設計自動化 68－21
(1993. 10. 29)

線形時相論理の為の効率的記号モデル検査方式

中根 清光
東京工業大学
情報工学科
nakane@cs.titech.ac.jp

米田 友洋
東京工業大学
情報工学科
yoneda@cs.titech.ac.jp

東京工業大学 工学部 情報工学科 〒152 東京都目黒区大岡山 2-12-1

あらまし 設計されたシステムが与えられた仕様どおりに動作するかどうかを確かめる設計検証はシステムの大規模化、複雑化により重要な役割を担うようになっている。一方、検証対象システムが大きくなると検証に要する計算量が急激に増加するという、いわゆる状態爆発という問題がある。この問題を解決する一つの方法として二分決定木 (Binary Desicion Diagram) を利用した論理関数処理に基づく記号モデル検査法が提案されている。本報告では、システムがペトリネットによって記述され、検証すべき性質が線形時相論理 (Liner-time Temporal Logic) で表されている場合に、記号モデル検査法を効率的に行う方法を提案する。

和文キーワード 記号モデル検査法、線形時相論理、ペトリネット、二分決定木

Efficient model checking for Liner-time Temporal Logic

Kiyomitsu Nakane
Department of Computer Science
Tokyo Institute of Technology
nakane@cs.titech.ac.jp

Tomohiro Yoneda
Department of Computer Science
Tokyo Institute of Technology
yoneda@cs.titech.ac.jp

Department of Computer Science, Tokyo Institute of Technology 2-12-1 Ookayama Meguro-ku Tokyo 152, Japan

Abstract The design verification to check whether designed systems satisfy given specifications becomes much important especially for complex, large-scale systems. On the other hand, as the systems to be verified become larger, the state explositon problem that the vefication costs increase exponentially. To avoid this problem, symbolic model checking based on manipulation of logic functions using Binary Decision Diaram has been proposed. This paper proposes some idea to accelerate symbolic model checking, where the systems are described by Petri-net, properties to be verified are formulated by Liner-time Temporal Logic.

英文 key words Symbolic model checking, Liner-time Temoral Logic, Petri-net, Binary Decision Diagram

1 はじめに

設計されたシステムが与えられた仕様どおりに動作するかを確かめる設計検証はシステムの大規模化、複雑化により重要な役割を担うようになっている。モデル検査法は検証を形式的、自動的に行う有力な方法の一つである。しかし、このような検証に際しての大きな問題は、検証システムの規模が大きくなると検証に要する計算量が爆発的に増加する、状態爆発という現象が生じる事である。この現象を低減する一つの方法として二分決定木 [1][2](Binary Desicion Diagram)(以下 BDD とする) を利用した論理関数処理に基づく記号モデル検査法が提案されている。[3], [4]

形式的検証をおこなう為、検証対象となるシステムのモデル化が必要である。このモデル化には、オートマトン、順序機械、状態遷移関数等もよく用いられるが、ペトリネットは容易に並列性を記述でき、システムをコンパクトに記述できるという特徴を持つ。一方、検証すべき性質も形式的に記述される必要があり、これには、CTL(Computation Tree Logic), LTL(Liner-time Temporal Logic) といった時相論理が使われる事が多い。

従来、CTL に基づく記号モデル検査法の効率化については活発に研究されており、システムをペトリネットで表す場合の効率的検証方法についても報告されている。[5][7]。LTL に基づく記号モデル検査法自体は [4] に提案されているが、その効率化に関する議論はあまりされていない。そこで、本報告では、LTL に基づき、かつシステムをペトリネットで表す場合の記号モデル検査法について議論する。

ペトリネットは状態遷移関数でその動作を表現し [5]、与えられた LTL 式のタブロを表す論理関数と論理積をとる。この論理関数の上で、LTL 式の Until オペレーターが満足される事を表す CTL 式の記号モデル検査を行う事により、LTL とペトリネットの記号モデル検査を実現する。この際、[5][7] で提案されている遷移関数の分割、到達可能状態を使った状態空間の削減、といった効率化手法とともに、ペトリネットの構造および検証される CTL 式の形に注目して状態遷移関数を効率的に表現し、拡張する事を考えた。

2 準備

2.1 BDD

BDD[1][2] は論理関数を表現するものであり、例を図 1に示す。これは、与えられた変数順序の下で、関数をシャノン展開して非巡回有向グラフで表現したものである。関数の意味は、根から葉へ経路を辿っていく事で与えられる。例えば、図 1の $x1 \cdot x2 + \overline{x1} \cdot \overline{x2}$は、$x2$ を 1 、$x1$ を 1 、と辿る時と、$x2$ を 0 、$x1$ を 0 、と辿る時に 1 となる。

