マルチメディア通信と分散処理 71−20
（１９９５．７．１４）

拡張リアルタイム時相論理による分散ソフトウェアの形式的検証

山根 智　　翁 崇恩
島根大学理学部情報科学科
松江市西川津１０６０

分散ソフトウェアは，多くのプロセスが並行動作してタイミング制約が厳しく，動作が正しいタイミングで行われるかどうかが非常に重要である．本稿では，リアルタイム時相論理の表現力を向上させた拡張TCTL(Timed CTL)を提案して，効率的なモデルチェッキング手法を提案する．本手法の主要な特徴は以下のとおりである．
(1)稠密時間モデルかつ分岐時相論理においてfreeze quantificationとbounded temporal operatorの融合により，簡潔で表現力の高いタイミング制約表現が可能となる．
(2)モデルチェッキングは不等式手法とラベリング手法の融合により実現する．

Formal verification of distributed software based on extended real-time logic

Satoshi Yamane　Suon Oh
Dept. of Computer science　Faculty of Science　Shimane University
1060 Nsihikawatu　Matue city

Distributed software consists of many concurrent processes and behaves on strict timing conditions. In this paper, we propose extended TCTL(Timed TCTL) and effective real-time model checking as follows.
(1)The timing description of extended TCTL consists of both freeze quantification and bounded temporal operator. For this extension, extended TCTL admits timing constraints between distant contexts.
(2)Real time model checking consists of both labelling algorithm and inequalities method.

１．まえがき

通信システムなどの分散ソフトウェアは，多くのプロセスが並行動作してタイミング制約が厳しく，システム全体の状態数は膨大である．このために，分散ソフトウェアは，タイミング検証は重要である．

分散ソフトウェアのタイミング検証方式は多くの研究がなされており，モデルチェッキングや言語包合問題などがある[1]．本稿では，興味深い検証性質が簡潔なリアルタイム時相論理で表現できることに着目して，モデルチェッキングによるタイミング検証を考える．リアルタイム時相論理によるモデルチェッキングでは，システム仕様やリアルタイム時相論理の時間モデルやリアルタイム時相論理のタイミング表現，タイミング検証方式が重要となる．

システム仕様やリアルタイム時相論理の時間モデルとしては，離散時間モデルや仮想クロックモデル，稠密時間モデルが提案されている[1]．しかし，離散時間モデルと仮想クロックモデルは同期型ソフトウェアしか扱えず，非同期型ソフトウェアの検証では稠密時間モデルを採用する必要がある．本稿では，システム仕様やリアルタイム時相論理が実数領域のタイミング制約表現である稠密時間モデルを採用する．システム仕様は各々のプロセスを時間オートマトン[2]により記述して，システム動作は時間オートマトンの並行動作として表現する．リアルタイム時相論理については，詳細な議論が必要なので以下に述べる．

リアルタイム時相論理のタイミング表現としては，bounded temporal operatorやFreeze quantification（Half-Order temporal logic)，explicit clock variable（First-Order temporal logic)が提案されている．bounded temporal operator[3]は

$$p\,U_{<10}\,q$$

といった記述形式が代表的であり，１０時刻以内でｑが成り立つまでｐが成り立つと解釈する．Freeze quantification[4]は

$$\Diamond x.(p \wedge x < 10)$$

といった記述形式が代表的であり，１０時刻以内でｐはある状態で成り立つと解釈する．explicit clock variable[5]は

$$\forall x \exists y \Box((p \wedge now = x) \rightarrow p\,U\,(q \wedge now = y \wedge y \leqq x + 10))$$

といった記述形式が代表的であり，すべてのｐが成り立つ状態は１０時刻以内でｑが成り立つ状態になると解釈する．ここで，nowはグローバルな動的クロック変数である．bounded temporal operatorは記述は簡潔であるが，隣接した状態間のタイミング制約しか記述できないために，

$$\Box x.(p \rightarrow \Diamond(q \wedge \Diamond y.(r \wedge y \leqq x + 5)))$$

が記述できない．また，explicit clock variableは表現力は高いが，動的クロック変数などの導入により記述が簡潔でない．本稿では，記述の表現力と簡潔さのバラ

ンスからfreeze quantificationとbounded temporal operatorの融合を考える．

従来のリアルタイム時相論理としては，稠密時間モデルのbounded temporal operatorのTCTL(Timed CTL)[6]や離散時間モデルのfreeze quantificationのTPTL(Timed PTL)[4]などがある．本稿では，稠密時間モデル上のfreeze quantificationとbounded temporal operatorの融合によるタイミング制約表現が可能な拡張TCTLを提案する．

