時相論理による仕様記述支援システム

5S-6

藤田　昌宏　　藤沢　久典

富士通研究所

1.はじめに

仕様記述、論理検証および論理合成に関する様々なツールの開発を行なってきた。しかし、これらのツールを実際の設計に対して適用しようとすると、まだ次のような問題が残されている。

(1) 慣れていない設計者にとって論理的な仕様記述は必ずしも容易ではない。

(2) 論理検証および論理合成システムにとって、大規模回路を扱うことは困難である。

ここでは、これらの問題のうち(1)を解決する方法として、時相論理による仕様記述を支援するツールを開発したので報告する。

2. YUKO： 命題時相論理による仕様記述支援ツール

YUKOは命題時相論理による同期回路の仕様記述を支援するツールである（YUKOとは「有効」を意味している）．YUKOは大きく分けて、グラフィックエディタ、時相論理式ジェネレイタ、時相論理式シミュレータの3部からなる。以下、順に説明する。

グラフィックエディタ

グラフィックエディタはタイムチャートや時相論理式の入力を行なうだけでなく、YUKOシステムの各処理の制御も行なう。グラフィックエディタはSUNTOOLSおよびC言語を用いてSUNワークステーション上に実装されている。すべてのコマンドはPOP-UPメニューのなかからマウスで選択することによって実行される。

グラフィックエディタには、タイムチャート、時相論理式、およびシステムコマンド向けに三つのウィンドウが用意されている。タイムチャート用ウィンドウには、inputs, outputs, terminals および true/false 信号の四つのタイプの変数が用意されている。各状態において（ここでは同期回路のみを考えているため、状態はクロックに対応する）、各変数は0、1、またはX（不定を表す）のうちいずれか一つの値をとる。inputs および outputs は設計されるシステムへの入力と出力に対応する。teminals は内部変数であり、インターバルや状態に対応して値が決まる。複雑なタイミング関係は内部変数を用いることにより、はるかに簡単なタイムチャートで記述することができる。true/false信号は特殊な信号であり、ハードウェアのファシリティとの対応はない。true/false信号は記述されたタイムチャートがtrue か false かを示すものであり、起こってはならない状態といったものを表現するのに用いられる。

時相論理式ジェネレイタ

経験から、制御回路を記述するのに必要とされる時相論理式の数はそう多くなく、同じタイプの式が何度も現われそれらの組み合わせによって記述が成り立っている。そこでタイムチャートを解釈するうえでいくつかのルールを定義し、これらのルールに従って適当な時相論理式を生成するインターフェイスプログラムを開発した。こうして生成された時相論理式は元のタイムチャートに示された関係を満たすという意味で等しい。このジェネレイタを実装するに当たって、ルールベースによるアプローチをとった。というのは、生成ルールを追加するのが簡単なためである。実装には、Prolog を用いた。

ルールには２つの種類が存在する。一つは一般化／特殊化ルールであり、もう一つはインターバル分割ルールである。先に述べたように、タイムチャートは厳密な意味を持っているわけではない。ただ時間的な遷移の一部を表現しているだけであって全ての可能な場合を表しているわけではない。このことは、タイムチャートだけでは必ずしも適当な論理式を得ることはできず、ある程度十分な論理式を得るにはたくさんのタイムチャートが必要であることを意味しておる。そのためには、これらの関係をできるかぎり一般化しておく必要がある。例えば、Fig.1 に示されている関係について考えてみよう。request と answer の二つの信号間の応答時間は時相論理式では指定されていない。即ち応答は適宜行なわれる（時相論理については[1]を参照のこと）。

□ (request → ▽ answer)　　(1)

もし設計者が応答時間を１に指定したい場合には、true/false信号を用いて応答時間が２の場合には誤りであることを示すタイムチャートを記述すればよい(Fig.2)．この場合には時相論理式は次のようになる。

□ (request → ○ answer)　　(2)

上述の式は次のようなルールに従って生成される。

(ステップ1)　式(1)は Fig.1 の矢印によって生成され、requestとanswerの間の応答時間は一般化ルールにより、必然的に決まる。この条件に合うルールが複数存在する場合には、最初に適合したルールがに従う。

(ステップ2)　シミュレーションによって式(1)が Fig.1 に示されたタイムチャートと矛盾しないかどうかのチェックを行なう（シミュレーションについては次節で説明す）。もしシミュレーション結果が正しくなければ、ステップ１に戻って他の適合するルールを選択する。

このステップにおいて、Fig.1 のタイムチャートを満たす時相論理式が得られる。さらにFig.2 が与えられた場合には、続けてステップ３を行なう。

(ステップ3)　シミュレーションによって、式(1)が Fig.2 の関係を満たしているかどうかチェックを行なう。もし true/false信号がある時刻で０であった場合には、その時刻において式(1)は０でなければならない。