また、複数の論理関数間でグラフの共有を許したものを共有二分決定グラフ (SBDD) と言う。

BDD、SBDD で表現される論理関数の node 数は、変数の順番によって大きく変化する為その事を考慮する必要がある。

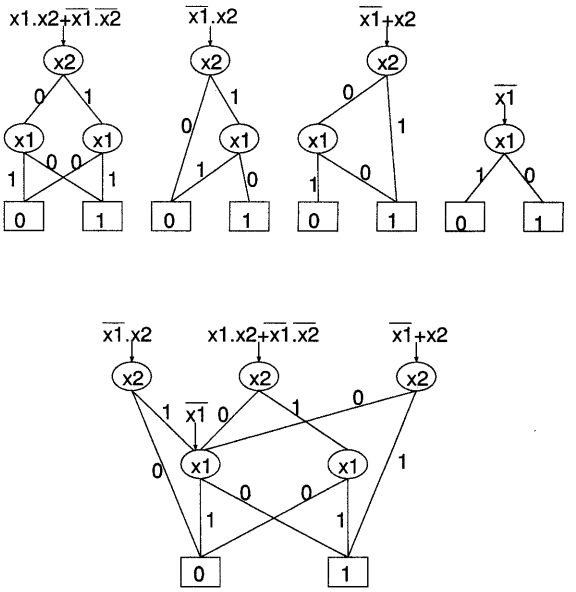

図 1: BDD(上) と SBDD(下) の例

2.2 ペトリネット

ペトリネット N とは 4 項組 $N = (P, T, F, m_0)$ である。

- $P = \{p_1, p_2, \ldots, p_n\}$ はプレース (place) の有限集合であり、$n \geq 0$ である。
- $T = \{t_1, t_2, \ldots, t_m\}$ はトランジション (transition) の有限集合であり、$m \geq 0$ である。
- $F \subseteq (P \times T), (T \times P)$ はフロー関係
- $m_0 \subseteq P$は初期マーキング

また、$\bullet t = \{p | (p,t) \in F\}, t\bullet = \{p | (t,p) \in F\}$ をそれぞれ、t の preset, postset と言う。簡単なネットの例を図 2に示す。

システムの状態はマーキングで定義され、それは Pの部分集合である。システムの動作は、トランジションの発火によって表現でき、その発火規則は以下のように定義される。

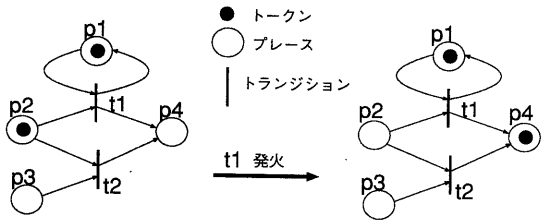

図 2: ネットの例

- あるトランジションtとマーキング m において、$\bullet t \in m$ の時、発火可能
- 発火可能なトランジションはいつ発火しても良いし、しなくても良い。

t がマーキング m で発火可能の時、$m' = \{m - \bullet t\} \cup t\bullet$ を t の発火結果であるとする。m と m' がこのような関係にある場合、$m \xrightarrow{t} m'$ と書く。遷移系列 $\sigma = s_0 s_1 s_2 \cdots s_n \cdots$ は s_0 を初期状態として $s_i \xrightarrow{t} s_{i+1}$ であるような無限または有限の系列である。

2.3 CTL(Computation Tree Logic)

CTL 式は再帰的に以下のように定義される。以下で AP とは Atomic Proposition(原始命題) の集合であるとする。

- $p \in AP$ なら、p は CTL 式
- f が CTL 式なら、$\neg f$, EXf, EGf も CTL 式
- f, g が CTL 式なら、$f \vee g$, $E[f\ U\ g]$ も CTL 式

CTL 式の真偽値は、Kripke-structure $K = (S, R, I)$ 上で定義される。

- S は有限状態集合
- $R \subset S \times S$ は S の要素間のつながり
- $I : S \rightarrow 2^{AP}$ は各状態において真となる AP の要素をラベル付けする関数

無限の状態系列 $\pi = s_0, s_1, \cdots$ を $path$ と言い、$\forall i \geq 0$, $(s_i, s_{i+1}) \in R$ となる。また $\pi(i) = s_i$ とする。