また，モデルチェッキングはリアルタイム時相論理の時間Kripke構造へのラベリングであり，タイミング制約を考慮する必要がある．実数領域のタイミング検証方式としては，(1)整数時刻で近似する手法[7]や(2)リージョングラフ手法[6]，(3)不等式手法[8]がある．(1)と(2)は状態空間の数がタイミング制約の長さの指数オーダで増加するので，状態空間の組み合わせ爆発が発生する．本稿では，不等式手法とラベリングアルゴリズム[12]の融合によるタイミング検証方式を提案して，状態空間の組み合わせ爆発を抑制する．

以上のアプローチにより，新しい分散ソフトウェアのタイミング検証方式を提案する．主要な特徴を以下に述べる．

(1)稠密時間モデル上のFreeze quantificationとbounded temporal operatorの融合によるタイミング制約表現が可能な拡張TCTLにより検証性質仕様を記述する．

(2)システム仕様は時間オートマトンの並行合成により記述して，時間Kripke構造を自動生成する．

(3)モデルチェッキングは不等式手法とラベリングアルゴリズムの融合により実現する．

以下の本稿は，2章では拡張TCTLを提案して，3章ではタイミング検証方式を提案する．4章では事例を通して本手法の有効性を示して，5章ではまとめを述べる．

2．拡張TCTLの提案

稠密時間モデルのシステム仕様に対して，検証性質仕様が表現可能な稠密時間モデル上のfreeze quantificationとbounded temporal operatorの融合によるタイミング制約表現が可能な拡張TCTLを提案する．

2．1　拡張TCTLの構文

拡張TCTLはタイミング制約記述に関して，リアルタイム分岐時相論理TCTL(Timed CTL)[6]をfreeze quantificationとbounded temporal operatorの融合により拡張したものである．

［定義1］（拡張TCTLの構文）

拡張TCTLのタイミング記述πと時相論理式ϕの構文を以下のように帰納的に定義する．

$$\pi ::= x + c \mid c$$

$$\phi ::= ap \mid \pi_1 \leqq \pi_2 \mid \neg\phi \mid \phi_1 \rightarrow \phi_2 \mid \exists \phi_1 U_{\sim c} \phi_2 \mid \forall \phi_1 U_{\sim c} \phi_2 \mid \exists x.\ \phi \mid \forall x.\ \phi$$

ここで，

$x \in V$（時間変数）

$ap \in AP$（原始命題）

$c \in N$（自然数）

$\sim : <, \leqq, =, \geqq, >$（二項関係）

$\phi(x)$は自由変数xを持っており，$x.\ \phi(x)$により束縛される．

また，以下の時相表現も可能である．

(1)$\exists\Diamond_{\sim c}\phi = \exists(\text{true}\, U_{\sim c}\phi)$

(2)$\forall\Diamond_{\sim c}\phi = \forall(\text{true}\, U_{\sim c}\phi)$

(3)$\exists\Box_{\sim c}\phi = \neg\forall\Diamond_{\sim c}\neg\phi$

(4)$\forall\Box_{\sim c}\phi = \neg\exists\Diamond_{\sim c}\neg\phi$

ここで，

$\Diamond$：ある性質がいつかは成り立つ

$\Box$：ある過去の性質が常に成り立つ

なお，稠密モデルでは次の状態が決まらないので，nextオペレータは定義できない．■

［例1］（拡張TCTLによる検証性質記述例）

(1)bounded temporal operatorによる検証性質記述

$\exists \phi_1 U_{<c} \phi_2$

ϕ_2が最後に成立してその間はϕ_1が成立するような時間cより小さいようなある計算パスが存在する．

(2)freeze quantificationによる検証性質記述（その1）

$\exists x.\ \Diamond y.\ (y \leqq x + 5 \wedge \phi)$

5時刻以内でϕが成り立つ計算パスが存在する．この性質はbounded temporal operatorで以下のように表現できる．

$\exists\Diamond_{\leqq 5}\phi = \exists(\text{true}\, U_{\leqq 5}\phi)$

この場合のような隣接した命題間のタイミング制約記述はbounded temporal operator記述のほうが簡潔に表現できる．

(3)freeze quantificationによる検証性質記述（その2）

$\forall\Box x.\ (\phi_1 \rightarrow \forall\Diamond(\phi_2 \wedge \forall\Diamond y.\ (\phi_3 \wedge y \leqq x + 5)))$

すべての計算パスにおいて以下が成り立つ．すべてのϕ_1はϕ_2が続いて，その後ϕ_3が続く．ϕ_1とϕ_3の時間間隔は5時刻以下である．この性質はbounded temporal operatorでは表現できない．■

