2J-7

時相論理型言語Tokio のインタプリタ

河野　真治・藤田　昌宏・田中　英彦・元岡　達
*
東大工学部　*現在　富士通研

1）階層的論理回路設計支援

論理回路設計に際し，仕様記述から，デバイスレベルまでの各段階でのサポートが望まれる。時相論理は，階層的なハードウェア記述，及びそのサポートとしての検証，合成系の十分な基礎を与えることができる〔1〕。Tokio は，この時相論理を直接に実行する言語である。Tokio は各段階のハードウェア記述を一貫して行うことができ，その仕様記述を実行し，シミュレーションする。

Tokio の記述は，上位ではアルゴリズムまたは宣言的な仕様である。これらは全て段階的にTokio による状態遷移表現に展開される。それらを，適切な状態遷移表に直すことにより，C-MOS ゲートアレイ等に合成することはすでに報告されている〔2〕。

2）Tokio の論理

Tokio の基づく論理は，ITL （Interval Temporal Logic ）〔3〕である。 ITLは，離散的な時間とその集合である時区間により真理値の定まる論理である。ITLの基本的な演算子は，@ (next) ，# (always) ，&& (chop) の３種ある。@ｐは次の時刻でｐが成立すること表す。#ｐは，ｐの属する時区間のすべての時刻でｐが成立することを表す。　ｐ&&ｑは，この時区間をふたつに分割し，前半でｐが成立し，後半でｑが成立することを示す。これらは，時区間を表す変数Ipを導入することにより，以下の様に定義される。

$$
\begin{aligned}
&Ipq \models P \&\& Q \equiv \\
&\quad \exists Ip, Iq \quad Ipq = Ip + Iq \\
&\quad \wedge Ip \models P \wedge Ip \models \text{not empty} \\
&\quad \wedge Iq \models Q \\
&Ip \models @P \equiv \\
&\quad \exists I0, I1 \quad Ip = I0 + I1 \\
&\quad \wedge I0 \models \text{length}(1) \wedge I1 \models P \\
&Ip \models \#P \equiv \\
&\quad \forall I0, I1 \quad Ip = I0 + I1 \\
&\quad \rightarrow I1 \models P \\
&Ip \models \text{keep}(P) \equiv \\
&\quad \forall I0, I1 \quad Ip = I0 + I1 \\
&\quad \rightarrow (I1 \models \text{not empty} \rightarrow I1 \models P) \\
&Ip \models \text{fin}(P) \equiv \\
&\quad \exists I0, I1 \quad Ip = I0 + I1 \\
&\quad \wedge I1 \models \text{empty} \wedge I1 \models P
\end{aligned}
$$

Fig.2 Formal Definition of Tokio

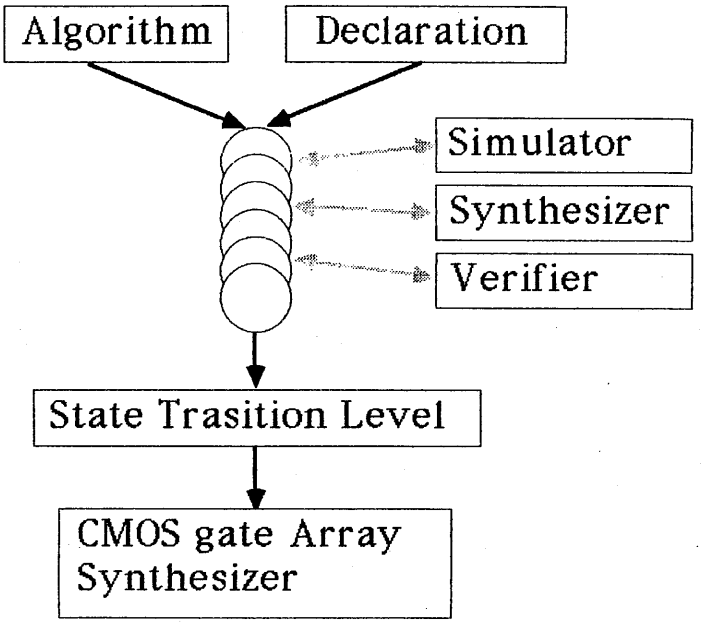

Fig.1 Hierarchical Logic Design

3）Tokio の実行

Tokio は，ITL をLTTL (Linear Time Temporal Logic) 〔4〕に展開しつつ実行していく。Tokio の実行は次の三っの部分よりなる。

・未来への縮退
・時相論理変数の単一化
・時区間の分割

3-1 ）未来への縮退

LTTLは，時区間の概念がなく，nextとuntil の二つが主要な演算子である。LTTLは，各時刻ごとに導出を行うことにより検証することができる。Tokio は，ITL をこのような方法により展開して行く。現在の時刻で縮退した論理式のなかに，@や#などがあると，それらは次の時刻に成立すべき式を生成する。これを未来への縮退とよぶ。ITL での時区間は,LTTL 上で，その時区間の終了の時刻と，その時区間中で成立する式と，その時区間の終了以降に成立すべき式の三組により表現することができる。