CTL 式は K 上の状態 s において、以下のように定義される。

- $(K, s) \models p(\in AP)$ iff $p \in I(s)$
- $(K, s) \models \neg f$ iff $s \not\models f$
- $(K, s) \models f \vee g$ iff $(K, s) \models f$ *or* $(K, s) \models g$
- $(K, s) \models EX\ f$ iff $(K, s') \models f$ $((s, s') \in R)$
- $(K, s) \models EG\ f$ iff K 上に $\pi(i) = f\ \forall i \geq 0$ となる $path$ π が存在する
- $(K, s) \models E[f\ U\ g]$ iff K 上に $\exists i \geq 0$, $(K, \pi(i)) \models g$, $(K, \pi(j)) \models f$, $0 \leq \forall j \leq i$ となる $path$ π が存在する

2.4 Fairness Constraints について

制約の集合 C を考え、ある $path$ において、C の各要素が無限にしばしば (infinitely often) 成り立つ時、その $path$ は C に関して $fair$ であると言う。このような $fair$ な $path$ 上でのみ成立する CTL 式 $EG\ f$ を $E_C G\ f$ と書く。すなわち、$(K, s) \models E_C G\ f$ iff $\pi(i) = f\ \forall i \geq 0$ となる fair な $path$ π が K 上に存在する、と言う解釈になる。例として、以下に $E_C G\ true$ がどのように求められるかを示す。

$$E_C G\ true = \bigwedge_{k=1}^{n} EX\ EF(E_c G\ true \wedge c_k) \cdots (1)$$
$$C = \{c_1, c_2, \cdots c_n\}$$

2.5 LTL(Liner-time Temporal Logic)

- $p \in AP$ なら、p は LTL 式
- f が LTL 式なら、$\neg f$ も LTL 式
- f, g が LTL 式なら、$f \vee g$, $f\ U\ g$ も LTL 式

LTL 式 ϕ は $path$ π に対して以下のように定義される。

- $(\pi, i) \models p(\in AP)$ iff $p \in I(s_i)$
- $(\pi, i) \models \neg f$ iff $(\pi, i) \not\models f$
- $(\pi, i) \models f \vee g$ iff $(\pi, i) \models f$ *or* $(\pi, i) \models g$
- $(\pi, i) \models f\ U\ g$ iff $\exists j \geq i$, $(\pi, j) \models g$, $(\pi, k) \models f$, $i \leq \forall k \leq j$

ペトリネット $N = (P, T, F, m_0)$ に対して LTL 式 ϕ の意味を以下のように与える。

$AP = P$、$\forall i \geq 0\ (m_i \xrightarrow{t} m_{i+1})$ なるマーキングの系列 π を $\pi = m_0 m_1 m_2 \cdots$ を path とし、このような $path$ の集合を B とする時、

$$N \models \phi \text{ iff } \forall \pi \in B,\ \pi \models \phi$$

と定義する。これは、ペトリネット上のあらゆる $path$ で LTL 式 ϕ が成立する、と言う事を意味する。

3 LTL 記号モデル検査アルゴリズム

この方法の原理は、ペトリネットの状態遷移関数と LTL 式の動作を表す状態遷移関数を作り、両方の動作を満たすようなパスを辿っていく事で式の真偽を判定しようというものである。

3.1 準備

$B = \{0, 1\}$, $E = B^n$とする。この時、集合$A \subseteq E$に対し、次の条件を満たす関数$\chi_A : E \to \{0, 1\}$をAの特性関数と呼ぶ。

$$\chi_A(x) = 1 \text{ iff } x \in A$$

fを論理関数、$\boldsymbol{x} = (x_1, x_2, \cdots x_n)$とする時、スムージング演算子$S$は以下のように定義される。

$$S_x f = S_{x_1} \cdots S_{x_n} f$$

$$S_{x_i} f = f_{x_i} + f_{\overline{x_i}}$$

但し、f_a, $f_{\overline{a}}$はfのaに1と0をそれぞれ代入して得られる論理関数である。

3.2 ペトリネットの状態遷移関数を作る方法

まず、マーキングを二値ベクトルに変換する関数、すなわち各プレースに関数$number : P \to N$ (Nは自然数) によって順番を与え、$\nu : 2^P \to B^{|P|}$を以下のように定義する。

- m:マーキング
- $\boldsymbol{v} = (v_1, v_2, \cdots v_{|P|}) \in B^{|P|}$
- $\nu(m) = \boldsymbol{v}$
- $\boldsymbol{v}_{number(p)} = 1$ iff $p \in m$ ($\boldsymbol{v}_i$は$\boldsymbol{v}$のi番目の要素)

状態遷移関数の論理関数表現はTrは、二つのマーキングm, m'のベクトル表現を引数とし、$m \xrightarrow{t} m'$の関係が成り立つなら1となる関数である。図2では、

$$Tr(\ \nu(m),\ \nu(m')\ ) = Tr(\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1\ ) = 1$$