2．2　拡張TCTLの意味

タイミング制約記述を含まない場合のKripke構造では，計算パスは状態のωシーケンスである．しかし，稠密時間モデルの計算パスは時間Kripke構造の時間付き状態のωシーケンスとして定義する必要がある．

［定義2］（時間Kripke構造）

時間Kripke構造は$T = (S', R', P', \pi')$により以下のように定義する．

ここで，

S'：状態の有限集合

R'：$S' \times S'$の状態遷移関係

P'：$S' \rightarrow 2^{AP}$は各状態に成立する原始命題を割り付けるラベリング関数（AP：原始命題）

π'：$S' \rightarrow 2^{\delta}$

(1)状態遷移関係R'が生成する状態列S^{ω}は以下のとおりである．

$$S^{\omega} = S0' \xrightarrow{t0'} S1' \xrightarrow{t1'} S2' \xrightarrow{t2'} S3' \xrightarrow{t3'} \cdots$$

ここで，

$Si' \in S'$

$ti' \in \text{real}\ (i = 1, 2, 3, \cdots)$

(2)タイミング制約δはクロック集合Xと整数の時刻定数dにより以下のように帰納的に定義する．

$\delta ::= X < d \mid d < X \mid \neg\delta \mid \delta_1 \wedge \delta_2 \mid X := 0$■

次に，時間Kripke構造の状態列S^{ω}に対して，拡張TCTLの意味を定義する．

［定義3］（拡張TCTLの意味）

時間Kripke構造の状態列S^{ω}に対して，状態$S0'$が拡張TCTLϕを充足することを以下のように表現する．

$(S0', S^{\omega}, E) \models \phi$

ここで，

E：時間変数→real

以下では$(S0', S^{\omega}, E) \models \phi$を$S0' \models \phi$を略記する．

(1)$S0' \models ap$
iff $P'(S0') \ni ap$
(2)$S0' \models \pi_1 \leqq \pi_2$
iff $E(\pi_1) \leqq E(\pi_2)$
(3)$S0' \models \neg\phi$
iff $S0' \not\models \phi$
(4)$S0' \models \phi_1 \rightarrow \phi_2$
iff $S0' \not\models \phi_1$ または $S0' \models \phi_2$
(5)$S0' \models \exists \phi_1 U_{\sim c} \phi_2$
iff ある状態列S^ωに対して，以下の(a)～(c)が成り立つ．
(a)Si' $(1 \leqq i < n) \models \phi_1$
(b)$Sn' \models \phi_2$
(c)$\upsilon_n \models u \sim c$が成り立つ
（Sn'の時のクロック変数の割り当て）
（uはリセットされないクロック変数）
(6)$S0' \models \forall \phi_1 U_{\sim c} \phi_2$
iff すべての状態列S^ωに対して，以下の(a)～(c)が成り立つ．
(a)Si' $(1 \leqq i < n) \models \phi_1$
(b)$Sn' \models \phi_2$
(c)$\upsilon_n \models u \sim c$が成り立つ
（Sn'の時のクロック変数の割り当て）
（uはリセットされないクロック変数）
(7)$S0' \models \exists x.\ \phi$
iff ある状態列S^ωに対して，
$S0' \models_{[x:=0]} \phi$
(8)$S0' \models \forall x.\ \phi$
iff すべての状態列S^ωに対して，
$S0' \models_{[x:=0]} \phi$ ■

3．リアルタイムモデルチェッキング

リアルタイムモデルチェッキングによるタイミング検証は，時間オートマトンによるシステム仕様から時間Kripke構造を自動生成して，不等式手法とラベリングアルゴリズム[9]の融合により実現する．

3．1　システム仕様

各々のプロセスを時間Buchiオートマトン[2]により記述して，システム仕様は時間Buchiオートマトンの並行動作として表現する．

［定義4］（時間Buchiオートマトン）
時間Buchiオートマトンは（Σ，S，S0，C，E,F）の6つ組で定義される．
ここで，
Σ：有限イベント集合
S：有限状態集合
$S0 \subseteq S$：初期状態集合
C：クロックの有限集合
$E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$：時間付遷移関数
$F \subseteq S$：受理状態集合
$\Phi(C)$：クロックCのタイミング制約式δ
δはクロック集合Cと整数の時刻定数dにより帰納的に定義される：
$\delta ::= C < d \mid d < C \mid \neg\delta \mid \delta_1 \wedge \delta_2 \mid X := 0.$
イベントΣ^ωによるオートマトンの無限の状態列$inf(r)$が$inf(r) \cap F \neq 0$ならば，$inf(r)$は受理状態である．■

