3K-1　テンポラルロジックとProlog を用いた論理設計の検証－ＤＤＬ、ゲートレベル

藤田　昌宏　田中　英彦　元岡　達

（　東京大学　工学部　）

1.　はじめに

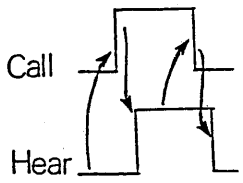
我々はすでに、テンポラルロジックによるハードウェア仕様記述から、ゲートレベル記述まで支援する検証システムに、Prolog を用いることを提案した［１］。

ここでは、ゲートレベルの検証にしぼり、ハンドシェイクを例にとって検証手法を説明する。

2.　仕様記述

タイミングチャートと、対応するテンポラルロジックの記述例を図１に示す。各演算子はそれぞれ、□：always、　▽：sometime、　Ｕ：until を表わす［２］。

例えば、Receiver の□（Call →▽Hear ）と□（～Hear →～Hear　Ｕ　Call ）で、Call が１になると、いつかHear が立ち上ることを示している。



(a) timing chart

□(～ Hear → ▽ Call)
□(Hear → ▽ ～Call)
□(～Call → ～Call U ～Hear)
□(Call → Call U Hear)
(Sender)

□(Call → ▽ Hear)
□(～Call → ▽ ～Hear)
□(～Hear → ～Hear U Call)
□(Hear → Hear U ～Call)
(Receiver)

(b) temporal logic

fig.1 handshaking sequencebetween Sender and Receiver

3.　検証プログラム

検証は時間に対して、前方推論（forward reasoning ）でも後方推論（backward reasoning）でも可能である。なお、前方推論については、与えられた仕様から自動的に検証プログラムを合成することも可能である［３］。本節では、前方推論について、□（A⊃▽B）を例にとって、検証プログラムを具体的に説明する。

□（A⊃▽B）を検証するには、全ての場合について、（A⊃▽B）を検証すればよい。背理法で行なうために、まず否定をとり、

～（A⊃▽B）＝（A∧～▽B）＝（A∧□～B）

（　～▽＝□～　公理より）

を得る。次に、A∧□～Bを満たすパスがないかを調べ、もしあれば反例として印刷する。推論を進めるに当って、前の状態と同じになったか（ループになったか）のチェックは、全てのフリップフロップ等の内部状態を持つ素子の状態が等しくなったかどうかで行なう。以下に、前方推論プログラムの流れを説明する。

① 初期状態ＩＮＩＴから始め、それがA∧～Bを満しているか調べる。満していれば、次の状態Nを得る。

② ループになるまで以下を繰り返す。

NがBを満たしているか調べる。もし満たしていれば、A∧□～Bではあり得ないので、強制的にバックトラックをかけ別のパスを調べる。もし、もうパスが他に無ければ検証を終了する。

Bを満たしていなければ、その次の状態を求める。

ループになったら③へ行く。

③ このパスはA∧□～Bを満たし、反例なのでそのパスを印刷し、また別のパスを調べるために強制的にバックトラック（Prolog ／KRでは、（false ））をかけ、②にもどる。

以上をProlog ／KRで記述したものが図２である。（ＳＴＴＲＡＮ　＊Ｐ　＊Ｎ）は、　現在の状態（＊Ｐ）と次の状態（＊Ｎ）の関係を表わしたものであり、ＤＤＬやゲート回路の記述から作る（図５、６参照）。（ＳＴＥＱＵ　＊Ｈ　＊Ｐ）は、現在の状態がそのパスの過去の状態（＊Ｈ）と等しいか調べるシステムプログラムであり、パターン照合機構により簡潔に記述されている。

また、（ＬＯＧＩＣＡ　＊ＩＮＩＴ）は、初期状態ＩＮＩＴが条件Aを満しているか調べるもので、

```
(ASSERT (VER *INIT) (LOGICA *INIT) (LOGIC~B *INIT) (STTRAN *INIT *N)
                    (LOGIC~B *N) (VER1 (*INIT) *N))
(ASSERT (VER1 *H *P)
     (IF (STEQU *H *P)
         (AND (PRINT (*H TYPE<> *P)) (FALSE))
         (AND (STTRAN *P *N) (LOGIC~B *N) (VER1 (*P . *H) *N))))
```

fig.2 forward reasoning program for □(A⊃▽B)

（LOGIC¯B *N）は、状態Nが条件～Bを満たしているか調べるものであり、ユーザが別に与える（図6参照）。このように、Prolog のもつ自動バックトラック機構と、強力なパターン照合能力により、プログラムが非常に簡単になっている。

