4K−3

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*

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][2][3]。本稿では、動的検証用のプログラム言語として論理型言語を用い、それを前提として、論理型言語の数学的性質を利用したアルゴリズミックなデバッグ手法(プログラム診断手法)を仕様の検証に適用することを検討したので報告する。この方法により、仕様記述の誤りをシステムとユーザとのインタラクションにより、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。

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[2]。動的検証用のプログラム言語として論理型言語(Prolog[5]、GHC[6]等)を用いる。対象の仕様を表現している論理型言語を以下に示すプログラム診断手法を用いて効率的にデバッグを行う。

2.2 論理型言語のプログラム診断手法

論理型言語の数学的性質を利用したアルゴリズミックなデバッグ手法(プログラム診断手法)が研究されている[7][8]。この手法は、バグの位置を発見するアルゴリズムを内蔵したシステム(デバッガ)がユーザに質問を発し、ユーザがそれに答を与えるという過程の繰返しで行われる。システムの発する質問は実際に行われた計算の履歴の中に現れる局所的な計算の正しさに関するものであり、ユーザはその質問に対して、それが自分の意図した結果かどうかを答える。システムはユーザの応答に応じてバグを絞り込み、有限回の質問応答の後、バグの位置を決定する。このようなシステムは、論理型言語のメタインタプリタの機能を用いて比較的容易に実現できる。

構造化分析法で記述された対象の仕様から論理型言語を生成し、その生成された論理型言語に対しプログラム診断手法を適用する。システムにより示されるバグを含む論理型言語の対応する位置(特定の節)から、元の構造化分析法のデータ制御フロー図、制御仕様書等の誤りを関連付けることができる。これにより、仕様の誤りを論理型言語のプログラム診断手法を用いて発見することが可能となる。

*High Level LSI Design Verification Based on Program Diagnosis Method in Logic Programming Languages

†Jiro Naganuma and Takeshi Ogura
‡NTT LSI Laboratories

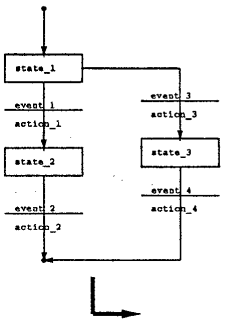
3 図形モデルと論理型言語記述の対応

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[9]。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論理型言語記述と対応させている。ここでは、制御仕様の代表的な状態遷移図とアクションロジックを用いた場合の生成規則を示す。

論理型言語では、状態遷移を伴う制御仕様で表される動作を、状態変数を引数に持つ述語、状態変数への次状態名のユニフィケーション(代入)、再帰的な述語呼出しを用いて実現する。このような論理型言語の生成規則とデータ制御フロー図との対応関係を表1に示す。また状態遷移図とそれを表す論理型言語の生成例を図1に示す。

表1: 図形モデルと論理型言語記述の対応

図形モデル	対応する論理型言語記述
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




```
test(....) :-
     State = state_1,
       test_state(State,....)
test(final,....) :-
     ! .

test_state(state_1,....) :-
     !, c_spec_state_1(State,....),
       test_state(State,....) .
test_state(state_2,....) :-
     !, c_spec_state_2(State,....),
       test_state(State,....) .
test_state(state_3,....) :-
     !, c_spec_state_3(State,....),
       test_state(State,....) .

c_spec_state_1(State,..,event_1,..) :-
     !, action_1(....),
       State = state_2 .
c_spec_state_1(State,..,event_3,..) :-
     !, action_3(....),
       State = state_3 .
c_spec_state_2(State,..,event_2,..) :-
     !, action_2(....),
       State = final .
c_spec_state_3(State,..,event_4,..) :-
     !, action_4(....),
       State = final .
```

図1: 状態遷移図と論理型言語の生成例

4 プログラム診断手法の適用実験

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)[10]、(2) 簡単な 8 ビット CPU[11] 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論理型言語 (Prolog) に変換した。対象の仕様の記述量を表 2に示す。

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

4.2 プログラム診断の適用

各対象の仕様に対し、以下のような誤りを挿入し、それをプログラム診断手法で検出した。

(1) 通信制御プロセッサ (USART)
初期化プロセスの中のコマンド書き込みプロセスにおいて、正しくは入力 8 ビットをコマンドレジスタに書き込むが、誤りでは入力 8 ビットをモードレジスタに書き込んだ。

