プログラミング－言語・基礎・実践－ 13－5
（１９９３．８．２０）

並行プログラムのためのプロセス依存ネット生成ツール

笠原義晃　程京徳　牛島和夫
九州大学工学部情報工学科

プログラムにおいて、ある文の実行結果がその後実行される文の実行に影響を与えるように、文の間には依存関係が存在する。並行プログラム中の各文間には５種類の基本的な依存関係がある。プロセス依存ネットは並行プログラム中の基本的な５種類の依存関係を明示的に表現するラベルつき有向グラフである。本論文では、並行プログラミング言語として実用言語 Ada を例にとり、Ada 並行プログラムからプロセス依存ネットを生成するアルゴリズムを示し、これに基づいて開発したプロセス依存ネット自動生成ツールの実装と構成について述べる。また、並行プログラム開発における本ツールの応用について述べる。

Process Dependence Net Generator for Concurrent Programs

Yoshiaki KASAHARA, Jingde CHENG, and Kazuo USHIJIMA

Departmant of Computer Science and Communication Engineering

Kyushu University

6-10-1 Hakozaki, Fukuoka 812, Japan

If the execution of a statement in a program affects the execution of another statement, there is a dependence relationship between the two statements. There are five types of basic program dependences in concurrent programs. Process Dependence Net(PDN) is an arc-classified digraph to explicitly represent the five types of basic program dependences in the concurrent programs. This paper describes algorithms to compute PDNs for a class of concurrent Ada programs, and shows the structure and implementation of PDN generator for concurrent Ada programs based on these algorithms. The paper also discusses some applications of the PDN generator to development of concurrent Ada programs.

1. はじめに

プログラムの動作を理解する際には、プログラムから制御の流れとデータの流れを読み取り、ある文が他の文とどのように関係しているかを知る必要がある。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、後者は前者に依存していると言う。プログラム依存性とは、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依存関係のことである。

デバッグの際にバグの位置を特定する時や、プログラムの変更の際にその影響範囲を考慮する時など、ソフトウェア開発の際にはプログラム依存性を考慮している場面が多い。対象プログラムのプログラム依存性を自動的に抽出し明示するツールは、このような場面でプログラマの負担を軽減することができる。

逐次プログラムにおける基本的なプログラム依存性には、制御の流れによって起こる制御依存性と、データの流れによって起こるデータ依存性がある。プログラム依存性を明示的に表現するモデルとしてプログラム依存グラフが提案されている。このモデルはもともと逐次プログラムを並列化するために考案されたものであるが、それ以外にもプログラムのテスト、デバッグ、保守、複雑さの評価といったさまざまな応用分野が考えられ、研究されている[10][12]。また、対象プログラムからプログラム依存グラフ生成を行うツールについての研究報告がなされている[11]。

一般に並行プログラムには複数のプロセスが存在し、(場合によっては非決定的に)相互作用しているため、プログラム依存性の把握は逐次プログラムより困難である。従って、並行プログラム開発時にプログラマにかかる負担は大きい。並行プログラムの依存性を自動的に抽出し明示するツールは並行プログラム開発の際に有用であると考えられる。しかし、並行プログラムのプログラム依存性を明示的に表現するモデルは今まで提案されていなかった。このため、このようなツールに関する研究も行われていなかった。

プログラム依存グラフは複数のプロセス間の相互作用を表現できないため、並行プログラムにおけるプログラム依存性を表現することができない。程(Cheng)は、並行プログラムにおけるプログラム依存性を表現するために、プロセス間の制御の流れによる同期依存性、プロセス間のデータの流れによる通信依存性、そして並行プログラムの非決定的な動作よる選択依存性の3種類の基本的なプログラム依存性を新たに提案した。そして、並行プログラムにおけるプログラム依存性を明示的に表現するモデルとしてプロセス依存ネットを提案した[5][6]。しかし、プロセス依存ネットを具体的にどのようにして対象プログラムのソースから生成するかについての研究はまだ行われていなかった。プロセス依存ネットを応用の場で使うためには、実際にプログラムのソースからプロセス依存ネットを生成する手法及びそれに基づいた自動生成ツールの開発が必要である。

本研究は、並行プログラミングが可能な実用言語Ada[16]について、Adaプログラムから各依存関係を抽出するアルゴリズムを考案し、それに基づいてプロセス依存ネットを自動生成するツールを開発することを目的とする。Adaでは並行実行単位をタスクと呼ぶため、Adaに適用したプロセス依存ネットのことをタスク依存ネットと呼ぶ。本論文では今後用語をタスク依存ネットで統一する。

