5N−4 並列論理型言語を用いた構造化分析法によるLSI仕様記述・検証法の検討*

長沼 次郎　小倉 武†
NTT LSI研究所‡

1 はじめに

近年、LSIの大規模化、複雑化に伴い、設計TATも増大している。設計TAT増大の一因は、LSI設計の流れの最上流に位置する仕様記述、理解の不完全性、困難性にある。このため、我々はLSI設計の短TAT化を図るため、構造化分析法に基づくLSI仕様記述、検証法の検討を進めている[1]-[3]。

本稿では、抽象度が高く、かつ、並列動作を含む仕様を検証するため、動的検証用のプログラム言語として並列論理型言語(FGHC)を用いることを検討した。その結果、仕様記述法にいくつかの制約を与えることにより、構造化分析手法による仕様記述からシミュレーション可能なFGHC記述を生成し、これによる動的検証が可能であることを示す。

2 システム概要

2.1 仕様記述・検証システムの全体構成

本仕様記述・検証システムは構造化分析法を支援する既存の上流CASEツールであるSoft DA/SAツール[4]に動的検証ツールを付加した構成となっている[1]。ここでは、動的検証用のプログラム言語として並列論理型言語(FGHC[5][6])を用いる。構造化分析法で記述した仕様が有する逐次処理、並列処理、階層処理等をFGHCを用いて表現する。

2.2 並列論理型言語(FGHC)の特徴

FGHC等の並列論理型言語は、AND(ストリーム)並列をモデルとした言語で以下のような特徴がある。

並列論理型言語のプログラムはガード付き節の集合、

H :- G1,...,Gm | B1,...,Bn. m,n ≧ 0

で表される。"|"はコミット演算子、"G1,...,Gm"はガード部(FGHCでは組込み述語のみ)、"B1,...,Bn"はボディ部と呼ばれ、ゴール節はゴールの並び、

:- P1,...,Pn. n > 0

で表される。

並列論理型言語の実行は、ゴールが与えられると、まず節のヘッド部のユニフィケーションおよびその節のガード部の実行を並列に行う。この時、変数のアクセスモードの制約により中断(suspend)が起こる場合がある。中断しなかった節("|"まで通過した唯一の節)は、その節のボディ部の実行を並列に行う。中断した場合は、それ以外の実行可能なゴールが実行される。また、中断したゴールは、以降の実行過程で再開(resume)可能となれば実行できる。

このような並列論理型言語の中断、再開等の処理を用いて、対象となる動作仕様の(1)逐次処理、(2)並列処理、(3)階層処理等を実現する。

3 図形モデルと並列論理型言語記述の対応

3.1 図形モデルの意味定義

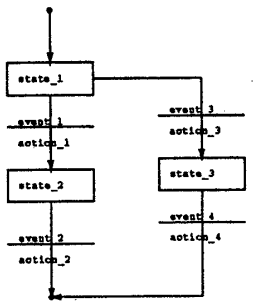
構造化分析法ではシステムの振舞いを簡単なシンボルを使ったデータフロー図で階層的に記述する[7]。仕様記述に用いるプロセス、データフロー等の各図形モデルを並列論理型言語記述と対応させている。状態遷移図とアクションロジックを用いた場合の対応関係を表1に示す。

表1: 図形モデルと並列論理型言語記述の対応

図形モデル	対応する並列論理型言語記述
状態遷移図	述語
状態名	状態変数の要素
状態遷移	状態変数の更新と述語呼出し
イベント	unification等(条件判断文)の条件部
アクション	アクションで示されるプロセスの起動

3.2 CASE上の各種動作仕様との対応

逐次動作　：　逐次に動作するプロセスは、CASE上では、状態の遷移に伴って順次起動するプロセスとして表現される。FGHCでは、変数のアクセスモードの制約を用いて、次の状態の動作を中断させる。前の状態のすべての動作が終了した時点で、次の状態の動作を再開させる。このような逐次動作は、図1に示すFGHCに対応付けることができる。




```
proc_n(....,Fin) :-
      true |
        proc_n_state(state_1,.....,Fin)
proc_n_state(final,....,Fin) :-
      true | ..., Fin = finished .
proc_n_state(state_1,.....) :-
      true | c_spec_state_1(State,.....)
        proc_n_state(State,.....) .
proc_n_state(state_2,.....) :-
      true | c_spec_state_2(State,.....)
        proc_n_state(State,.....) .
proc_n_state(state_3,.....) :-
      true | c_spec_state_3(State,.....)
        proc_n_state(State,.....) .
c_spec_state_1(State,..,event_1,..) :-
      true | action_1(....,Fin),
        next(Fin,State,state_2) .
c_spec_state_1(State,..,event_3,..) :-
      true | action_3(....,Fin),
        next(Fin,State,state_3) .
c_spec_state_2(State,..,event_2,..) :-
      true | action_2(....,Fin),
        next(Fin,State,final) .
c_spec_state_3(State,..,event_4,..) :-
      true | action_4(....,Fin),
        next(Fin,State,final) .

next(finished,State,Next) :-
      true | State = Next .
```

図1: 逐次動作と対応するFGHCの生成例

*High Level LSI Design Specification and Verification Based on Structured Analysis Method Using Concurrent Logic Programming Language