となる。

これをペトリネットの構造のみから (状態解析をする事なく) 以下のように求める。

ここで、$P = \{p_1, p_2, \ldots, p_{|P|}\}$をプレースに対応する状態論理変数集合、$P' = \{p'_1, p'_2, \ldots, p'_{|P|}\}$を$P$の次状態論理変数集合 ($p'_1$は$p_1$の次状態変数、$p'_2$は$p_2$の次状態変数を表す) とし、$\boldsymbol{p} = (p_1, p_2, \cdots p_{|P|})$, $\boldsymbol{p'} = (p'_1, p'_2, \cdots p'_{|P|})$とする。またプレースの集合である$\bullet t$, $t\bullet$を状態論理変数の集合と拡大解釈する。関数$F_t(\boldsymbol{p}, \boldsymbol{p'})$を以下のように定義する。

$$F_t(\boldsymbol{p}, \boldsymbol{p'}) = \bigwedge_{p \in \bullet t} p \wedge \bigwedge_{p \in t\bullet} p' \wedge \bigwedge_{p \in \bullet t - t\bullet} \neg p' \wedge \bigwedge_{p \notin (\bullet t \cup t\bullet)} (p = p')$$

第一項から第三項まではトランジションtが発火したときのマーキングの変化を表し、第四項はtにより変化しないトークンを表す。従って、$F_t(\nu(m), \nu(m'))$は、mでtが発火可能であり、かつ$m \xrightarrow{t} m'$の時、1となる。また発火可能なトランジションが存在する場合にマーキングの変化が起こらないことを表す為の関数$G(\boldsymbol{p}, \boldsymbol{p'})$を以下のように定義する。

$$G(\boldsymbol{p}, \boldsymbol{p'}) = \neg \bigvee_{t \in T} \Big(\bigwedge_{p \in \bullet t} p \Big) \to \bigwedge_{p \in P,\ p' \in P'} (p = p')$$

これらを用い、

$$Tr_N(\boldsymbol{p}, \boldsymbol{p'}) = \bigvee_{t \in T} F_t(\boldsymbol{p}, \boldsymbol{p'}) \vee G(\boldsymbol{p}, \boldsymbol{p'})$$

図3の例では、

- $Tr(\nu(m), \nu(m')) = F_{t1}(\nu(m), \nu(m')) \vee F_{t2}(\nu(m), \nu(m')) \vee G(\nu(m), \nu(m'))$
- $F_{t1}(\nu(m), \nu(m')) = p1 \wedge \neg p1' \wedge p3' \wedge (p2 = p2') \wedge (p4 = p4')$
- $F_{t2}(\nu(m), \nu(m')) = p1 \wedge p2 \wedge \neg p1' \wedge p2' \wedge p4' \wedge (p3 = p3')$
- $G(\nu(m), \nu(m')) = \neg(p1 \wedge \neg p3 \ \vee\ p1 \wedge p2 \wedge \neg p4) \to ((p1 = p1') \wedge \cdots (p4 = p4'))$

となる。

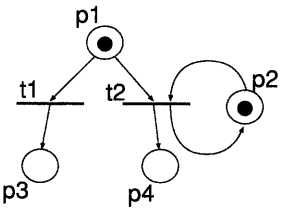

図 3: ネットの例

3.3 LTL 式の状態遷移関数を作る方法

以下に示すように、与えられたLTL式を満たすような動作のみを実現する状態グラフを生成する。ϕを与えられたLTL式とする。

1. LTL式のelementary subformulaを生成する。elementary subformulaとは以下の再帰的手続きによって構成される集合である。
   - $el(p) = \{\ p\ \} : p \in AP$
   - $el(\neg f) = \{\ f\ \}$
   - $el(f \vee g) = el(f) \cup el(g)$
   - $el(X\ f) = \{X\ f\} \cup el(f)$
   - $el(f\ U\ g) = \{X(f\ U\ g)\} \cup el(f) \cup el(g)$
2. 以下のようなStructure $K_L = (S_L,\ R_L)$を考える。
   - $S_L = 2^{el(\phi)}$
   - $R_L \subseteq S_L \times S_L$は$S_L$間の遷移関係

   $\alpha(f)$は$s \in S_L$を引数とし、sでfがラベル付けされているなら1を返す関数であり、以下のように再帰的に定義される。

- $\alpha(f)(s) = 1$ iff $f \in s$ or $f = true$
- $\alpha(\neg f)(s) = \neg\alpha(f)(s)$
- $\alpha(f \vee g)(s) = \alpha(f)(s) \vee \alpha(g)(s)$
- $\alpha(fUg)(s) = \alpha(g)(s) \vee (\alpha(f)(s) \wedge \alpha(X(fUg))(s))$