以下、第2章では、プログラム依存性及びタスク依存ネットについて述べる。第3章では、タスク依存ネット生成の手順について述べる。第4章では、タスク依存ネット生成ツールの実現について述べる。第5章では、タスク依存ネットの応用について述べる。第6章では、本研究のまとめを行い、今後の課題について述べる。

2. タスク依存ネット

タスク依存ネットは、Adaプログラム中の各単純文及び制御論理式を節点とし、5種類の基本的な依存関係を表す5種類の異なる有向枝によって各文間に存在する各種依存関係を表現するラベル付き有向グラフである[5]。

図1にタスク依存ネットの例を示す。この図は、図2に載せた3つのタスク、T、PERCENT、FACTを持つAdaプログラムのタスク依存ネットである。

図2の行頭の記号は図1の各節点に対応している。なお、図1では、他の節点と接続していない節点t_1、t_2、p_1、f_1、及びメインタスクの節点m_1、m_2を省略している。

5種類の基本的な依存関係について簡単に説明する。それらの形式的定義については[5]を参照されたい。なお、説明の例にある節点は全て図1及び図2のものである。また、今後Adaの用語を断りなしに使用する(詳しくは[16]を参照されたい)。

2.1 単一タスク内に存在する依存性

各タスクはそれ単体で見ると逐次プログラムであると考えることができる。従って、タスクの内部では従来の逐次プログラムと同じく制御依存性とデータ依存性が存在する。選択依存性もタスク内の制御の流れの変化に関する依存性であるため、ここで説明する。

制御依存性 制御依存性は1つのタスクの中に存在する制御の流れの分岐によって発生する依存関係である。if文などにおける条件式vの評価によって文uが実行されるかどうかが決まる場合にuはv

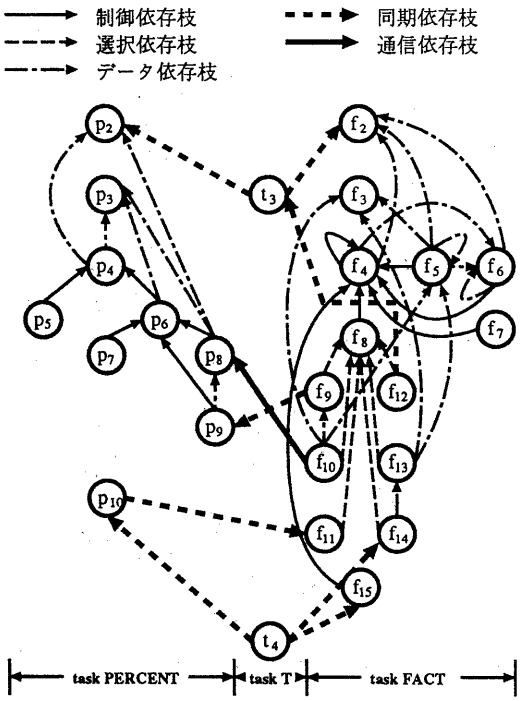

図 1: タスク依存ネットの例

に制御依存している。

データ依存性 データ依存性は1つのタスクの中に存在するデータの流れによって発生する依存関係である。文 v で計算されている変数が文 u で計算される変数の値に影響を与える場合に u は v にデータ依存している。

選択依存性 制御の流れが非決定的選択によって変化する場合に、非決定的選択文と選択肢の間には選択依存性が存在する。非決定的選択とは、プログラムに同じ入力が与えられても、各タスクの実行タイミングにより実行のたびに選択肢が変わる可能性のある選択である。