†Jiro Naganuma and Takeshi Ogura

‡NTT LSI Laboratories

並列動作 ： 並列に動作するプロセスは、CASE上では、同一の発火規則に持つアクションテーブル上の複数のプロセスとして表現される。FGHCでは、入力に関して変数のアクセスモードの制約の無い複数のプロセスとして実現する。だだし、出力に関しては、同一の変数(ファシリティ)に対するデータの衝突を回避するため、各々のプロセスの出力値を調べる。このような並列動作は、図 2に示すFGHCに対応付けることができる。

Actions	Invocation Processes				
	proc_1	proc_2	proc_3	proc_4	proc_5
action_1	1				1
action_2		1	1		1
action_3			1		1
action_4				1	


```
action_1(....,Fin) :-
     true |
       proc_1(....,01,Fin1), proc_5(....,05,Fin5),
       o_check([01,05]), f_check([Fin1,Fin5],Fin) .
action_2(....,Fin) :-
     true |
       proc_2(....,02,Fin2), proc_3(....,03,Fin3),
       proc_5(....,05,Fin5), o_check([02,03,05]),
       f_check([Fin2,Fin3,Fin5],Fin) .
action_3(....,Fin) :-
     true |
       proc_3(....,03,Fin3), proc_5(....,05,Fin5),
       o_check([03,05]), f_check([Fin3,Fin5],Fin) .
action_4(....,Fin) :-
     true |
       proc_4(....,04,Fin4), o_check([04]),
       f_check([Fin4],Fin) .

o_check(Output_Lists) :-
     "check whether each output is unifiable or not."

f_check(Finish_Lists,Finish) :-
     "check whether each process is finished or not,"
       Finish = finished .
```

図 2: 並列動作と対応するFGHCの生成例

階層的な動作 ： 階層的に記述されたプロセスは、CASE上では、任意の大きさのプロセス(サイクル数が未定)の起動として表現される。これは、論理型言語の階層的な述語呼出しとして容易に実現できる。

4 適用実験

4.1 対象の仕様記述

仕様記述・検証の実験の対象として、(1)通信制御プロセッサ(USART)[8]、(2) 簡単な8ビットCPU[9]を取り上げた。両仕様をSoft DA/SAツールで記述し、先に示した生成規則に従ってハンド・コーディングにより並列論理型言語(FGHC)に変換した。対象の仕様の記述量を表 2に示す。

4.2 生成実験

生成されたFGHCの記述は、USARTは約700行、8ビットCPUは約550行であった。この記述を用いて、動作の記号レベルでのシミュレーションを行った。この結果、対象が有する並列動作、逐次動作の実現を確認した。

表 2: 対象の仕様記述量

	データ 制御フロー	制御仕様	プロセス
USART	6	5	21
8-bit CPU	29	28	28

4.3 プログラム診断の適用

CASEの記述を表す論理型言語(Prolog)は、プログラム診断手法により効率的に仕様の検証を行うことが可能である[3]。並列論理型言語の場合、その実現はより複雑となるが[10]、基本的には、Prologの場合と同様に適用できる。この場合、(1)誤った答えを返す場合、(2)デッドロックになる場合を取り扱うことができる。これにより、抽象度の高い並列動作を含むハードウェアの動作仕様の誤りをアルゴリズミックに、効率的に発見することが可能となる。

5 おわりに

抽象度が高く、かつ、並列動作を含む仕様を検証するため、動的検証用のプログラム言語として並列論理型言語(FGHC)を用いることを検討した。その結果、仕様記述法にいくつかの制約を与えることにより、構造化分析手法による仕様記述からシミュレーション可能なFGHC記述を生成し、動的検証が可能であることを示した。今後、各種の仕様記述に適用し、仕様記述法、生成規則の完成度向上を図るとともに、CASEデータベースから並列論理型言語の自動生成等を検討する。またプログラム診断手法の適用効果を定量化する。

参考文献

[1] 長沼他, “論理型言語を用いた構造化分析法によるLSI仕様記述・検証法の検討,” 第44回情処全大, 6E-5, Mar. 1992.

[2] 長沼他, “構造化分析法を用いたLSI仕様記述・検証法の検討,” 情処学会DAシンポジウム '92, Aug. 1992.

[3] 長沼、小倉, “論理型言語のプログラム診断手法を用いたLSI仕様検証法の検討,” 第45回情処全大, 4K-3, Oct. 1992.

[4] 磯田他, “設計情報とコードの一体管理方式に基づくソフトウェア開発支援システム(Soft DA/SA),” NTT R&D, Vol.38, No.11, 1989.

[5] K.Ueda, “Guarded Horn Clauses,” ICOT Tech. Rep. TR-103, July 1985.

[6] 淵他, “並列論理型言語GHCとその応用,” 共立出版, 1987.

[7] D.J.Hatley and I.A.Pirbhai, “リアルタイムシステムの構造化分析,” 日経BP社, 立田監訳, 1989.

[8] “Microcommunications Handbook,” Intel, 1988.

[9] “論理設計CADに関する調査報告書,” 日本電子工業振興協会, 1986.

[10] A. Takeuchi, “Algorithmic Debugging of GHC programs and its Implementation in GHC,” ICOT Tech. Rep. TR-185, 1986.