時間Buchiオートマトンの並行動作表現は積演算をであり，時間Buchiオートマトンは積演算閉包性がある[2]．

モデルチェッキングで検証するためには，システム仕様を時間Kripke構造として解釈する必要がある．時間Kripke構造の生成手法はシステム仕様のタイミング制約が付いた枝を状態として各イベントを原始命題と見なす[10]．

3．2　検証アルゴリズム

モデルチェッキングはリアルタイム分岐時相論理の拡張TCTL(Timed CTL)の時間Kripke構造へのラベリングであり，タイミング制約を考慮する必要がある．本稿では，不等式手法とラベリングアルゴリズムの融合によるタイミング検証方式を提案して，状態空間の組み合わせ爆発を抑制する．

タイミング検証方式は以下の手順から構成する．
(1)個々の分散プロセスの時間オートマトンからシステム仕様を自動合成する．
(2)時間Buchiオートマトンから時間Kripke構造を生成する．
(3)タイミング制約を考慮しないラベリングを行う．部分論理式をラベリングした状態列毎に以下の(4)～(6)を行う．
(4)各状態毎に成立するクロック制約により，DBM (Difference Bounds Matrices)[11),12)]を生成する．
(5)Floyd-Warshallのアルゴリズム[7]により，正規形DBMを求める．
(6)状態遷移元と状態遷移先の正規形DBMのintersectionを生成して，クロック制約の到達関係をチェックする．もし，正規形DBMの間に到達関係があれば，該当のラベリング状態列は存在する．そうでなければ該当のラベリング状態列は存在しない．

以下では，(3)～(6)を説明する．まず，DBMを定義する．

［定義5］（DBM(Difference Bounds Matrices)）
DBMは各状態に成立するクロック制約割り当てのマトリックスである．クロック制約割り当ての形式は以下のとおりである．
$t_i - t_j \leqq d_{ij}$
ここで，
t_i，t_j：クロック変数
d_{ij}：クロック定数
DBMの（i，j）成分はd_{ij}である．
なお，仮想クロック$t_0 = 0$と定義する．■

次に，クロック制約の到達関係の判定法を定義する．

［定義6］（クロック制約の到達関係の判定）
状態遷移元と状態遷移先の正規形DBMのintersectionを生成して，クロック制約の到達関係をチェックする．
(1)Floyd-Warshallのアルゴリズムにより各状態の正規形DBMを求める．
(2)状態遷移系列において，状態遷移元と状態遷移先の正規形(canonical form)DBMのintersectionを生成する．
intersection　$DBM = \min\{d_{ij}, d'_{ij}\}$
ここで，
$[d_{ij}]$：状態遷移元の正規形DBM
$[d'_{ij}]$：状態遷移先の正規形DBM
(3)intersectionDBMに負のパスが存在すれば到達関係を充足しない．そうでなければ到達関係を充足する．■

以下にモデルチェッキングアルゴリズムを定義する．

［定義7］（モデルチェッキングアルゴリズム）
時間Kripke構造$T = (S', R', P', \pi')$をTCTLの時相論理式ψでラベリングする．CTLの場合[9]と同様に，第1段階ではψの長さ1の部分論理式をラベリングして，第2段階ではψの長さ2の部分論理式をラベリングして，以下同様である．最後に，時相論

理式ϕの長さの論理式をラベリングして終了する．各部分論理式ϕに対するラベリングアルゴリズムは以下のように定義できる．

(1)$\phi \in$ap（原始命題）のとき
$\phi \in P'$（S0′）で矛盾がなければ，ϕでTをラベリングする．そうでないならば$\neg\phi$でラベリングする．

(2)$\phi = \pi_1 \leqq \pi_2$ のとき
E（π_1）≦E（π_2）で矛盾がなければ，ϕでTをラベリングする．そうでないならば$\neg\phi$でラベリングする．

(3)$\phi = \neg\phi$
$\neg\phi$でTをラベリングする．

(4)$\phi = \phi_1 \rightarrow \phi_2$
Tが$\neg\phi_1$またはϕ_2でラベリングされればϕでラベリングする．そうでなければ$\neg\phi$でラベリングする．

(5)$\phi = \exists \phi_1 U_{\sim c} \phi_2$
ある状態列S$^\omega$に対して，以下の(a)～(d)が成り立つ．
(a)Si′（1≦i＜n）がϕ_1でラベリングされる．
(b)Sn′がϕ_2でラベリングされる．
(c)$\upsilon_n \models$u～cが成り立つ
（Sn′の時のクロック変数の割り当て）
（uはリセットされないクロック変数）
(d)DBMが状態遷移関係を充足する