選択依存性と制御依存性は同一タスク内での制御の流れの変化という点で似ている。しかし、制御依存性は制御依存している文の実行が条件分岐文の評価、ひいてはプログラムへの入力によって一意に決定されるのに対し、選択依存性はプログラムへの入力が同じでも複数のタスクの実行タイミングの変化により選択肢が異なってくる可能性がある。このため、制御依存性と区別して考慮する必要がある。

```
        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 2: サンプルプログラム

2.2 2つのタスク間に存在する依存性

タスク間で同期を取ったり、通信が行われると、それらのタスクの間には依存関係が発生する。これらの依存関係を表現する依存性が同期依存性と通信依存性である。

同期依存性 同期依存性は2つのタスク間での同期によって発生する依存関係である。あるタスクにある文 v の処理の開始や終了によって別のタスクにある文 u の処理の開始や終了が影響を受ける可能性がある場合に u は v に同期依存している。

通信依存性 通信依存性は2つのタスク間の通信によって発生する依存関係である。あるプロセスにある文vで計算される変数が別のタスクにある文uで計算される変数の値に影響を与える可能性がある場合にuはvに通信依存している。

3. タスク依存ネット生成の手順

タスク依存ネット生成の手順は大きく次の4つのステップからなる。

- 対象プログラムのソースを解析し、制御の流れ、変数への値の定義及び変数の値の使用、タスクの親子関係やランデブによるタスク間の同期及び通信の情報を取り出す。これらの情報に基づいて定義使用グラフ (後述) を生成する。
- 定義使用グラフの情報から、各タスクごとにタスク内の制御依存性、データ依存性、選択依存性を抽出する。
- タスク内の依存関係と定義使用グラフの情報から同期依存性、通信依存性を抽出する。
- 全ての依存関係をまとめてタスク依存ネットを生成する。

以下それぞれについて順に説明する。

3.1 定義使用グラフの生成

定義使用グラフとは、プログラムにおける各単純文及び制御論理式を表す節点と、制御の流れを示す有向枝により構成される有向グラフである。各節点にはその節点で値を定義している変数、値を使用している変数、他のタスクへのエントリ呼び出し、他のタスクからのエントリ呼び出しの受け付けに関する情報を記す。図3に図2のプログラムに関する定義使用グラフを示す。

図3において、Dは各節点で定義されている変数の集合、Uは各節点で使用されている変数の集合、Sは各節点がどのエントリへの呼び出し開始や受け付け終了による相手の待機状態解消に関係するか、Rは各節点がどのエントリの受け付け開始や呼び出し終了に伴なう待機状態発生に関係するかを示す。太線の有向枝は親タスクが子タスクを生成することによる並行的な制御の分岐、及び子タスクの終了による親タスクへの制御の収束を示す。また、破線の有向枝は非決定的な選択による制御の流れ、その他の有向枝は通常の制御の流れを表す。節点番号の右肩の'*'はその節点からタスク本体の実行が始まることを示す。これは、タスクの親子関係による依存性を求める際に必要である。p_1、t_1、f_1は各タスクの宣言部の実行が開始される文を示している。

タスク依存ネットは定義使用グラフに基づいて定義されているため、対象プログラムから定義使用グラフ

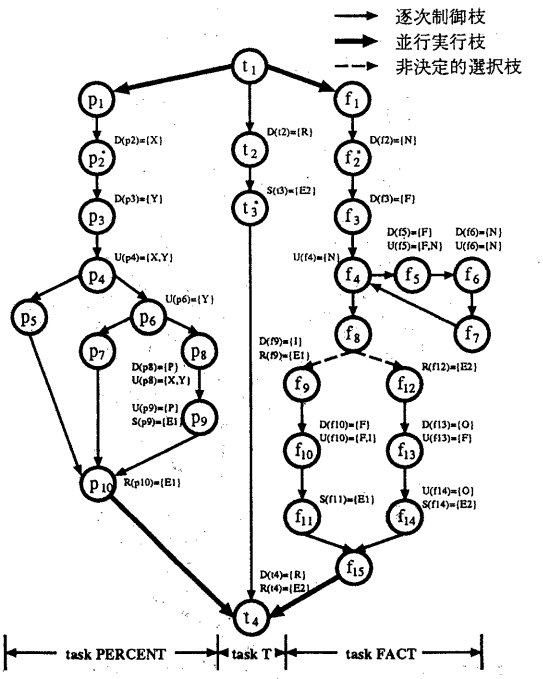

図 3: 定義使用グラフの例

を生成すれば、それをもとにタスク依存ネットを生成することができる。

3.2 各タスクごとの依存グラフ生成

各タスクの内部に存在する制御依存性及びデータ依存性は、本質的に逐次プログラムにおけるものと同じものであり、逐次プログラムのプログラム依存グラフ生成で用いられている手法を使うことができる[2][11]。また、選択依存性を求める際には制御依存性の場合と同じ手法が使えるため、制御依存性を求める際に同時に計算することができる。

3.3 タスク間の依存関係の抽出

タスクの親子関係とタスク間の呼び出し、受け付けの情報から、タスク間の依存関係を抽出する。解析は静的に行うため、同期、通信の可能性がある部分はすべて依存していると考える。従って、呼び出しと受け付けとのすべての組合せについて同期依存関係があるとする。親子関係による同期依存性は、タスクの親子関係と各タスクの本文の開始点を示す節点、及び各タスクの終了点を示す節点から決定できる。

通信依存性は、タスクのランデブ時に受け渡される引数に関するデータ依存関係から求める。Adaでは複数のタスクから可視の大域変数を用いてデータのやりとりが可能であるため、大域変数による通信依存性も

