3K-2

テンポラルロジックとProlog を用いた論理設計の検証－テンポラルロジックレベル

濱中　直樹　藤田　昌宏　田中　英彦　元岡　達
(　東京大学　工学部　)

1．はじめに

テンポラルロジックで記述された仕様は決定手続き[1]によって検証することができる。
Prolog を用いると比較的小さなプログラムによって実現でき、また検証プログラム自体の自動合成も容易で汎用性が高い。本稿ではテンポラルロジックで与えられた仕様記述の検証法の原理と例を示す。

2．レジスタと端子の扱い

本来テンポラルロジックには、DDL[2]等にあるレジスタ、端子等の属性はないが、レジスタに相当する命題にはレジスタの性質を表わす式をuntil 演算子を用いて記述する必要があり、レジスタが多い場合にはこの条件の考慮はユーザの負担になる。そこでレジスタ、端子という属性宣言を行ない、 until 演算子を持った変数のばたつきを禁止する式を挿入する等して、レジスタの性質を自動的に処理するようなシステムを考えている。

3．検証法－式の変形

図1は検証に必要なテンポラルロジックの恒等式[1]である。図の第1番目の式は、「いつもF」ということが「今Fでかつ次の時刻からもいつもF」であることを、表わしている。第2番目以降の式についても同様である。

仕様の検証は背理法による。検証したい式の否定をとり、仕様を記述した式に加え、図1の変形定理を用い、これらの式を“現在“に関する部分と“次の時刻“に関する部分に分ける。“次の時刻“に対しても同様な処理を行ない遷移ネットワークをつくる。全てのパスが矛盾に到るならば仕様は検証内容を満足し、ネットワークにループができれば仕様は検証内容を満足せず、このループが反例になる。

$$\Box F \equiv F \wedge \bigcirc \Box F$$
$$\nabla F \equiv F \vee \bigcirc \nabla F$$
$$F1 \cup F2 \equiv F2 \vee (F1 \wedge \bigcirc (F1 \cup F2))$$
$$\sim \bigcirc F \equiv \bigcirc \sim F$$
$$\sim \Box F \equiv \sim F \vee \bigcirc \sim \Box F$$
$$\sim \nabla F \equiv \sim F \wedge \bigcirc \sim \nabla F$$
$$\sim (F1 \cup F2) \equiv \sim F2 \wedge (\sim F1 \vee \bigcirc \sim (F1 \cup F2))$$

図1　式の変形

4．Prolog ／KR[3]による表現

図1をProlog ／KRで表現すると図2のようになる。Prolog ／KRのパターン照合機構により図2のような表明が自動的に選択されて評価される。

5．Eventuality の確認

遷移ネットワーク中のループのうち各節点に同じ∇Fという式があって、このFという式がループ中のどこでも実現されていない場合は∇Fはみたされておらず、反例ではないから取り除かねばならない。

```
(ASSERT(ALWAYS *f) *f (NEXT(ALWAYS *f)))
(ASSERT(SOMETIME *f)(OR *f (NEXT(SOMETIME *f))))
(ASSERT(UNTIL *f1 *f2)(OR *f2 (AND *f1 (NEXT(UNTIL *f1 *f2)))))
(ASSERT(NOT(NEXT *f))(NEXT(NOT *f)))
(ASSERT(NOT(ALWAYS *f))(OR(NOT *f)(NEXT(NOT(ALWAYS *f)))))
(ASSERT(NOT(SOMETIME *f))(NOT *f)(NEXT(NOT(SOMETIME *f))))
(ASSERT(NOT(UNTIL *f1 *f2))(NOT *f2)
                           (OR(NOT *f1)(NEXT(NOT(UNTIL *f1 *f2)))))
```

図2　Prolog ／KRでの恒等式の表現

6．検証例

［例１］

□（A→○B）∧Aという仕様に対して▽Bを検証してみよう。まず検証したいことの否定をとると、

$$\sim\nabla B \equiv \Box\sim B \quad \text{（公理による）}$$

となる。次に□（A→○B）∧A∧□～Bという式に対して遷移ネットワークをつくる。（図３）このネットワークでは全てのパスが矛盾に到るので背理法の仮定が否定され、▽Bが検証された。

［例２］ ハンドシェイク[4]（図４参照）

ハンドシェイクの仕様に対して

□（Call →▽～Hear ）

を検証してみた。検証したいことの否定は、

▽（Call ∧□Hear ）

である。これを仕様に加えてProlog ／KRの検証プログラムで検証すると、ＨＩＴＡＣ Ｍ 280Ｈの ＣＰＵ時間で13秒程度となる。

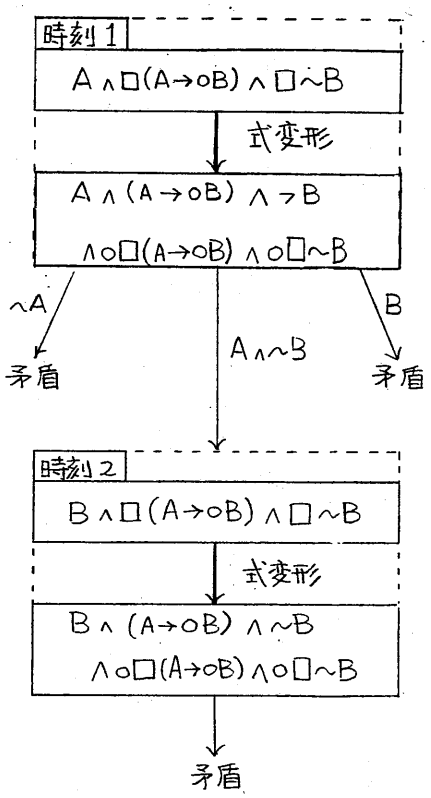
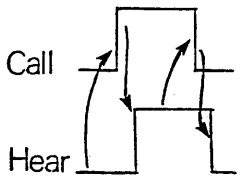


図３ □(A→○B)∧Aに対する▽Bの検証

7．おわりに

以上で、テンポラルロジックで与えられた仕様をProlog で検証できることが示された。ただし実用規模の検証の場合、検証時間が指数的に増大することが予想されるので処理効率の向上をはかることが必要である。

また、現在テンポラルロジックの式を変形する際に生じる遷移ネットワークが、与えられた式を満足する状態遷移を表わすことに注目して、この遷移ネットワークからＤＤＬ（Digital System Design Language ）の記述を合成[5]することについて検討している。簡単な例については自動合成を行なえることが確認できたことを付記しておく。



(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

図４ ハンドシェイク

8．参考文献

<1> Pierre Wolper : Temporal logic can be more expressive, 22nd Annual Symposium of Foundations of Computer Science, October 1981

<2> J. R. Duley, D. L. Dietmeyer: A digital system design language (DDL), IEEE Trans Computer, Vol. C-17, 1968

<3> 中島：Prolog ／KR マニュアル, METR82-4, 東京大学工学部

<4> 第２５回情報処理学会全国大会 ３Ｋ－１

<5> Z. Manna, P. Wolper : Synthesis of communicating processes from Temporal Logic specifications, Proceeding of Logics of Programs , 1981