プログラムの自動生成における入出力条件に関する一考案

恐　神　正　博*

A study of Input and Output Conditions for Automatic Program Generation

Masahiro Osogami*

Abstract

Since the 1980's, object oriented programming and structured programming have been hot topics for increasing software productivity. Usually, software has been designed by programmers while other products are machine made. However, demand for software has been increasing due to the influence of factory and office automation, so a software crisis has begun.

In this paper, a method of automatic program generation is proposed. This paper uses specifications written in easily understood language, and data structures with customized target objects using the generic PROLOG module library. During program generation, the automatic program generation system MAPP tries to compensate for any missing specifications which were not completely described using input and output conditions of modules. In addition Petri Nets will be used during the generated programs conditions consideration.

１．まえがき

１９８０年代の中ごろから構造化プログラミング設計法やオブジェクト指向の手法等が提唱され，プログラミングの生産性向上の研究及びそれらの実用化が進められている．一方，従来より手作業によって行われてきたプログラミングという作業を機械化する試み，すなわちプログラムの自動生成に関する技術についても研究が進められている．

ここではモジュールの機能や処理対象のデータの型などに関する属性項目を記述したモジュールの見出し表を作成し，そこから仕様に対して適用可能なモジュールを検索し，C 言語のプログラムを生成する一つの方法と，その中でモジュールの自動リンクの際行われる入出力条件のチェック法において，従来行ってきた[2][3]Prolog のリスト処理の他に，ペトリネットを用いた方法について検討を行う．

２．モジュールの構成

我々は，従来より Prolog を用いた自動生成システムの一つの構成法について，その実験システム MAPP (Module Aided Programming system by Prolog) を通し報告を行なってきているため[2][3]，ここでは

＊ 経営情報学科

Prologのリスト処理を用いたモジュールの構成及びその検索法について説明を行う．一般にMAPP内においてモジュールは，次のような構成を持つ．

```
module(Head,Tail).                                    (2-1)
```

ここでHeadやTailの引数部はリスト形式を取る．Headはモジュールの機能の種別を表す呼び出し文項目や処理対象のデータ構造の項目などからなる頭部であり，Tailはリスト形式により不定長・不定数の属性項目からなる尾部である．尾部の項目としては，モジュールの関数の型と機能を日本語文で説明するコメント項目，この関数をC言語プログラムでコールするときの関数表現の項目，引き数部の処理対象の変数の型や，関数の戻り値の型を記述する対応項目，このモジュールを適用するのに必要な入力データやその条件を記述する入力項目，適用後，出力するデータの名前や性質を記述する出力項目，このモジュールを収納するファイル名項目などである．これらの属性項目の記述は，リストを用いることにより不定長でよく，また属性項目の種類はモジュールを通じて同じである必要はない．

3．ソースコードの生成

概略仕様は処理関連対象の型やデータ構造を指定するデータ構造仕様と，処理方法を指定する処理仕様に大別し，データ構造仕様を，

```
obj_list(spec(NUM)):-obj(x1), obj(x2),･･･,obj(xn).    (3-1)
```

の形で与える．

次に処理仕様を，それぞれの処理項目をprocess_1, process_2,･･･,process_nとした時，

```
process(spec(NUM)):-proc_list([ process_1,
        process_2,
        ･･･  ,
        process_n ]).                                 (3-2)
```

で与える．[3]

概略仕様として上の式(3-1)および式(3-2)がそろった後，次の式(3-3)を実行することでC言語のソースコードが生成される．[4]

```
prog(spec(NUM)):-
    lang(c),obj_list(spec(NUM)),
    main_head,process(spec(NUM)),
    main_end.                                         (3-3)
```

式(2-1)で与えるモジュールには，モジュールの適用条件である入力項目 in(IN)，処理後の状態や出力データを記述する出力項目out(OUT)を設けている．従って，これらの知識を用いることにより，モジュールを組み合わせて作ったプログラムの実行可能性に関するチェックや，逆に入出力条件を与えこれを満たすプログラムをいくつかのモジュールをリンクして生成することができる．その概念図を図1に示す．