(2) 簡単な 8 ビット CPU
オペランドフェッチプロセスにおいて、正しくは、プログラムカウンタ値を用いてプログラム RAM をアクセスし、その内容をオペランドレジスタに格納するが、誤りではプログラムカウンタ値をそのままオペランドレジスタに格納した。

これらの誤りを挿入した Prolog 記述に対し、プログラム診断手法を用いたデバッグの様子 (USART の例) を図 2に示す。システムの発する質問 ("write_mode"、"send_mark" 等のプロセスの入出力関係) に対し、ユーザはそれが自分の意図した結果かどうかを答える (true または false)。このような繰り返しの後、システムはバグを含む Prolog プログラムの対応する位置 (特定の節) を提示する (この場合 "write_command" プロセス)。

上記実行における Prolog の全トレース数と誤り発見までにシステムが発した質問数を表 3に示す。

表 3: 全実行トレース数とシステムからの質問数

	全トレース数	質問数
USART(initialize)	49	4
8-bit CPU	149	15

表 3に示すように、全トレース数に比べて、極めて少ないシステムからの質問に答えるだけで、システムによりバグが発見される。このように、従来の単純な実行トレースに比べ、仕様の誤りをアルゴリズミックに、かつ効率的に発見することが可能となる。

5 おわりに

LSI仕様を対象とした仕様の検証法に関し、論理型言語のプログラム診断手法を仕様の検証に適用した。本手法の適用により、構造化分析法で得られた仕様記述の誤りをシステムとユーザとのインタラクションにより、アルゴリズミックに、効率的に発見することが可能であることを示した。今後、本手法適用の効果を他の手法との比較検討により定量化していく。また、論理型言語の自動生成、並列動作を含む仕様の検証を行うための並列論理型言語への展開等を検討する。

```
| ?- backtrace((data(In),initialize(_,In,_,_)),CE).
write_mode(                              % <- システムからの質問 (1)
        [_400,_402,_404,_406,_408,_410,_412],
        [1,1,1,1,1,1,1,1],_366,
        [_400,[1,1,1,1,1,1,1,1],_404,_406,_408,_410,_412])
True or False ?
|: true.          % <- ユーザの答え (1)

send_mark(                               % <- システムからの質問 (2)
        [_400,[1,1,1,1,1,1,1,1],_404,_406,_408,_410,_412],
        _373,1,
        [_400,[1,1,1,1,1,1,1,1],_404,_406,_408,_410,_412])
True or False ?
|: true.          % <- ユーザの答え (2)

c_spec_wait_mode(                        % <- システムからの質問 (3)
        complete_mode,
        [_400,_402,_404,_406,_408,_410,_412],
        [1,[1,1,1,1,1,1,1,1],1,1,0],[1,0],
        [_400,[1,1,1,1,1,1,1,1],_404,_406,_408,_410,_412])
True or False ?
|: true.          % <- ユーザの答え (3)

write_command(                           % <- システムからの質問 (4)
        [_400,[1,1,1,1,1,1,1,1],_404,_406,_408,_410,_412],
        [0,0,0,0,0,0,0,0],_725,
        [_400,[0,0,0,0,0,0,0,0],_404,_406,_408,_410,_412])
True or False ?
|: false.         % <- ユーザの答え (4)


CE = write_command(.....) :- [...]=[...], [...]=[...] .
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

yes              % バグを含むプロセス
| ?-
```

図 2: USART のデバッグの様子

参考文献

[1] 松田, 小倉, "LSI 仕様記述と仕様検証の一手法について," 情処学会 DA シンポジウム '91, 論文集, pp.33-36, Aug. 1991.

[2] 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.

[3] 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.

[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] W.F.Clocksin and C.S.Mellish, "Programming in Prolog," Springer-Verlag, 1981.

[6] K.Ueda, "Guarded Horn Clauses," ICOT Tech. Rep. TR-103, July 1985.

[7] E.Y.Shapiro, "Algorithmic Program Debugging," The MIT Press, 1983.

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

[9] 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.

[10] "Microcommunications Handbook," Intel, 1988.

[11] "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.