```
-- 入力: 定義使用グラフ、タスクごとのプログラム依存グラフ
-- 出力: タスク依存ネットの同期依存部分
for each タスク T loop
   if T が子タスクを持つ then
      親の実行開始点から子の実行開始点に同期依存枝接続;
      親の実行終了点から子の実行終了点に同期依存枝接続;
   end if;
   if T が他のタスクのエントリを呼び出している then
      for each T のエントリ呼び出し文に対応する節点 n loop
         節点 n での呼び出しに対応する受け付け側の節点 m(複数可) から
         節点 n に同期依存枝接続;
         呼び出し側節点 n の次に処理の移る節点 n' から
         呼び出しに対応する受け付け終了節点 m'(複数可) に同期依存枝接続;
      end loop;
   end if;
end loop;
```

図 4: 同期依存抽出アルゴリズム

```
-- 入力: 定義使用グラフ、タスク毎のプログラム依存グラフ
-- 出力: タスク依存ネットの通信依存部分
for each タスク T loop
   if T が他のタスクのエントリを呼び出している then
      for each T のエントリ呼び出し文に対応する節点 n loop
         if n で引数を渡している then
            呼び出しに対応する受け付け側の節点 m(複数可) にデータ依存する節点から、
            n がデータ依存する節点に通信依存枝接続;
         end if;
         if n で引数を受け取っている then
            n にデータ依存する節点から受け付けの終了節点 m' が
            データ依存する節点に通信依存枝接続;
         end if;
      end loop;
   end if;
end loop;
大域変数の依存関係を全て接続
```

図 5: 通信依存抽出アルゴリズム

考慮する必要がある。

大域変数については単に各タスク内で局所的でない変数の定義や使用について、全ての依存関係の組合せを求める。静的解析で実行時にはありえない依存関係を割り出すことは非常に困難であり、今回は考慮していない。

図 4に同期依存性を、図 5に通信依存性をそれぞれ抽出するアルゴリズムの概要を示す。各タスク内の制御依存性、データ依存性は求まっているものとする。

4. タスク依存ネット生成ツールの実現

本研究では、Adaのためのタスク依存ネット生成ツールのプロトタイプをAdaで開発した。プログラムの概要は図6の通りである。

プログラムは、大きく分けて定義使用グラフ生成部、プログラム依存グラフ生成部、タスク依存ネット生成部の3つの部分により構成されている。それぞれについて順に説明する。

4.1 定義使用グラフ生成部

定義使用グラフ生成のためには対象プログラムのソースを字句・構文解析する必要がある。本ツールの開発には、コンパイラコンパイラ aflex と ayacc を利用した[13][15]。これらのツールは Ada のソースを出力することができる。

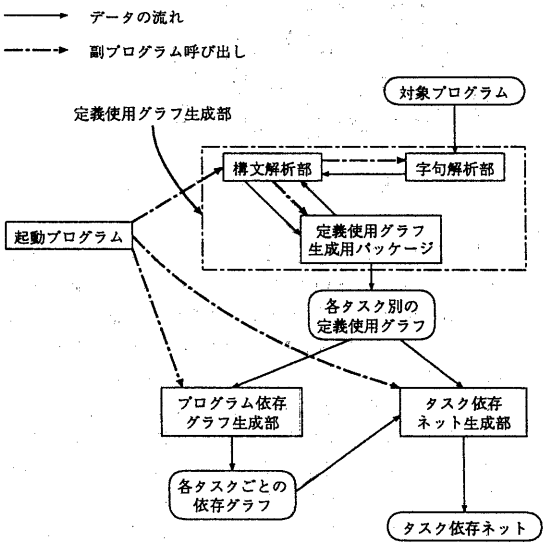

図 6: タスク依存ネット生成ツールの構成図

ツール実行時の処理の流れは以下の通りである。

i) 起動プログラムが入力となる対象プログラムの

ソースファイルを開いて構文解析部を起動する。

ii) 構文解析部は字句解析部を呼び出すことによりソースファイルをトークンに分けて取得し、構文規則に従って構文解析を行う。

iii) 構文解析部は構文解析の結果に従って、定義使用グラフ生成用パッケージ内の各種サブルーチンを呼び出すことにより必要な情報の受け渡しを行い、定義使用グラフを生成する。

iv) ソースファイルを最後まで処理したら、定義使用グラフを各タスク別にファイルとして出力する。

各節点の情報、各タスクの情報はすべてレコード型の変数で保持し、それをアクセス型のリンクで相互に接続する。各タスク間の親子関係もリンクによる木構造で保持し、これを上下に辿ることにより変数やエントリの可視規則を処理する。各変数には、大域、局所を問わず対象プログラム全体で一意的な変数番号を与える。プログラム中に出現した変数はその時点で名前と出現位置から変数番号を特定し、定義使用グラフには変数を変数番号で記録する。

定義使用グラフは通常のテキスト形式で出力する。図7に例を示す。これは図2のプログラム中のタスクFACTの定義使用グラフである。