これらの展開は，時間を含まない式ではPrologと全く同一である。Tokio は現在時刻での縮退がすべて失敗すると過去へバックトラックする。

3-2）時相論理変数の単一化

Tokio の変数は，各時刻で異なる値をもつ。Tokio はこの複数の値に対し，一つの時刻での単一化と，ある時刻以降の全ての時間での単一化の２種類の単一化を行う。この単一化に対し次の二つの実装が考えられる。

a）時相論理変数は，その時刻の値と次の時刻の値の二つの変数で表現される。全時間に対する単一化は，各時刻での展開時に行う。

eq（A，B）　－－＞　　＃（A＝B）．

b）時相論理変数を，ストリームにより表す。

$t-$t-$t-D

A　B　C

このとき，A，B，Cは，各時刻での値を表し，Dは，以降の値の集合を表している。$t はストリームのノードを表す。

b のアルゴリズムは，構造体の単一化の際にやや複雑である。この方法は，Dのような変数が多く出現する場合，つまり時相演算子が少ない場合に，高速に動作するという利点がある。またａには不可能である変数の値の未来への先読みができる。ａでは，全時間での単一化の手間は，常に通常の単一化の倍である。ａでは構造体の生成がないためにメモリ効率はｂより良い。

この二つの方法についてインタプリタを作成したが，速度の差は殆どない。また構造複写の実装をするときは，単一化の手間の小さいｂの方法がより効率的である。

Tokio の変数と，過去へのポインタがなく，また，次の時刻で必要な変数は必ず現在時刻で，一度参照されると言う特徴がある。この時参照と同時に別な領域へのコピーを行うことにより，簡単にガーベジコレクションを行うことができる。このとき，過去へのバックトラックは禁止されるが，仕様のシミュレーションの様な場合は，アルゴリズムは決定的なはずであり問題ない。過去へのバックトラックは，検証などの用途のための機能である。

3-3）時区間の分割

時区間は＆＆により任意に分割される。この分割は，全体の時間が最も短くなる様に，バックトラックにより調整される。

ITL は時区間の最後でのみ成立するfin と最後以外で成立するkeepの二つの演算子を持つので，時区間が確定するまでそれらの実行は停止する必要がある。これらは fin queue，keep queue により実現される。したがって Tokio は，次の時刻で実行されるnext queueとあわせて３種のキューを持つ。

4）構造共有による単一化

Cによるインタプリタでは，ｂ）の方法を構造共有により実現している。変数は環境とフレームの二つの組で表される。時相論理変数のノードは，フレームへのポインタのタグとして表される。構造体とこのノードの間で単一化が起きた場合は，まず構造体の環境が現在と未来用に二つ生成される。元の構造体の変数は時相論理変数のノードとなる。その現在の値の部分は，生成された現在の環境を指し，未来の値の部分は，未来の環境を指す。単一化される時相論理変数側でも同じ代入が起きる。この時，ここでの変数のフレームはすべて元の構造体のフレームを指している。この操作は構造体が異なる複数の環境よりなる場合は再帰的に行われる。

5）おわりに

Tokio の記述例〔５〕は，ハードウェア記述の各段階に対応している。またコンパイラも並行して製作している〔６〕。

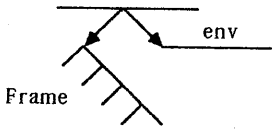

Fig 3.1 Structure Share Representation

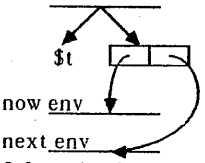

Fig 3.2 node of temporal variable

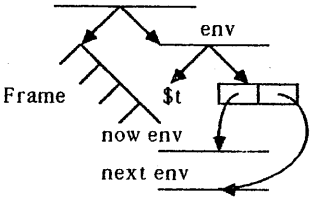

Fig 3.3 now and next environment in unification of structure

参考文献

〔１〕M.Fujita，New Generation Computing，Vol.1,No.2,pp.195-203 1983

〔２〕藤田，石曽根，Proc. of the Logic Prog. Conf.，ICOT，'85 13.1

〔３〕B.Moszkowski，IEEE Computer Magazine，February 1985

〔４〕P.Wolper，22nd Annual Symposium on Foundation of Computer Science，October　1981

〔５〕情報処理学会第31回全国大会，2J-9 (1985)

〔６〕情報処理学会第31回全国大会，2J-8 (1985)

〔７〕河野，Proc. of the Logic Prog. Conf.，ICOT，'85 8.2

〔８〕青柳，Proc. of the Logic Prog. Conf.，ICOT，'85 8.1