト表示ツールと連携して、「ネット上の任意の点を指定すると対応する文を表示する」、「ソースの任意の範囲を指定すると対応するネットを表示する」といった相互の関係を表示することによって、プログラムの構成の把握を支援する。

3.7 複雑さ計測ツール

[6]に基づいて対象プログラムの複雑さをプロセス従属ネットのさまざまな観点から計測する。注目する従属枝の種類により、逐次的なデータ流れ/制御流れの複雑さ、プロセス間相互作用の複雑さ等の複雑さ計測を行う。

4 おわりに

本論文では、並行プログラムのプログラム従属性を明示的に表現するプロセス従属ネットとその応用について説明し、これに基づいた並行プログラムの統合的な開発支援環境の構築について述べた。最終的には、プログラム従属性に基づいて、逐次・並行ソフトウェアの設計、仕様記述からコーディング、テスト、デバッグ、保守、内容理解に至る、ソフトウェア開発時のさまざまな活動を支援する統合的開発支援環境を構築することが目標である。

今後、ツールと統合的支援環境のインターフェイスの設計、およびこのモデルに基づく各種開発支援ツールの開発を進めていく。

その他の課題としては、プロセス従属ネット自体の表現力を拡張する必要がある。また従属性理論を基礎とした仕様記述法、バグ位置特定支援法、プログラムテスト法などの研究を進め、これらを実装したツールをこの統合的支援環境に組み込むことにより、より広範囲なソフトウェア開発活動の支援を行う環境の構築を目指す。

参考文献

[1] Agrawal, H. and Horgan, J. R.: Dynamic Program Slicing, *Proc. ACM SIGPLAN'90*, pp. 246–256, 1990.

[2] Agrawal, H., Demillo, R. A. and Spafford, E. H., Debugging with Dynamic Slicing and Backtracking, *Softw. Pract. Exper.*, Vol. 23, No. 6, pp. 589–616, 1993.

[3] Cheng, J.: Task Dependence Net as a Representation for Concurrent Ada Programs, in *Lecture Notes in Computer Science*, Vol. 603, pp. 150–164, Springer-Verlag, 1992.

[4] Cheng, J.: The Tasking Dependence Net in Ada Software Development, *ACM Ada Letters*, Vol. 12, No. 4, pp. 24–35, 1992.

[5] Cheng, J.: Process Dependence Net of Distributed Programs and Its Applications in

Development of Distributed Systems, *Proc. IEEE-CS 17th Annual COMPSAC*, pp. 231–240, 1993.

[6] Cheng, J.: Complexity Metrics for Distributed Programs, *Proc. IEEE-CS 4th ISSRE*, pp. 132–141, 1993.

[7] Cheng, J.: Slicing Concurrent Programs — A Graph-Theoretical Approach, in *Lecture Notes in Computer Science*, Vol. 749, pp. 223–240, Springer-Verlag, 1993.

[8] Cheng, J., Kasahara, Y., Kamachi, M., Nomura, Y. and Ushijima, K.: Compiling Programs to Their Dependence-based Representations, *Proc. IEEE TENCON*, 1993.

[9] Ferrante, J., Ottenstein, K. J. and Warren, J. D.: The Program Dependence Graph and Its Use in Optimization, *ACM Trans. Prog. Lang. Syst.*, Vol. 9, No. 3, pp. 319–349, 1987.

[10] Gallagher, K. B. and Lyle, J. R.: Using Program Slicing in Software Maintenance, *IEEE Trans. Softw. Eng.*, Vol. 17, No. 8, pp. 751–761, 1991.

[11] Horwitz, S. and Reps, T.: The Use of Program Dependence Graphs in Software Engineering, *Proc. 14th ICSE*, pp. 392–411, 1992.

[12] Kasahara, Y., Cheng, J. and Ushijima, K.: A Task Dependence Net Generator for Concurrent Ada Programs, *Proc. the IPSJ & KISS Joint International Conference on Software Engineering '93*, pp. 315–322, 1993.

[13] Korel, B. and Laski, J.: Dynamic Program Slicing, *Inf. Process. Lett.*, Vol. 29, No. 10, pp. 155–163, 1988.

[14] Ottenstein, K. J. and Ottenstein, L. M.: The Program Dependence Graph in a Software Development Environment, *ACM Software Engineering Notes*, Vol. 9, No. 3, pp. 177–184, 1984.

[15] Ottenstein, K. J. and Ellcey, S. J.: Experience Compiling Fortran to Program Dependence Graphs, *Softw. Pract. Exper.*, Vol. 22, No. 1, pp. 41–62, 1992.

[16] Podgurski, A. and Clarke, L. A.: A Formal Model of Program Dependences and Its Implications for Software Testing, Debugging, and Maintenance, *IEEE Trans. Softw. Eng.*, Vol. 16, No. 9, pp. 965–979, 1990.

[17] 下村隆夫: Program Slicing 技術とテスト, デバッグ, 保守への応用, 情報処理学会誌, Vol. 33, No. 9, pp. 1078–1086, 1992.

[18] Tip, F.: A Survey of Program Slicing Techniques, Report CS-R9438, Centrum voor Wiskunde en Informatica (CWI), 1994.

[19] Weiser, M.: Program Slicing, *IEEE Trans. Softw. Eng.*, Vol. SE–10, No. 4, pp. 352–357, 1984.