3X-4

# RTL-Tokio 記述からの制御系の抽出

中村 宏　　久木元 裕治　　田中 英彦

東京大学工学部

## 1 はじめに

我々は従来より時相論理型言語 Tokio、RTL-Tokio を用いた論理設計支援を目指している [4]。我々が目指す論理設計支援システムの全体構成は図 1のようである。この構成図のうちシミュレータ、データパス検証部、及び制御系検証システムが現在までに完成している。本稿では、データパス検証部における制御系の抽出について述べる。

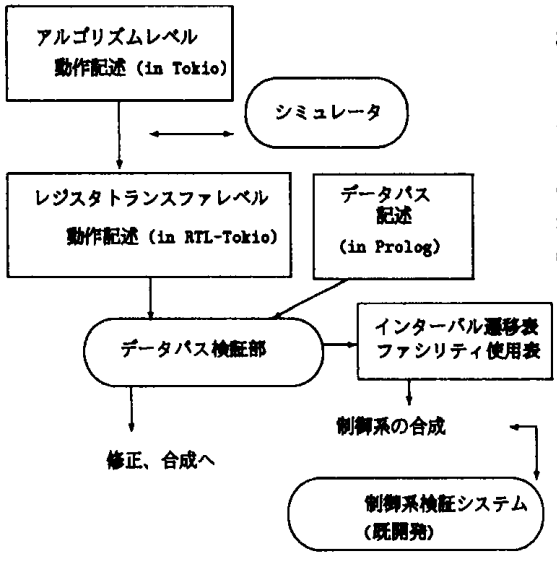


図 1: 論理設計支援システム

## 2 Tokio と RTL-Tokio

Tokio[1] は時区間時相論理に基づくハードウェア動作記述言語であり、RTL-Tokio は Tokio に対し、'レジスタ間転送を長さ 1のインターバルで行ない、バックトラックをしない'という制限を加えた [5] レジスタトランスファレベル動作記述言語である。

一般に、アルゴリズムレベル記述からレジスタトランスファレベル記述を導出する際には、演算子 (operation) の scheduling、それに伴うレジスタあるいは演算器 (operator) 等のデータパスの allocation が行なわれる。さらに、付随して制御系の状態遷移が決定する。

RTL-Tokio の記述においては、レジスタとメモリ、それらのデータに対する演算とその時間関係が陽に記述されているが、個々の演算を実現する演算器とデータ転送路に関しては宣言されていない。動作記述中の演算子とそれを実現する構造 (facility) との対応は、データパス検証部が自動的にとる。

## 3 データパス検証部における制御系の抽出

データパス検証部 [2] は、まず RTL-Tokio の各サブインターバル ('&&' で区切られたインターバル) に固有に Id をつける。この Id は、predicateId(headName および arity)、clauseNumber(同一 predicate での何番目の clause か)、intervalNumber(clause 中で何番目のインターバルか) から subintId: (predicateId,clauseNumber,intervalNumber) として作られる。

次に、固有の Id を持つ各インターバル間の遷移関係をインターバル遷移表として得る。この表は、'&&' 演算子による遷移関係と predicateCall による遷移関係のみを考慮すれば良い。

最後に、各インターバルにおける演算と構造記述中のファシリティを対応付け、ファシリティ使用表を導出し、データパスの二重使用を検出する。

ここで、各インターバル Id と状態を 1 対 1 に対応させることにすれば、インターバル遷移表とファシリティ使用表が、そのまま制御系の情報になる。そのために、データパス検証部が受理する RTL-Tokio は以下の構文に制限されている。

**データパス検証部が受理する RTL-Tokio**

```
head(arg) :- localConds,!,actions.
head(arg) :- localConds,!,actions && actions
             && ..... && predicateCall.
```

(ただし、actions は、predicateCall を含まないデータ転送)

これは即ち、predicate-call は必ず最後のサブインターバルに限ることを意味する。この条件を満たしていない場合、各インターバル Id と状態を 1 対 1 に対応させると以下のような問題が起こる。

---

[0]Derivation of Control Part from RTL-Tokio
Hiroshi NAKAMURA, Yuji KUKIMOTO, and Hidehiko TANAKA
Faculty of Engineering, University of Tokyo

RTL-Tokio でない例

(ア) 式

```
pred1 :- cond1, !, action12 && pred3 && action13.
pred2 :- cond2, !, action21 && pred3 && action23.
pred3 :- !, action3. % 簡単のため、条件分岐は無し
```