図1では，まず function_n がリンクされる際に入力条件を蓄えているリスト(apply_conditions)に自身の入力条件（input condition）が存在しているかどうかを調べ，存在した場合自身の出力条件(output condition)をリスト上に追加しリンクされる．次にfunction_n+1がリンクされる際，同様に自身の入力条件が既に存在しているかどうかを調べる．例えば，function_n+1の入力条件がfunction_nの出力条件であった場合，

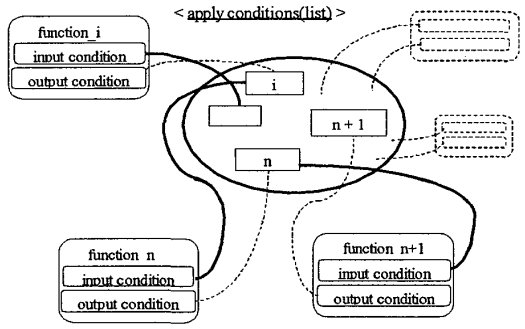

図１．入出力条件のチェック概念図

すでに apply conditions のリストに存在しているので function_*n*+1 がリンクできることになる．すなわち，finction_*n*+1 がリンクされるための前提条件が function_*n* であったことになる．

このように，設計文に記述されているすべてのモジュールに対しそれらの入力条件をチェックしていくが，１つでも条件を満たしていない場合，このリンクは失敗に終わってしまう．そこで，もし自分の入力条件が存在しなかった場合，モジュールを蓄えている辞書の中から必要な前提条件をもつモジュールを

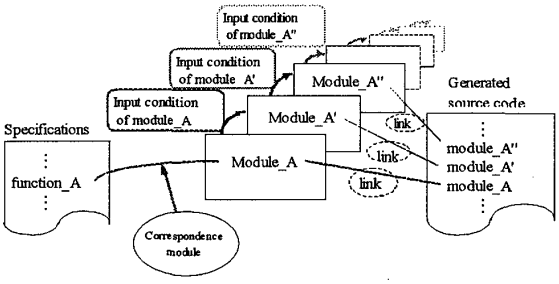

図２．設計文における欠落部分の自動補填

探索し，自動的に自分の前にリンクしていく自動補填の考え方を取り入れる．

図２では，最初の処理仕様において記述されていた function_A のみの部分について，最終的にその入力条件を満たすために function_A に対応する module_A のみならず module_A'および module_A''が自動的に補填されており，結果としてすべての入力条件が満たされた形でプログラムが生成されていることが示されている．

この自動補填を行っていくためには，式(3-4)の出力述語 ouput(OUT)の本体部が真となるように，本体部の述語を逐次実行していけばよい．まず，モジュールの尾部 Tail の出力項目 out(OUT)の引き数が仕様の出力条件 OUT と一致するモジュールを探索する．成功すればそのモジュールの入力項目 in(IN)やタイプ項目の引き数 TYPE が，仕様で与えた入力述語 input(IN)の引き数 IN やタイプ述語 typep(TYPE)の引き数とそれぞれ一致するかどうかを調べる．一致すれば④～⑥で型宣言すべき関数やパラメータ変数，一時置換変数をモジュール尾部の各項目から type_member 述語の引き数に登録し⑦でインクルードすべきファイルがあればこれを登録し，⑩で OUT を満たす手続きを与える関数表現を出力し，⑪でその関数表現に対する日本語文を注釈用にモジュール尾部の com 項目から com_set 術語の引き数部にコメント用として収録する．