Structure 上の遷移関係 R_L は以下のように定義される。

$$(s, s') \in R_L \equiv \bigwedge_{Xg \in el(\phi)} \alpha(X\,g)(s) \leftrightarrow \alpha(g)(s')$$

すなわち、状態 s において、式 $X\,g$ がラベル付けされているなら、状態 s' では式 g がラベル付けされる。および、その逆が成り立つとき、s' を s の次状態としている。

ここまでの操作を簡単な例を用いて説明する。

LTL 式を $\phi = true\ U\ a$ とする。これは、$path$ 上でいつかは a が真となることを表現する。

(a) $el(\phi) = \{a, X(true\ U\ a)\}$ となり、S_L は以下の図 4のようになる

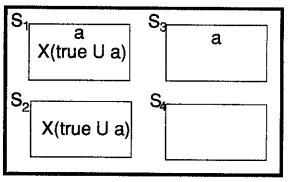

図 4: S_L

(b) 各式に対するαは以下のようになる。

- $\alpha(a)(s_1) = 1$
- $\alpha(X(true\ U\ a))(s_1) = 1$
- $\alpha(true\ U\ a)(s_1) = 1$
- $\alpha(a)(s_2) = 0$
- $\alpha(X(true\ U\ a))(s_2) = 1$
- $\alpha(true\ U\ a)(s_2) = 1$
- $\alpha(a)(s_3) = 1$
- $\alpha(X(true\ U\ a))(s_3) = 0$
- $\alpha(true\ U\ a)(s_3) = 1$
- $\alpha(a)(s_4) = 0$
- $\alpha(X(true\ U\ a))(s_4) = 0$
- $\alpha(true\ U\ a)(s_4) = 0$

従って、$\alpha(X(true\ U\ a))$ が 1 である状態 s_1 の次状態は$\alpha(true\ U\ a)$ が 1 となる s_1, s_2, s_3 となる。これを繰り返すと遷移関係は、図 5のように求められる

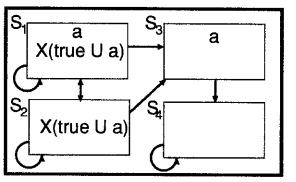

図 5: 遷移状態

記号モデル検査法に用いる為に、この遷移関係の論理関数表現 Tr_ϕ を 3.2 と同様に以下のように求める。

各 elementary-subformula の要素に関数 $number'$: $el(\phi) \to N$ (Nは自然数) によって順番を与え、$\mu : 2^{el(\phi)} \to B^{el(\phi)}$ を以下のように定義する。

- $s \in S_L$
- $\boldsymbol{e} = (e_1, e_2, \cdots e_{|el(\phi)|}) \in B^{|el(\phi)|}$
- $\mu(s) = \boldsymbol{e}$
- $\boldsymbol{e}_{number(e)} = 1$ iff $e \in s$ ($\boldsymbol{e}_i$は$\boldsymbol{e}$の i 番目の要素)

$AP_L = \{e_1, e_2, \cdots e_{|el(\phi)|}\}$, $AP'_L = \{e'_1, e'_2, \cdots e'_{|el(\phi)|}\}$ を elementary-subformula の要素に対応する論理変数集合とし、$\eta : B^{|el(\phi)|} \to 2^{|el(\phi)|}$ を以下のように定義する。但し、$\boldsymbol{e} = (e_1, e_2, \cdots e_{|el(\phi)|})$ とする。

$$\eta(\boldsymbol{e}) = \{e \mid \boldsymbol{e}_{number'(e)} = 1\}$$

これらより、

$$Tr_\phi(\boldsymbol{e}, \boldsymbol{e}') = \bigwedge_{X\,g \in el(\phi)} \alpha(X\,g)(\eta(\boldsymbol{e})) \leftrightarrow \alpha(g)(\eta(\boldsymbol{e}'))$$

と定義する事ができる。これより、$(s, s') \in R_L$ という関係は $Tr_\phi(\mu(s), \mu(s')) = 1$ と表すことができる。

3.4 記号モデル検査法のアルゴリズム

次に前述の LTL 式の状態遷移関数とシステム (ペトリネット) の状態遷移関数を用いて、LTL 式の記号モデル検査を行う方法を示す。全体の概略を図 6に示す。以下で、$\boldsymbol{s}$ は $\boldsymbol{s} = (s_1, s_2, \cdots s_{|el(\phi) \cup P|}) \in B^{|el(\phi) \cup P|}$ である。