を考える。この記述では、pred1 の中で呼ばれる pred3 と、pred2 の中から呼ばれる pred3 は、その後 action13 に遷移するか action23 に遷移するかが違うので、別の状態と考えるべきである。しかし現在のデータパス検証部では、action3 には、(pred3/0,0,0) という一つの subintId しか付かず、どちらから呼ばれた pred3 も同じ状態で同じファシリティを使うと判断する。

構文上の制限を設ける場合には、その構文で必ず全ての記述ができるかという点が問題になる。まず、(ア) 式の action3 を、同じ状態と見なすか違う状態と見なすかを、RTL-Tokio の構文で明記できることを示す。

RTL-Tokio への変換

(1)(違う状態)(ア) 式に対応する RTL-Tokio

```
pred1 :- cond1, !, action12 && action3 && action13.
pred2 :- cond2, !, action21 && action3 && action23.
```

(2)(同じ状態)(ア) 式に対応する RTL-Tokio

```
pred1 :- cond1, !, action12, *reg <= 1 && pred3.
pred2 :- cond2, !, actoin21, *reg <= 2 && pred3.
pred3 :- !, action3 && pred_call.
pred_call :- *reg1 = 1, !, action13.
pred_call :- *reg1 = 2, !, action23.
```

ここで、(1) では、(pred1/0,0,1), (pred2/0,0,1) という異なる subintId が 2 つの action3 に付き、異なる状態と見なされるが、(2) の記述では (pred3/0,0,0) という一つの subintId しか付かず、同一の状態と見なされる代わりに、flag 変数が一つ (*reg) 用意される。(1) と (2) の記述は、文献 [3] における、local slicing, global slicing にそれぞれ対応する。

次に、再帰的な predicate-call が途中のサブインターバルにある場合について、考える。

(イ) 式

```
pred :- cond, !, action1 && pred && action2.
pred :- !, action3.
```

この場合も同様に、何回再帰が繰り返されているかを数えるカウンタがあれば RTL-Tokio の構文 (tail-recursion に他ならない) に変換できる。この記述例では、以下のように一つのカウンタを用いれば良い。

```
pred :- cond, !, action1, *reg <= *reg + 1 && pred.
        % *reg は 0 に初期設定されているとする
pred :- !, action3 && pred_tail.
pred_tail :- *reg > 0, !, action2, *reg <= *reg - 1
       && pred_tail.
pred_tail :- !, true.
```

(イ) 式の様な記述は アルゴリズムレベルなら頻繁に現れる。従って、今述べた RTL-Tokio の構文への変換はアルゴリズムレベルの動作記述をレジスタトランスファレベルの動作記述に変換する段階の一つと考えられる。

## 4 おわりに

本稿では、RTL-Tokio に構文上の制限を加えることにより、制御系の抽出が容易に行なえることを述べた。この構文上の制限を満たすように動作記述を変更することを支援することが、アルゴリズムレベル動作記述からレジスタトランスファレベル動作記述の導出を支援することと考えられる。この部分の支援について今後考えていきたい。

**謝辞** 日頃御討論頂く富士通研究所藤田昌宏博士、ソニー CSL 研究所河野真治博士、及びシステムの実装を精力的にして頂いた第一勧業銀行中井正弥君に感謝致します。

なお、本研究は一部文部省科学研究費による。

## 参考文献

[1] M. Fujita, S. Kono, H. Tanaka, and T. Moto-oka. Aid to Hierarchical and Structured Logic Design Using Temporal Logic and Prolog. In *Prodeedings.Pt.E*, pp. 283–294. IEE, 1986.

[2] H. Nakamura, M. Fujita, S. Kono, M. Nakai, and H. Tanaka. A Data Path Verification System using Temporal Logic Based Language Tokio. In *IFIP WG10.2 Working Conference on the CAD Systems Using AI Techniques*. IFIP, June 1989.

[3] C.J. Tseng, R.S. Wei, S.G. Rothweiler, M.M. Tong, and A.K Bose. Bridge: A Versatile Behavioral Synthesis System. In *25th Design Automation Conference*, pp. 415–420. ACM/IEEE, 1988.

[4] 中村宏, 中井正弥, 藤田昌宏, 河野真治, 田中英彦. 時相論理型言語 Tokio による論理設計支援. In *Logic Programming Conference '89*. ICOT, 1989.

[5] 中村宏, 藤田昌宏, 河野真治, 田中英彦. RTL-Tokio: レジスタトランスファレベル記述言語. 第 37 回全国大会 *1U-3*. 情報処理学会, 1988.