状態集合の分割による時相論理の検証の並列化

Parallelization of Temporal Logic Verification by Dividing State Space

河野 真治　　池村正之*
Shinji KONO　　Masayuki IKEMURA

琉球大学 工学部 情報工学科
Faculty of Information Engineering,
University of the Ryukyus

概 要

時相論理は、ハードウェアやソフトウェアの時間に依存して起こる事象の記述に適している。特に ITL (Interval Temporal Logic) は、オートマトンの動作 (状態遷移の様子) や時間に関する性質の検証において重要となっている。ここでは時相論理の検証として、Deterministic Tableau Expansion 法を用いた展開を行う。しかしこの方法は、変数に対して指数オーダーの計算量が必要である。特に、可能な状態を展開し記憶しておく領域が問題となる。そこで、Binary Subterm Tree を使って表現された状態データベースをネットワーククラスタ上に分割し、ITL の検証を並列に行うシステムを提案する。本システムの実装は、Prolog 上に Reliable な Datagram 通信ライブラリを構築し、その上で行った。

1　時区間時相論理 ITL

検証に使用する論理は、時相論理の一つである時区間時相論理 (Interval Temporal Logic：ITL) である。この論理は、離散時間と時区間の概念を持ち、無限に続く時間線に終了時刻が存在する。さらにその時区間を、長さ 0 を意味する 'empty' と、長さ 0 以上を意味する 'more' に分けて考える。

ITL では、普通の論理演算子 ('AND'、'OR'、'NOT' など) に加え、以下の時相演算子を用いる。またこれらを用いて、演算子 sometime-◇、always-□ などを表現することができる。

- $@P$: 次の時刻から始まる時区間で P が起こる。時区間が終り、次の時刻がないときは F となる。(next)
- $P\&Q$: 時区間を二つに分け、前半で P、後半で Q が起こる。(chop)

2　Deterministic Tableau Expansion

ITL では、時相論理式 P を二つの部分：現時刻とそれ以上の時刻に分けることができる。この分割を、*empty* と @-operator を含んだ disjunctive normal form を用いて

$$\vdash P \Leftrightarrow (empty \wedge P_e) \vee \bigvee_i P_i \wedge @X_i$$

と表すことができる。このような形を $@-normal\ form$ と呼ぶ。時相演算子を含む Px_i は状態式であり、これを時間的に再帰的に分割していく。この式は、条件 P_i で次の状態 $@X_i$ に遷移する非決定性オートマトンを生成すると考えて良い。さらに、展開時に遷移条件 P_i を相互に排他にすることにより、決定性オートマトンを生成することができる。この方法は、非決定性オートマトンから決定性オートマトンへの変換を含む Deterministic Tableau Expanstion である。

2.1 Binary Subterm Diagram

この展開により、Xi という状態式が生成される。ITL の変数を時刻にしか依存しないように制限することで、Xi の生成は有限となり、ITL は決定可能となる。状態を表す Xi は一般に複雑な時相論理式となるが、Binary Subterm Diagram を用いることにより、比較的コンパクトな表現とすることができる。

Binary Subterm Diagram は、もとの時相論理式の部分項を条件とする二分木である。ここでは、$?(Condition, True, False)$ という形で二分木を表す。ここで Condition は変数、または時相論理演算子を先頭に持つ部分項である。True、False は、Condition が真、偽の時に対応する二分木である。

例えば $\Diamond\Box p$ は、まず & を用いて $T\&\neg(T\&\neg p)$ に変換される。ここで $\neg p$ を $?(p,F,T)$ に変換し、$s_1 = T\&?(p,F,T)$ とすると、元の式を $s_2 = T\&?(s_1,F,T)$ と表現できる。この式を展開すると $\neg(T\&\neg p) \vee (T\&\neg(T\&\neg p))$ が生成される。この式は複雑に見えるが Binary Subterm Tree に変換すると $s_3 = ?(s_2, T, ?(s_1, F, T))$ と単純になる (図 1)。

$$\underbrace{\neg(\underbrace{T\&\neg p}_{s1}) \vee \underbrace{(T\&\neg(\underbrace{T\&\neg p}_{s1}))}_{s2}}_{s3}$$

図 1　Binary Subterm Tree

部分項は有限ではあるが、展開時に生成されることもある。部分項の順序を固定し、木を最大限に共有することで Binary Subterm Diagram を構成する。したがって、ITL の状態空間は、部分項と、Binary Subterm Diagram の二つの要素から構成される。

3　検証の並列処理法

検証を並列に実行するために、状態式の展開により生成された状態式の集合を分割し、その展開処理をデータ並列的に行う。分割は、Binary Subterm Diagram にハッシュ関数を定義することで行う。ただし、部分項は共有されているものとする。部分項を共有しないと、Binary Subterm Diagram の形の一意性を保証することはできない。

ここでは、ハッシュ関数として以下の関数を用いた。

```
hash(true)  = 1
hash(false) = 2
hash(?(C,True,False))
      = number(C)+hash(T)*3-hash(F)
```