```
 1: line 31 connect 2
 2: line 34 connect 3 def 5
 3: line 35 connect 4 def 6
 4: line 37 connect 5 8 use 5
 5: line 38 connect 6 def 6 use 5 6
 6: line 39 connect 7 def 5 use 5
 7: line 40 connect 4
 8: line 41 s-connect 9 12
 9: line 42 connect 10 def 7 receive 1
10: line 43 connect 11 def 6 use 7 6
11: line 44 connect 15 send 1
12: line 46 connect 13 receive 2
13: line 47 connect 14 def 8 use 6
14: line 48 connect 15 use 8 send 2
15: line 50 p-connect 2-4
```

図 7: 定義使用グラフの出力例

図7において、行頭の数字は各タスクごとに1から割り振られている節点番号、line に続く数値は行番号、connect に続く数値はその節点に対応する文が実行された後次に制御が移る可能性のある文に対応する節点の番号、s-connect に続く数値は非決定的な選択によって制御が移る可能性のある文に対応する節点の番号、p-connect に続く数値は他のタスクへの並行的な制御の流れ(ここではタスクの終了による親タスクへの収束)を表す。また、def はその節点で値を定義された変数、use はその節点で値が使用された変数、send はその節点での他のタスクへの呼び出し、receive はその節点での他のタスクからの呼び出し受け付けを示す。節点と行番号の対応はプログラム依存グラフ生成には必要がないが、定義使用グラフや、これに基づいて求めたプログラム依存グラフ、タスク依存ネットを実際に使用する際に、ソースのどの部分を示しているかを対応づけるために必要となる場合に備えて付加してある。

現在、この例でもわかる通り節点とソースの対応は行番号のみで行っている。プログラムは各文に1つの操作しか行っていないように整形されているものと仮定している。実用に際しては、任意のソースをその要求に沿うように整形する整形プログラムを用意するか、ある節点がソースのどの部分を指しているかをより正確に示すような表現を考える必要がある。

4.2 プログラム依存グラフ生成部

第3章で述べたように、タスク内の依存関係を抽出する際には逐次プログラムのプログラム依存グラフ生成法を用いることができる。このため、我々の研究室で別に開発された逐次プログラム用のプログラム依存グラフ生成ツールを利用した[7][9]。

各タスク別に出力された定義使用グラフはそれぞれプログラム依存グラフ生成部に渡され、各タスク別のプログラム依存グラフに相当する出力となる。出力されたプログラム依存グラフの例を図8に示す。これも同じく図2のプログラム中のタスクFACTに関する依存グラフである。

