5U-2

Tokioによる論理回路の検証 1
—高速命題論理検証系の実装—

中村　宏・藤田　昌宏*・田中　英彦・元岡　達
東大工学部　*現在富士通研

1. はじめに

我々は、時相論理（Temporal Logic）[1, 2]を用いた論理設計検証手法を提案し、論理型プログラミング言語Prolog を用いた実験システムを作成してきた。これは、命題論理の範囲で仕様を記述し、時相論理の決定手続きを用いて、その仕様が設計を満たしていることを検証するもので、デジタルシステム内の制御部を対象にしている。[3]

しかし、このシステムはスピード，及びメモリ所要の点で必ずしも満足のいくものではなかった。ところで、実際に推論を行う部分のアルゴリズムは極めて簡単である。そこで、論理式を積和形で表すカバー表現を導入し手続き型言語で扱いやすくし、処理の高速化を図ったので報告する。

2. 検証系の構成

検証系は、図1のような構成をしている。HSLは、ハードウェア記述言語、Tokioは時相論理に基く言語である。斜線部はProlog で、それ以外はC及びYacc , Lexを用いて実装する。

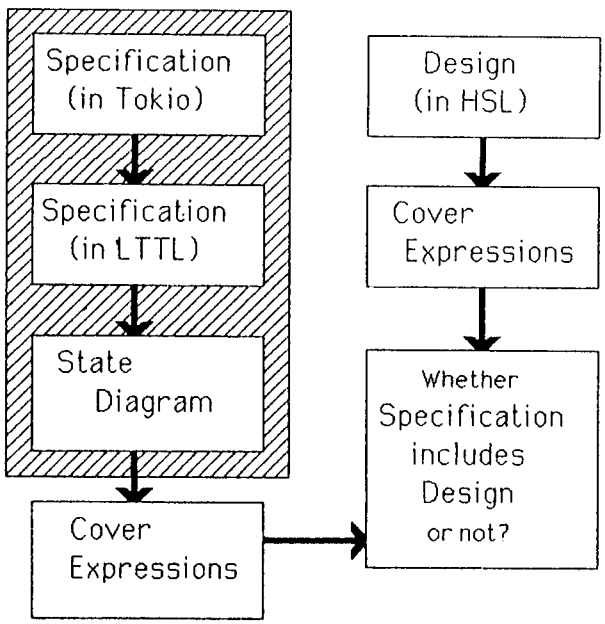

図1　検証系の構成図

3. Tokioと検証系

Tokio[4, 5]はInterval Temporal Logic（ITL）[2]を実行する言語で、まずLinear Time Temporal Logic（LTTL）[1]に展開される。

LTTLには4つの時相演算子○, □, ◇,　があり、各々以下のような意味を持っている。

○P：次の時刻にPが成り立つ。
□P：現在より考えて全ての時刻でPが成り立つ。
◇P：現在から考えていつかはPが成り立つ。
P　Q：現在から考えてQが成り立つまではPであり続ける。

LTTLから状態遷移図へはLTTLの決定手続きを用いて展開する。例えば□（P→○Q）は、

□（P→○Q）＝（P→○Q）　○□（P→○Q）
＝（～P　（P　○Q））　○□（P→○Q）
＝（～P　○□（P→○Q））
　　（P　○Q　○□（P→○Q））

となり、次の時刻に対する条件は□（P→○Q）又はQ　□（P→○Q）であり、図2のような状態遷移図を得る。

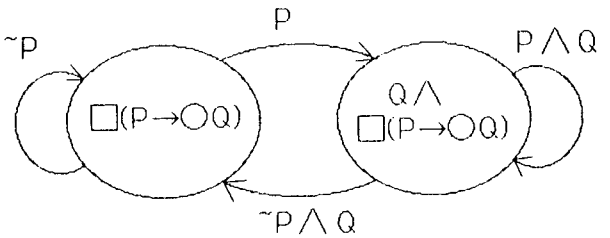

図2　状態遷移図

4. カバー表現を用いた論理設計検証[6]