もし，このような入力データが含まれていなければ，式(3-5)の関係を用いて，X を出力項目に含むモジュールの探索を繰り返す．探索に失敗すればプログラム生成は失敗に終わる．

```
output(OUT):- module(Head,Tail),
    member(out(OUT),Tail),                          ①
    member(in(IN),Tail),input(IN),                  ②
    member(type(TYPE),Tail),typep(TYPE),            ③
    memebr(funct_type(FUNCTTYPE),Tail),             ④
    funct_declared(FUNCTTYPE),
    (memebr(para_obj(OBJ,type(OBJTYPE)),Tail),      ⑤
    para_declared_list(OBJ,OBJTYPE);none),
    (member(tmpvar_type(VARTYPE),Tail),             ⑥
    tmpvar_declared(VARTYPE);none),
    (member(file_name(FILE_NAME),Tail),             ⑦
    include(FILE_NAME);name),
    member(funct(FUNCT),Tail),                      ⑧
    (state(STATE),member(FUNCT,STATE);              ⑨
    nl,writelist(FUNCT),write(','),                 ⑩
    member(com(COM),Tail),com_set(SET),             ⑪
    add_com_set(COM,SET),addstate(FUNCT,STATE)).    ⑫      (3-4)
input(X):-output(X).                                       (3-5)
```

他方，このような探索を繰り返して，仕様で与えられた入力条件を満たすモジュールに達すれば，これまでたどってきた探索の形路とは逆方向に初めの出力条件 OUT を含むモジュールに向かって，各モジュールにおける Tail 部に記載された各種の変数や関数の型の登録，ファイルの登録，関数 FUNCT の出力を再び繰返すことで自動補填を行いつつメイン関数のプログラムを生成することができる．

４．入出力条件のチェックにおける検討

上述したように，入出力条件のチェックはおのおののモジュールごとに入出力条件を設け，モジュールがリンクする際，前提としてその入力条件がすでに存在するかどうかでチェックを行っていた。

確かに，モジュールの適用条件として，その前提となる入力条件をもつ他のすべてのモジュールを事前に，もしくは同時に適用するこの方法は，小さなプログラムを生成する場合は十分有効であると思われるが，規模の大きなプログラムを生成していく場合，条件が複雑に絡み合うことで，特に自動補填を行った場合等，予期せぬ障害が生じる可能性を否定できない．

そこで，システムの動作可能性を検証するために，ペトリネットを用いることを考える．

４．１　ペトリネット

ペトリネットとは，ドイツ人、C.A. Petri によって 1962 年に提案された概念であり，システムにおける条件と事象の概念を用いてシステムをグラフモデル化したものである．

ペトリネットには，可到達性，有界性，安全性，活性などの性質があり，モデル化されたシステムについてこれらの性質を解析することで，システムの構造と動的な挙動について評価・変更・改良をすることが可能となる．また，グラフィックなツールとして，フローチャートやブロックダイヤグラム・ネットワ

ークと同じようにシステム構造の可視的な表現手段として使用できるだけでなく，ペトリネットの中のトークンと呼ばれるものを使用することにより，システムの並行的でダイナミックな事象をシミュレートすることができる．

一方，これらの性質はシステムの挙動を表現する状態方程式や代数方程式等を用いて，数学的に解析することが可能であり，入出力条件のチェックにおける数学的な裏づけを得ることが可能になる．

これは，システムをモデル化したペトリネットにおける接続行列 A，その rank を r，列数を m，行数を n とした時，

$$A = \begin{array}{c|cc} & m-r & r \\ \hline r & A_{11} & A_{12} \\ n-r & A_{21} & A_{22} \end{array} \qquad \text{(4-2-1)}$$

で表される A_{11}, A_{12} および m-r 次の単位行列を I_μ を用いて，

$$B_f = [I_\mu : -A_{11}{}^{\mathrm{T}} \ (A_{12}{}^{\mathrm{T}})] \qquad \text{(4-2-3)}$$

で表わされる B_f を求め，さらに，初期マーキングを M_0，最終マーキングを M_d とした場合のマーキング差 $\triangle M = M_d - M_0$ と B_f を用いると，B_f と $\triangle M$ 間には，M_0 から M_d が可到達であるなら，

$$B_f \triangle M = 0 \qquad \text{(4-2-3)}$$

が成立つことが言える[1]ため，システムを表すペトリネットにおいて式(4-2-3)が成立つかどうかを調べることで，システムの可動性を検証することができる．

5　入出力条件の検証を行うための例

今簡単のために，process_1, process_2 の２つの処理が順番に行われる場合を考える。ペトリネットで表わすと図３のようになり，process_1 が t_1，process_2 が t_2 に対応し，process_1 の入力条件が p_1，出力条件が p_2，process_2 の入力条件が p_2 出力条件が p_3 となる．