Determistic Tableau により展開され生成された状態式は、Binary Subterm Diagram に変換され、ハッシュ関数を計算し、その値により各計算機に配られる (図 2)。一度送信したものは自分の状態データベースに追加し、通信に負荷をかけないようにする。状態を表す Binary Subterm Tree を受け取った計算機は、自分の持つ状態データベースと比較し、既に処理してあるものならば、そのままにし、新しいものは未処理のリストに追加する。変換時に生成する必要のある部分項は、部分項サーバによって大域的に番号づけられる。また、展開サーバが部分項サーバに、それぞれが持っている未展開の状態の数を報告し、未展開の状態がなくなった所で展開の終了を知ることができる。状態の通信量は、新たに生成される状態を台数で割ったものとなる。

3.1 Datagram 通信ライブラリ

本システムの実装は、SICStus Prolog を使い、スィチングネットワークで接続された PC クラスタ上で行った。本システムは、データベースの交換をハッシュ関数によって行うために、かなり自由な通信を必要とする。したがって、Stream 型のような回線型の接続方法では、台数分の 2 乗のソケットが必要になる。そこで、回線接続

図 2　システムの構成

module 指定	?-use_module(library(datagram)).
Stream Open	?-datagram(+IP_host:IP_port,-Stream).
Address 登録	?-datagram_address_registration (+Stream,+IP_host:IP_port,-Id).
送信先選択	?-datagram_destination(+Stream,+Id).
パケットのチェック	?-datagram_ready(+Stream,+Timeout,Ready).

表 1

の必要のない Datagram 通信ライブラリを作成した。これにより、計算機一台につき、状態データベース交換用と、部分項サーバへの通信用の 2 本のソケットしか消費しない。

Datagram へのアクセスは、Prolog Stream を経由しておこなうようになっており、メッセージ送信のような特別な構文を使う必要はない (表 1)。

Stream ではあるが、実際の通信は Datagram なので、ブロック単位での通信となる。ブロックの到着順序は保証されない。Stream の flush 毎に一つのブロックを構成する。

Datagram には信頼性がないが、本ライブラリが信頼性を提供する。TCP Stream のコネクションの代わりに、Datagram Stream 毎にアドレスデータベースを持つ。このデータベースの作成はユーザが行う。アドレスデータベースは、それぞれブロッキング・デブロッキングを行うキュー構造を持つ。したがってブロックの大きさは、ネットワークの MTU(Maximum Transfer Unit) に影響されない。SICStus Prolog/Quintus Prolog の fast read write library と共に使うことにより効率の良い通信を行うことができる。

ブロック単位で Prolog の項一つがそのまま転送される。

SICStus Prolog には、Linda 型のタプル通信ライブラリが付属しているが、サーバを必ず経由しないと通信することができない。サーバが集中的にアクセスされるので、Linda 型の通信は並列処理には向かない。Datagram 通信ライブラリは、完全結合通信を提供するので、スィチング・ネットワークなどと併用することにより、安価で効果的な並列処理を提供することができる。

状態空間の通信は、バッファを用いて、展開の合間に非同期に行われる。これに対し部分項サーバへは、Datagram ライブラリを使って作成された Remote Procedure Call を使用し、同期的にアクセスする。したがって、二本の別なソケットが必要となる。部分項サーバへのアクセスは、状態空間の大きさに比べて十分に小さいと考えら得るので、部分項サーバは全体で一台で良い。

4　比較

並列検証システムとしては、CMU で研究された BDD を共有メモリマシンで処理しモデル検証を行うというものが知られている。また ICOT では、LTTL の検証系を GHC によって並列実行するものが作成されている。これらはいずれも、共有メモリ上で処理するものであり、大規模な状態空間の処理では限界がある。また東大情報科学科では、AP3000 上の状態遷移木の追跡が作成されている。本研究では、状態空間を比較的小さな Binary Subterm Tree の集合として表現するところが異なる。また、Datagram ライブラリを使用することにより、特別なアーキテクチャを要求せずに安価な PC Unix クラスタ上で実現できるところが一つの利点である。

5　まとめ

本論文では、ITL の 状態を Binary Subterm Diagram にそって分割することにより、より大

きな状態空間を扱える並列検証システムの設計と実装について述べた。

このシステムの効率性は、状態空間がどのように分割されるかに依存する。あまりにばらばらでは、分散された状態空間間での通信がネックとなる。しかし、まったく分散せずに一つの計算機に留まるようでは台数効果はでず、状態空間も大きく使うことができない。そこで、仕様の性質に沿った分割が望ましいと考えられる。今後は、そのためのハッシュ関数の設計、および動的に生成される部分項と変数の順序などの調整により、より効率の良い並列検証システムを目指す。

参考文献

[1] Shinji Kono, “A Combination of Clausal and Non Clausal Temporal Logic Program”, July 12 1993.

[2] Masahiro Fujita and Shinji Kono, “Synthesis of controllers from Interval Temporal Logic specification”, February 2 1993.

[3] Shinji Kono, “Automatic Verification of Interval Temporal Logic”, Apr 24 1994.