```
 1:
 2:
 3:
 4: control 4 data 6 2
 5: control 4 data 6 2 5 3
 6: control 4 data 6 2
 7: control 4
 8: control 4
 9: control 8
10: control 8 data 9 5 3
11: control 8
12: control 8
13: control 8 data 5 3
14: control 8 data 13
15: control 4
```

図 8: プログラム依存グラフの出力例

図8において、control に続く数値はその節点が制御依存している節点番号、data に続く数値はその節点がデータ依存している節点番号である。この図では、選択依存性が制御依存性として出力されているのがわかる。これはプログラム依存グラフ生成部が逐次プログラム用であるため制御依存性と選択依存性を区別しないためである。必要に応じて定義使用グラフの情報によりタスク依存ネット生成部側で選択依存性に変換する。

4.3 タスク依存ネット生成部

タスク依存ネット生成部は、定義使用グラフ生成部で対象プログラムのソースから得た各種情報と、プログラム依存グラフ生成ツールの出力の両方を利用し、第3章で説明したアルゴリズムに従ってタスク依存ネットを生成する。

出力されたタスク依存ネットの例を図9に示す。

図9において、cont、data、sele の後の数値はそ

れぞれその節点が制御依存、データ依存、選択依存する節点番号である。syncとcommはそれぞれ同期依存、通信依存を表している。これに続くハイフンでつながった2つの数値は、前が相手のタスク番号、後が依存する相手タスクの節点の番号である。