4. ゲートのProlog による表現とその検証

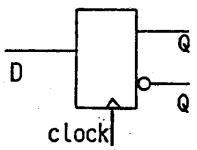
本節では、制御を明確に指定することもできるProlog /KR［4］を用いた、ゲート回路の表現法について述べる。以下、組み合せ回路には遅延がないとする。AND、NOT等の基本ゲートは、入力と出力の関係を表の形で記述され、図3のようにProlog ではゴールのみの形で表現できる。Prolog /KRでは、ASSERTを使ってプレディケイトを表現する。ASSERTの次にくる項がゴールで、それ以降の項が条件となる。ASSERT中、最後の項が出力で、それ以外の項が入力である。

```
(ASSERT (FAND 0 0 0))  (ASSERT (FNOT 0 1))
(ASSERT (FAND 0 1 0))  (ASSERT (FNOT 1 0))
(ASSERT (FAND 1 0 0))
(ASSERT (FAND 1 1 1))
```

fig.3 primitive gates descriptions in Prolog/KR

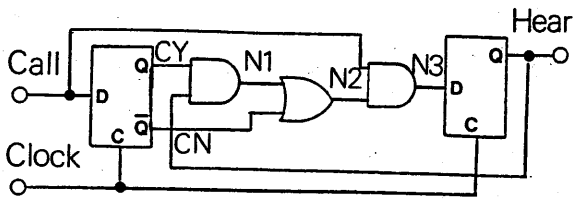
フリップフロップは、現在の内部状態と、次の内部状態の関係として表現される。D－フリップフロップのProlog /KRによる記述を図4に示す。ASSERT中、最初の項がD入力であり、次の2項が現在の内部状態Qと～Q、その次の2項が、次の内部状態Qn と～Qn 、そして最後の項がクロックである。従って最初のASSERTは、もし現在の内部状態がリセット（Q＝0、～Q＝1）で、かつクロックが0（クロックがこない）なら、D入力の値にかかわらず（*は変数）、次の内部状態はリセット（Q＝0、～Q＝1）となることを示している。

```
(ASSERT (FDFF * 0 1 0 1 0))
(ASSERT (FDFF 1 0 1 1 0 1))
(ASSERT (FDFF 0 0 1 0 1 1))
(ASSERT (FDFF * 1 0 1 0 0))
(ASSERT (FDFF 1 1 0 1 0 1))
(ASSERT (FDFF 0 1 0 0 1 1))
```



D-input Q Q Qn Qn clock

fig.4 D-flipflop in Prolog/KR



(a) circuit diagram

```
(ASSERT (PAULC (*CALL *HEAR *CY *CN)
               (*@CALL *@HEAR *@CY *@CN)
               *CL)
        (FDFF *CALL *CY *CN *@CY *@CN *CL)
        (FAND *CY *HEAR *N1) (FOR *N1 *CN *N2)
        (FAND *CALL *N2 *N3)
        (FDFF *N3 *HEAR *¯HEAR *@HEAR *@¯HEAR *CL))
```

(b) in Prolog/KR

fig.5 a gate level design of Paul

順序回路は最終的に、現在の値と次の値の関係として、Prolog で記述される。ハンドシェイクの受信側をゲートで設計した例を図5（a）に、Prolog に直したものを（b）に示す。このように回路中の配線の表現は、同じ変数（*で始まるアトム）名によって、つながっている各素子の端子を同じ値にすることによって行なう。図中、*の後に@があるものが次の時刻での値で、ないものが現在での値である。モジュール全体に対する入出力端子と、内部で使用するフリップフロップの状態をProlog での記述における外部への引数とする。このようなモジュール全体の記述は、回路の接続のみ与えられれば、自動的に合成することができる。最後にReceiverの回路について、□（Call ⊃▽～Hear ）の検証例を図6に示す。

5. おわりに

ゲートレベルの設計に対し、与えられたテンポラルロジックの仕様の検証手法について具体的に述べた。同じようにして、DDLも検証できる。

＊ 参考文献

［1］情処全大25回、3N－1

［2］情処、設計自動化研資、14－1

［3］電気学会、電子デバイス研資、EDD 83-38

［4］中嶋、“Prolog /KR Manual ”、METR82－4、東京大学


```
(ASSERT (STTRAN *P *N) (PAULC *P *N 1))  ; Verifying Sender
(ASSERT (LOGICA (1 *HEAR *CY *CN))   ; Call=1
(ASSERT (LOGIC¯B (*CALL 0 *CY *CN))  ; Hear=0

:(ver (*call *hear *cy *cn))      ; Start of Verification
(((1 0 1 0)) TYPE<> (*@CALL_3894 0 1 0))  ; Counter example
NIL
:
```

fig.6 verification example of gate-level : □(Call ⊃ ▽Hear)