(ステップ4)　Fig.2 においてある一定期間true/false信号が０であり、その間シミュレーション結果が正しくなければ、式(1)を修正する。式(1)の answer と request の応答時間は Fig.2 に示された関係を満たすように修正される。ここで、▽が○に置き換えられるのは容易に推論できる。なぜなら、true/false信号が０であるのは１つの状態しかないからである。このようにtrue/false信号を用いることにより、起こってはならない状況を指定することができる。

A Tool to Gererate and Debug Specifications in Propositional Temporal Logic

Masahiro FUJITA and Hisanori FUJISAWA

FUJITSU LABORATORIES LTD.

二つめのルールは、インターバル分割ルールである。Fig.3 (a) を考えてみよう。Fig.3 (a) はつぎのことを表している。「もしエラーが発生したら、システムは待ち状態にはいり recover信号および restart信号が入ってくるのを待つ。そうして、両方の信号が1となると、システムは作動状態となる。」。図中に現われる三つの矢印は上の複雑な関係を表している。これらの関係はterminals を導入することにより最初に一つにまとめられる。terminalsはいくつかの状態またはインターバル（インターバルについては[2]を参照のこと）を表現している。Fig.3 (b) では、二つのterminals が自動的に生成され、矢印で示される先に表されている。時相論理はこれまで述べてきたような方法で生成される。もちろん設計者はFig.3 (b) に示されているタイムチャートを直接入力することも可能である

また、設計者が自分の設計スタイルに合わせた特別なルールを定義することができる。ルールベース システムを採用しているために、この定義は簡単である。現在、時相論理式ジェネレイタの実装はSUNワークステーションの Quintus Prolog 上で行なっている。

時相論理式 シミュレータ

直接入力または生成によって得られた時相論理式はシミュレーションによる検証がのぞまれる。そこで、シミュレーションデータからボトムアップに時相論理式の値を直接決定するシミュレータを開発した。各時相演算子は現時刻における式と次の時刻における式とに分けられる[1]。各状態において、時相論理式の値は現時刻における式の値と次時刻における式の値とからボトムアップに決定される。このことは過去からに未来にわたって値をトレースすることによって各状態における時相論理式の値が決定されることを意味している。

シミュレータの実装にはC言語を用いている。シミュレーションの時間は、シミュレーションを行なう区間の長さと時相論理式の長さの積に比例するため、ほとんど無視することができる。

3. まとめ

時相論理による仕様記述支援ツール YUKO について述べた。YUKOを試験的に実装し、現在評価を行なっている。

参考文献

[1] M. Fujita, et al. , "Logic design assistance with temporal logic", IFIP 7th CHDL, August 1985.
[2] B. C. Moszkowski, "Reasoning about Digital Circuits", Stanford Univ. , STAN-CS-83-970, June, 1983.

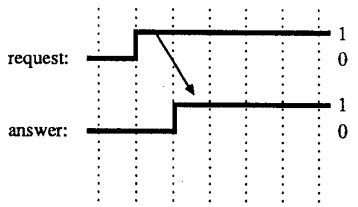


Fig.1 A timing diagram example

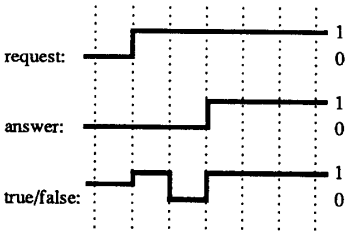
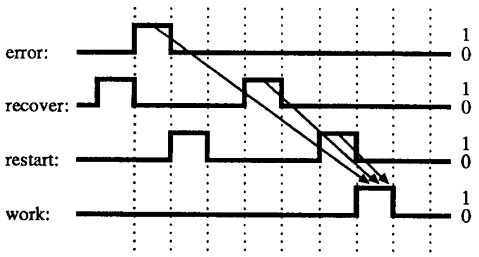
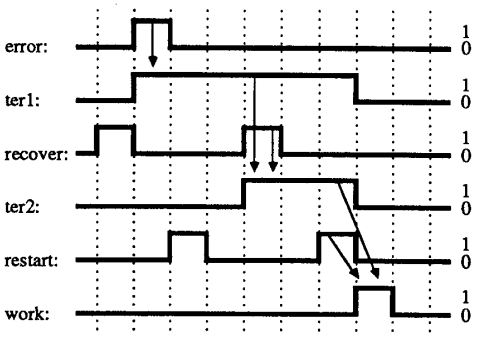


Fig.2 A timing diagram to show that the length is 1



(a) A timing for processor error recovery



(b) A simplified one from (a) using terminals

Fig.3 Decomposing conditions by introducing terminals