1. LTL 式ϕから、状態遷移関数と S_0を生成する。但し、$S_0(\boldsymbol{e})$ は $\{\mu(s) \mid \alpha(\phi)(s) = 1\}$ の特性関数とする。

2. 与えられたペトリネット $N = (P, T, F, m_0)$ から状態遷移関数論理関数表現 Tr_N と M_0を生成する。但し、$M_0(\boldsymbol{p})$ は $\{\nu(m_0)\}$ の特性関数とする。

3. $Tr(\boldsymbol{s}, \boldsymbol{s}') = Tr_N(\boldsymbol{p}, \boldsymbol{p}') \wedge Tr_\phi(\boldsymbol{e}, \boldsymbol{e}')$ を求める。この関数は直感的にペトリネットの状態遷移関数から LTL 式を満たさなくなる動作を取り除いた物と考える事ができる。

4. $S_0(e) \wedge M_0(p)$ を求め、これを $S_{init}(s, s')$ とする。

5. Kを $2^{el(\phi)\cup P}$上の Trにより表される Kripke-Structure、$S \subseteq 2^{el(\phi)\cup P}$を S_{init}で表される Kの状態の集合とする。Sの各要素 s に対し、$(K, s) \models E_c G\ true$ を調べ、$\exists s \in S$, $(K, s) \models E_c G\ true$ なら $N \not\models \neg\phi$ となる。

   但し、制約集合は $s \in 2^{el(\phi)\cup P}$に対し、

   $C = \{\neg\alpha(g\ U\ h)(s) \vee \alpha(h)(s) = 1 \mid g\ U\ h$ がϕに含まれる $\}$

   である。

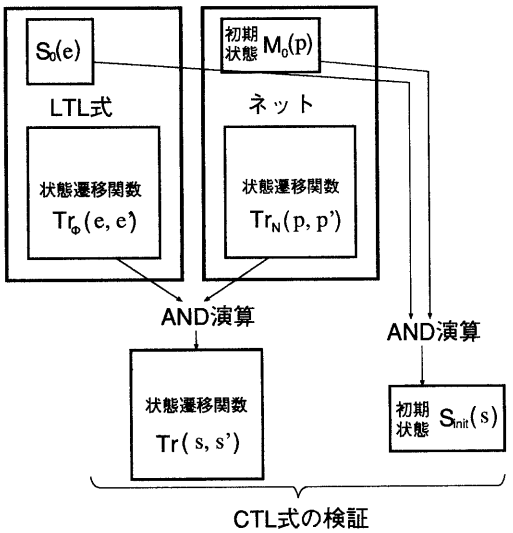

図 6: 検証の流れ

4 高速に検証を行う為の方法

ここでは、前述の LTL 式の記号モデル検査アルゴリズムを高速化するいくつかの手法について述べる。

4.1,4.2 の方法は、[5],[7] で提案されている方法で 4.3,4.4 が今回提案するものである。なお 4.1～4.3 の方法は CTL の記号モデル検査法にも摘要できるものである。

4.1 状態空間の限定

CTL 式の検証を行う為には、逆像計算が重要となる。CTL 式ϕが真となる状態の集合の特性関数をχ_ϕとする。例えば、$EX\ f$を満たす状態集合の特性関数$\chi_{EX\ f}$は、

$$\chi_{EX\ f}(s) = S_{s'}(\chi_f(s') \wedge Tr(s, s'))$$

となる (Sはスムージング演算子)。この$\chi_{EX\ f}(s)$を$\chi_f(s)$の逆像と呼ぶ。$EF\ f$に対して$\chi_{EF\ f}$は、

$$F_0(s) = \chi_f(s)$$

$$F_{i+1}(s) = F_i(s) \vee \chi_{EX\ F_i(s)}$$

とした時、$F_{i+1} = F_i$となる i に対し、$\chi_{EF\ f} = F_i$である。

この時、実際には初期状態から到達しない状態に遡る事もありうる。これを防ぐ為、初期状態が与えられた時、この初期状態から到達できる空間 (到達可能空間) を最初に求めておき、図 7に示すように逆像計算を行った時に考慮する必要のない状態を削り、網の部分のみを用いる事にする。