```
Task No. 1(MAIN)
 1: line 57 sync 2-3
 2: line 58 sync 2-4
Task No. 2(T)
 1: line 8
 2: line 9
 3: line 53 sync 4-2 3-2
 4: line 54 sync 4-10 3-15 3-14
Task No. 3(FACT)
 1: line 31
 2: line 34
 3: line 35
 4: line 37 cont 4 data 2 6
 5: line 38 cont 4 data 3 5 2 6
 6: line 39 cont 4 data 2 6
 7: line 40 cont 4
 8: line 41 cont 4
 9: line 42 sele 8 sync 4-9
10: line 43 data 3 5 9 sele 8 comm 4-8
11: line 44 sele 8
12: line 46 sele 8 sync 2-3
13: line 47 data 3 5 sele 8
14: line 48 data 13 sele 8
15: line 50 cont 4
Task No. 4(PERCENT)
 1: line 16
 2: line 19
 3: line 20
 4: line 21 data 2 3
 5: line 22 cont 4
 6: line 23 cont 4 data 3
 7: line 24 cont 6
 8: line 26 cont 6 data 2 3
 9: line 27 cont 6 data 8
10: line 29 sync 3-11
```

図 9: タスク依存ネットの出力例

現在の実装では、図6中のタスク依存ネット生成部は1つのパッケージからなっているが、我々は現在各依存関係ごとに生成ツールを分割し、独立で動作するように変更を行っている。これは、応用に際して一部の依存関係のみを必要とするような場合に利用しやすいようにするためである。各種依存関係をまとめて完全なタスク依存ネットにするのは容易である。

5. タスク依存ネットの応用

タスク依存ネットは、Ada並行プログラムの表現モデルとして、多くの応用の可能性がある。

5.1 プログラムスライス

タスク依存ネットが対象プログラム中にあるさまざまな依存関係を表現していることから、タスク依存ネットの最も直接的な応用として、プログラムのスライス生成[1][8][14][17]が考えられる。

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。タスク依存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依存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。

タスク依存ネットに基づき、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。プログラムのデバッギングで最も手間のかかる作業はバグの位置を特定することであり、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。

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、タスク依存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、取得した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。

静的スライスはタスク依存ネットのみで求められる分容易に求められるが、依存する可能性がある文はすべてスライスに入るためスライスは大きくなりすぎる傾向にある。動的スライスは、異常が発生した実行時の履歴があれば、その実行に関する異常に関係する部分がスライスに入るため、デバッグについてはより有効だと考えられる。しかしながら、実行履歴の取得は対象プログラムの実行に影響を与えるため、使用には注意が必要である。

5.2 タスク影響ネット

プログラムの内容の理解やプログラムの保守、維持の際に、ある文の実行結果がプログラム全体のうちどの程度の範囲に影響を与えているかを知りたいことがある。このような場合には、プログラムの前方スライスが役に立つ。

プログラムの前方スライスとは、ある文の実行やそこで定義されている変数の値によって影響を与えられ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。

前方スライスを求める際には、タスク影響ネットを利用することができる。タスク影響ネットとは、タスク依存ネットの有向枝を全て逆転させたグラフである。この時、逆転した有向枝は、その節点の実行により影響を与える可能性のある節点を指すことになる。よって、タスク影響ネットに基づいて、静的前方スライス、動的前方スライスを得ることができる。

5.3 その他の応用

タスク依存ネット及びタスク影響ネットはAda並行プログラム中にある各タスク内、またタスク間の各種依存関係を表現している。これらを用いることにより、並行プログラムの並列化、依存関係に基づいたプログラムのテストケース評価基準の定義、依存関係の複雑さからプログラムの複雑さの尺度を定義することができると考えられる。

6. おわりに

本研究では、タスク依存ネットに基づいて、Adaプログラムのソースからタスク依存ネットを生成するツールのプロトタイプを開発した。現在このツールは、ブロックや副プログラム、動的に生成されるタスク型を含まないAdaプログラムのソースから、タスク依存ネットを生成することができる。このツールによって、今まで概念的モデルだけであったタスク依存ネットを実際にAdaプログラムのソースから生成することができるようになり、タスク依存ネットを利用することが可能となった。

今後の課題としては、第一に、このツールを任意のAdaプログラムに適用できるように拡張することが挙げられる。

現在本ツールで用いているタスク依存ネット生成アルゴリズムは、配列型、レコード型、アクセス型といった複雑な型の変数を考慮していない。任意のAdaプログラムに対応するためには、今後これらに対応する処理を実装していく必要がある。

現在提案されているタスク依存ネットは、ブロックや副プログラム、プログラムの実行時に動的に生成されるタスクを扱うことができない。このため、このモデルをもとに開発された本ツールもブロックや副プログラム、動的に生成されるタスクを使用したプログラムを正しく解析することができない。タスク依存ネットにこれらに対応する定義を加え、それに併わせてツールを拡張する必要がある。動的に生成されるタスクは、タスク依存ネット生成が対象プログラムの静的解析であることを考えると正しく処理することは不可能である。

また、実行時間や実行時の消費メモリ量といった、ツールの性能評価を行う必要がある。

タスク依存ネット及びタスク依存ネット生成ツールは、タスク依存ネットを用いた応用ツールを開発することにより意味を増す。したがって、今後このツールによって出力されたタスク依存ネットを用いた応用ツールの開発研究を行っていく必要がある。具体的には、デバッギングへの応用としてAdaプログラムに対するスライス生成ツールの開発を行っていく。また、第5章で述べたその他の応用についても研究を進めていく。

参考文献

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

[2] A. V. エイホ, R. セシィ, J. D. ウルマン共著, 原田賢一訳, "コンパイラ I,II: 原理・技法・ツール", サイエンス社, 1990.

[3] 程京徳, 荒木啓二郎, 牛島和夫, "Ada並列プログラムの事象駆動型実行モニター EDEN の開発と応用", 情報処理学会論文誌, Vol. 30, No. 1, pp. 11-24, 1989.

[4] J. Cheng, Y. Kasahara, and K. Ushijima, "A Tasking Deadlock Detector for Ada Programs," Proc. IEEE-CS 15th Annual COMPSAC, pp. 56-63, 1991.

[5] J. Cheng, "Task Dependence Net as a Representation for Concurrent Ada Programs," in J. van Katwijk (ed.), "Ada: Moving towards 2000," LNCS, Vol. 603, pp. 150-164, 1992.

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

[7] 蒲池正幸, "Pascal プログラムのためのプログラム依存グラフの生成", 九州大学工学部電子工学科卒業論文, 1993.

[8] B. Korel and J. Laski, "Dynamic Program Slicing," Information Processing Letters, Vol. 29, No. 10, pp. 155-163, 1988.

[9] 乃村能成, "C プログラムのためのプログラム依存グラフの生成", 九州大学工学部電子工学科卒業論文, 1993.

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

[11] K. J. Ottenstein and S. J. Ellcey, "Experience Compiling Fortran to Program Dependence Graphs," Software–Practice and Experience, Vol. 22, No. 1, pp. 41-62, 1992.

[12] A. Podgurski and L. A. Clarke, "A Formal Model of Program Dependences and Its Implications for Software Testing, Debugging, and Maintenance," IEEE-CS TOSE, Vol. 16, No. 9, pp. 965-979, 1990.

[13] J. Self, "Aflex – An Ada Lexical Analyzer Generator Version 1.1," UCI-90-18, 1990.

[14] 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.

[15] D. Taback, D. Tolani, and R. J. Schmalz, "Ayacc User's Manual Version 1.0," UCI-88-16, 1988.

[16] United States Department of Defense, "Reference Manual for Ada Programming Language," ANSI/MIL-STD-1815A, 1983.

[17] M. Weiser, "Program Slicing," IEEE-CS TOSE, Vol. SE-10, No. 4, pp. 352-357, 1984.