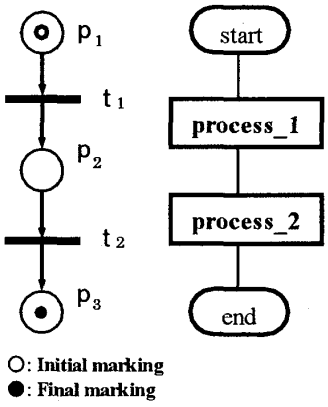

図３　process_1, process_2 の順次処理をペトリネットで表した場合の例

ここで，t_1，t_2 は実行により発火されるトランジションを表し，p_1，p_2 はそれぞれのプレースを表している．

ここで，このペトリネットにおける接続行列 A は

$$A = \begin{vmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{vmatrix} \tag{5-1}$$

であり，このとき

$$A_{11} = \begin{vmatrix} 1 & 0 \end{vmatrix} \tag{5-2}$$

$$A_{12} = \begin{vmatrix} -1 & 1 \\ -1 & 1 \end{vmatrix} \tag{5-3}$$

である．また，式(4-2-3)から，

$$B_f = \begin{vmatrix} 1 & 1 & 1 \end{vmatrix} \tag{5-4}$$

となり，初期マーキング $M_0 = \begin{vmatrix} 1 & 0 & 0 \end{vmatrix}^T$，最終マーキング $M_d = \begin{vmatrix} 0 & 0 & 1 \end{vmatrix}^T$，であることから，そのマーキング差$\triangle M$は，

$$\begin{aligned} \triangle M &= M_d - M_0 \\ &= \begin{vmatrix} -1 & 0 & 1 \end{vmatrix}^T \end{aligned} \tag{5-4}$$

ここで確かに

$$B_f \triangle M = 0 \tag{5-5}$$

となるため，式(4-2-3)からM_0からM_dは可到達，すなわちシステムが実行可能であることが検証できる．

６．あとがき

プログラムの自動生成において，従来おのおののモジュールごとに設けた入力条件がすでに満たされているかどうかで行っていたチェック法を，でき上がったプログラムに対するペトリネットモデルを用い，その解析を行うことでシステムの実行可能性を検証する方法について検討を行った．これは，システム全体の安全性や可動性について数学的に検証することができるなど多くのメリットがある．

今回は，非常に単純な順次処理について検討を行ったが，複雑なシステムについても同様の手法が利用できるはずである．また，この検証法が十分有効な手法であることが確認できたため，今後はこの検証法を開発中の MAPP に組み込み，自動生成により作られたプログラムを MAPP 内で自動的にチェックして行けるよう改良を進める必要がある．さらに，非常に大きなシステムをモデル化していった場合の状態爆発の問題などについても検討していく必要がある．

参考文献

[1]村田：”ペトリネットの解析と応用”, 近代科学社, 1992

[2]恐神,西田：”プロログによるモジュールの検索とプログラムの合成”, 福井工業大学研究紀要第 23 号, pp.313-320(1993.3).

[3]恐神,西田：”入出力条件によるプログラムの合成”, 情報処理学会第 52 回全国大会, pp.5-47-48(1996.3)

[4] Osogami, Nishida: "A Method of Automatic Program Designing and Soft and Code Generation using Informal Procedure Call Sentences", Proc. of IASTED -ASC'98, pp.161-164, May 1998.

[5]恐神：”プログラム生成におけるリスト処理を用いた入出力条件のチェック”, 福井工業大学研究紀要第 29 号, pp.289-296(1999.3).

[6]恐神：”プログラムの自動生成における入出力条件について”, 電子情報通信学会 FIT2006 第 5 回情報科学技術フォーラム講演論文集, B-023, pp.121-122.

（平成19年３月22日受理）