実際、考慮すべき空間は全状態空間 (place 数が n 個なら 2^n個の状態がある) に比べて非常に小さいと思われるので、効果が望める。

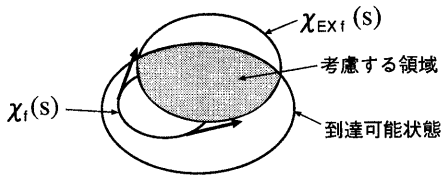

図 7: 領域の限定

4.2 状態遷移関数の分割

ペトリネットの状態遷移関数の論理関数表現 Tr_Nは 2.3 で述べた方法で求めるが、検証対象が大きくなるとそれを表現する BDD の node 数が非常に大きくなる。そこで、その状態遷移関数を分割して保持する。具体的には、トランジションをいくつかのグループに分け、各グループごとに状態遷移関数を求める。グループ i ($1 \le i \le n$:n はグループ数) の状態遷移関数の論理関数表現を $f_i(\ s,\ s'\ )$ とすると、以下の関係が成り立つ。

$$Tr_N(\ s,\ s'\ ) = \vee_{1\le i\le n} f_i(\ s,\ s'\ )$$

実際のトランジションのグループ分けの方法は、親子関係にあるトランジションを同一グループにまとめ、並列なトランジションは異なるグループに属するようにする。これは、各グループ内でトランジション発火系列が長くなった方が検証時間を短縮できるという経験則に基づくものである。図 8は二通りの分割を示しているが、上記の方針によると、$\{(Block1), (Block2, Block4), (Block3, Block5)\}$のように分割する。

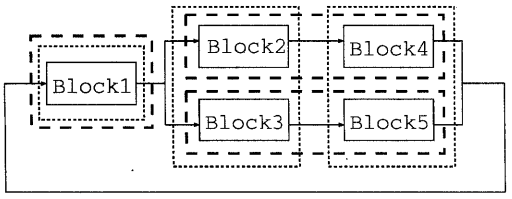

図 8: トランジションのブロック分割の仕方

4.3 状態遷移関数の効率的実現について

同じ論理関数を表現する場合でも、BDD のサイズは変数の順番によって大きく異なる。ペトリネットの状態遷移関数の論理関数表現 Tr_N を作る場合も同様で、あるトランジション t の preset,postset 中のプレースがあまり離れた順番で宣言されると BDD のサイズが大きくなる事が経験上分かっている。そこでこうした状況をさける為に以下の方法を用いる。

プレースの順番付けは 4.2 で分割した論理関数 f_i に着目してペトリネットの一部に対して行うのではなく、ペトリネット全体に対して行う。

```
・count := 1
・P := φ
while(選ばれていないプレースが存在する){
    ・if (P == φ)
        選ばれていないプレース p を取りだし、P := {p}
    while(P ≠ φ){
        ・集合 Pからプレース p を一つ取り出し、
                        count をラベル付けする
        ・count := count + 1
        ・T := {t | p ∈ •t or t•}
        while(T ≠ φ ){
            ・集合 Tからトランジション t を一つ取り出す
            ・P' := P' ∪ {p | p ∈ •t or p ∈ t•}
        }
    }
    ・P := P'
}
```

これは、あるプレースを取り出したら番号を付け、それにトランジションを介してつながるプレースを広さ優先で捜して番号をつけていく方法である。

4.4 状態遷移関数の拡張

今まで、説明してきた状態遷移関数はある状態 s と、s で発火可能な一つのトランジションの発火後の状態 s' との情報しか持っていない。s' を、s から一ステップで到達できる状態、この状態遷移関数を一ステップの情報を持つ関数、と呼ぶことにする。この状態遷移関数が、二つ以上のステップの情報を持つようにする事を考える。以下の計算により、ある整数 m に対し、2^{m-1} ステップ以下の情報を持つ関数 $Tr_m(v, v')$ を求める事が出来る。

1. $Tr_1(v, v') = Tr(v, v')$
2. $Tr_{i+1}(v, v') = \bigvee_{for\ all\ v''} Tr_i(v, v'') \cdot Tr_i(v'', v'),\ 1 \leq i$
3. 2 を繰り返す。

関数 $Tr_i(v, v')$ は状態 v から状態 v' へ長さ 2^{i-1} 以下の $path$ 存在する時、1 となる関数である。

LTL 式を検証するには、それと等価な CTL 式を作って検証するが、その形は決まっている。(式 (1)) 上記の Tr_i を使うと $EF\ f$ の形の検証が効率的になると思われる。これは Tr では一回の計算は一ステップしか遡る事しかできないのに対し、Tr_m は一回の計算で 2^{m-1} 以内のステップ遡ることができ、繰り返し回数が少なくなるためである。

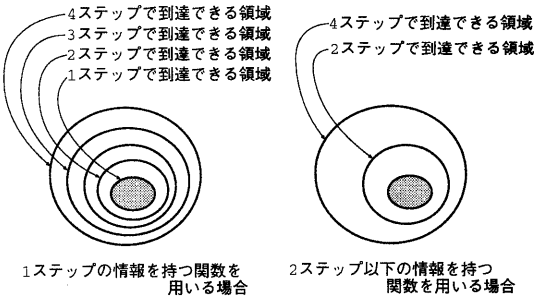

図 9: 関数の拡張

図 9は、網かけの部分から 4 つ分遡るとする時、1 ステップの情報のみを持つ関数は 4 回計算を要するが、2 ステップ以下の情報を持つ関数は 2 回の計算で辿ることが出来る、と言う事を表している。