ここでは、検証対象回路を図3のように分けた組み合わせ回路部分のシミュレーションを、カバーを用いて計算することで高速に検証を行う手法を述べる。

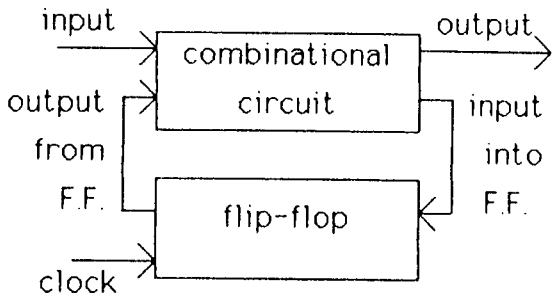

図3　検証対象回路の構成

・キューブとカバー

キューブは1つの積項を、カバーは積和形で表された論理式を表現する。n 入力m 出力の1つの積項 p に対し、大きさが2ビットの要素n 個と大きさ1ビットの要素をm 個を持つベクタがキューブである。そして与えられた論理式を積和形に変形しキューブの集合として表現したものがカバーである。

例　$f1 = AB + \overline{B}C + AC$

$f2 = \overline{B}C + \overline{C}\overline{D}$

という論理式は、

A	B	C	D	f1	f2	
01	01	11	11	1	0	AB
11	10	01	11	1	1	$\overline{B}C$
01	11	01	11	1	0	AC
11	11	10	10	0	1	$\overline{C}\overline{D}$

という4つのキューブの集合からなるカバーで表現される。

・onカバーとoff カバー

ある出力変数に対してそれを1とするような入力変数の論理式をonカバーといい、0とするような入力変数の論理式をoff カバーという。

カバー表現を用いた検証には順方向推論と逆方向推論が考えられるが、ここでは順方向推論を考える。アルゴリズムは次のようになる。

（1）検証すべき回路のうちの組み合わせ回路の部分のonカバーとoff カバーをCon, Coff とし、初期条件をCcondとする。またフリップフロップの接続から、出力変数の現在の値と入力変数の次の値の間の関係Rを求める。

（2）状態遷移図SD上の現在の状態を初期状態にする。

（3）SD上で現在の状態から1つ状態遷移し、ループか否かを調べ（即ち前と同じ状態になったか否かを調べる。ただし、状態とはSDの状態と検証対象回路のフリップフロップの内部状態を併せたものである。）、ループなら誤設計でありその状態遷移列が反例になる。ループでなければ遷移条件を求め、それをCt とする。もし、現在の状態からの状態遷移を全て処理した場合には、1つ前の状態にもどり調べる。全ての状態を調べた場合には、設計は正しいとして終了する。

（4）Con・Ct ・Ccond, Coff ・Ct ・Ccond（”・”はカバー同士の積を表す）をそれぞれ計算する。もしある出力変数に対する要素の値が1となるキューブがどちらの結果にもないときは、（3）に戻る。Con・Ct ・Ccondのみに1となるキューブがない時は、その出力変数の値を0とし、Coff ・Ct ・Ccondのみに1となるキューブがない時は、その出力変数の値を1とする。

（5）得られた出力変数の値から、Rを用いて次の時刻の各入力変数の値に対する条件をキューブCcondとして求め（3）へ行く。

このアルゴリズムは行列計算が大部分を占め、手続き型言語での実装が容易である。

この部分はC言語で約 400行程度になる。

5．おわりに

論理式をカバーという積和形で表すことにより、手続き型言語での実装が容易になり、処理の高速化を実現できることを示した。今後は、仕様を状態遷移図に変換する部分についても検討していきたい。

6．参考文献

[1] Z. Manna：”Verification of Concurrent Programs，Part 1，The Temporal Framework”，Stanford Univ.Report STAN_CS_81_836 (1981)

[2] B. Moszkowski：”Reasoning about Digital Circuits”，Stanford Univ. Report STAN_CS_83_970 (1985)

[3] M. Fujita：”Logic Design Assistance with Temporal Logic”，IFIP 7th CHDL (1985)

[4] 青柳：”Tokioかける言語”，Logic Programmimg Conference 8.2 (1985)

[5] 河野：”時相論理型言語Tokioの実装”，Logic Programming Conference (1985)

[6] 藤田：”時相論理を用いた論理設計検証およびテスト生成のカバー表現に基づく高速化”，電子通信学会　FTS研究会　Nov. (1985)