但し、大きな m に対して Tr_m を求めようとすると、BDD によって表現される関数の規模が大きくなり、折角分割した意味がなくなってしまう。上記の計算はグループ間にまたがる遷移も考えるからである。

そこで、グループ間の遷移は考えず同一グループ内の遷移を考慮して、上記の計算を行う。すなわち

$$Tr(\boldsymbol{s}, \boldsymbol{s}') = Tr_N(\boldsymbol{p}, \boldsymbol{p}') \wedge Tr_\phi(\boldsymbol{e}, \boldsymbol{e}')$$

ペトリネット	LTL 式	time(sec)		
		(a)	(b)	(c)
45p44t	f1	2.72	2.83	2.596
	f2	3.27	3.30	3.00
	f3	2.58	2.50	2.42
	f4	3.12	2.89	2.77
104p115t	f1	87.58	46.91	33.43
	f2	509.24	102.5	60.98
	f3	117.43	61.74	30.21
	f4	277.54	110.09	62.83
253p188t	f1	-	33748	22962
	f2	-	21289	13777
	f3	-	719	665
	f4	-	1857	1344

表 1: 結果

$$= \bigvee_{1 \le i \le n} f_i(\boldsymbol{p}, \boldsymbol{p}') \wedge Tr_\phi(\boldsymbol{e}, \boldsymbol{e}')$$

であるから、各 $f_i(\boldsymbol{p}, \boldsymbol{p}') \wedge Tr_\phi(\boldsymbol{e}, \boldsymbol{e}')$ に対して、上記の計算を行う事にする。

5 実験結果

検証に使ったLTL式は以下の通りである。

$f1 = \Box(\ m\_req\_to\_alu\_c \rightarrow (\ m\_req\_to\_alu\_c\ U\ reg\_to\_alu\_c\ )\ )$

$f2 = \Box(\ reg\_to\_alu\_c \rightarrow (\ reg\_to\_alu\_c\ U\ m\_req\_to\_alu\_nc\ )\ )$

$f3 = \Box(\ m\_req\_to\_alu\_nc \rightarrow (\ m\_req\_to\_alu\_nc\ U\ reg\_to\_alu\_nc\ )\ )$

$f4 = \Box(\ reg\_to\_alu\_nc \rightarrow (\ reg\_to\_alu\_nc\ U\ m\_req\_to\_alu\_c\ )\ )$

使用した三種類のペトリネットは、それぞれ

- c1.45p44t : プレース数 45, トランジション数 44
- c2.104p115t : プレース数 104, トランジション数 115
- c3.253p188t : プレース数 253, トランジション数 188

である。

表 1 で、(a) は 4.1,4.2 のみ用いた場合、(b) は 4.1,4.2,4.3 のみ用いた場合、(c) は 4.1,4.2,4.3,4.4 を用いた場合を表す。－となっているのは検証を完了する事ができなかった事を示す。このように、大きなペトリネットに対して 4.3,4.4 の効果が分かる。

6 まとめと今後の課題

本稿では、ペトリネットと LTL を用いた記号モデル検査法について、BDD の変数の並べ方と遷移関数の拡張により検証の効率化を達成した。

今後の課題としては、現在トランジションのグループ分けはペトリネットの構造を見て人手で行っているので、変数の順番付けと同様に何らかの自動化を考えていきたい。

参考文献

[1] RANDEL E.BRYANT: Graph-Based Algorithm for Boolean Function Manipulation, Transaction on computers VOL C-35, NO 8, AUGUST 1986.

[2] Shin-ichi MINATO, Nagisa ISHIURA, Shuzo YAJIMA :Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipuration: 27th ACM/IEEE Design Automation Conference 1990.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan :Sequential Circuit Verification Using Symbolic Model Checking: 27th ACM/IEEE Design Automation Conference 1990.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan :Symbolic Model Checking:10^{20} States and Beyond: Academic Press,1992.

[5] Kiyoharu Hamaguchi, Hiromi Hiraishi, Shuzo Yajima :Design Verification of Asynchronous Sequential Circuit Using Symbolic Model Checking: hama@kuis.kyoto-u.ac.jp.

[6] Tomohiro Yoneda: On Model Checking for Petri-Nets and a Liner-Time Temporal Logic: IEICE FTS92.

[7] Kiyoharu Hamaguchi,Shuzo Yajima: Symbolic Model Checking for Petri Nets Using Block Partitions: FTC 研究会, 1993.