(6)$\phi = \forall \phi_1 U_{\sim c} \phi_2$
すべての状態列S$^\omega$に対して，以下の(a)～(d)が成り立つ．
(a)Si′（1≦i＜n）がϕ_1でラベリングされる．
(b)Sn′がϕ_2でラベリングされる．
(c)$\upsilon_n \models$u～cが成り立つ
（Sn′の時のクロック変数の割り当て）
（uはリセットされないクロック変数）
(d)DBMが状態遷移関係を充足する

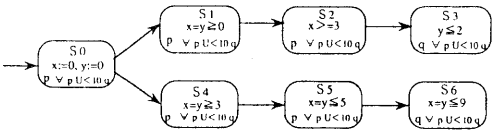
(7)$\phi = \exists x.\ \phi$
ある状態列S$^\omega$がϕ［x：＝0］でラベリングされてDBMが状態遷移関係を充足するならば，ϕでTをラベリングする．そうでないならば$\neg\phi$でラベリングする．

(8)$\phi = \forall x.\ \phi$
すべての状態列S$^\omega$がϕ［x：＝0］でラベリングされてDBMが状態遷移関係を充足するならば，ϕでTをラベリングする．そうでないならば$\neg\phi$でラベリングする．■

次に，モデルチェッキングの例を示す．

［例1］（モデルチェッキングの例）

拡張TCTL式$\forall$ p $U_{<10}$qのラベリングアルゴリズムにより，図1(1)の時間Kripke構造が生成できたとする．次に，ラベリング状態列のタイミング制約を計算する必要がある．ラベリングするときに，不等式を生成してDBMを作成する．この場合，4つの状態列のパスがあるので，4つのDBMを作成してタイミング制約の充足性を判定する．図1(2)のcanonical form DBMのD3とD4のintersectionDBMに負のパスが存在するために，状態遷移関係S3→S4はタイミング制約を充足しないことがわかる．ゆえに，拡張TCTL式$\forall$ p $U_{<10}$qは充足されない．なお，DBM中の＊はパスが存在しない成分である．■



(1)時間Kripke構造の状態列
(1)State sequences of timed Kripke structure

At S0, from x=y=0
DBM D1=0＊＊ / 0＊＊ 正規形 DBM=0 0 0 / 0 0 0 / 0 0 0

At S1, from x=y≧0.
DBM D2=＊ 0 0 / ＊ ＊ 0 / ＊ 0 ＊ 正規形 DBM=＊ 0 0 / ＊ ＊ 0 / ＊ 0 0

At S2, from x=y≧3.
DBM D3=＊ ＊ 0 / ＊ ＊ 0 / ＊ 0 ＊ 正規形 DBM=＊ 0 0 / ＊ ＊ 0 / ＊ 0 0

At S3, from x=y≦2.
DBM D4=2 ＊ 0 / 2 ＊ 0 / 2 0 ＊ 正規形 DBM=2 0 0 / 2 0 0 / 2 0 0

At S2→S3, from x=y≦2 after x=y≧3.
intersection D3∩D4 DBM=2 ＊ 0 / 2 ＊ 0 / 2 0 ＊ 正規形 DBM=-3-6 -9 / -6-9-12

ここで
＊：当該成分の不等式が存在しない

(2)DBMによる到達可能解析の例
(2)Example of reachability analysis by DBM

図1 リアルタイムモデルチェッキングの例
Fig.1 Example of real-time model checking

4．検証システムとタイミング検証の事例

4．1 検証システム

本稿で説明した手法を支援する検証システムを実現した．検証システムは図2に示すようにコンパイラや時間Kripke構造生成器，モデルチェッカから構成して，約3kstepで実現した．コンパイラは実時間分散ソフトウェアの仕様から時間Buchiオートマトンを生成する．時間Kripke構造生成器は時間Buchiオートマトンから時間Kripke構造を生成する．モデルチェッカは時間Kripke構造が検証性質仕様（時相論理式）を充足するかどうかを検証する．

4．2 検証事例

本稿では，イーサネットのCSMA/CDにより提案した検証手法と検証システムの有効性を示す．

(1)問題の概要

イーサネットのCSMA/CD[13]はLANで広く使われており，以下の特徴を有する．送信局はデータを送信すると，チャネルの応答を感知する．もし，チャネルがアイドルならば送信局はデータを送信する．しかし，チャネルがbusyであったりデータが破壊したら，ある時間待って再送する．以上のイーサネットのCSMA/CDプロトコルは図3の時間ステートチャートにより仕様記述できる．

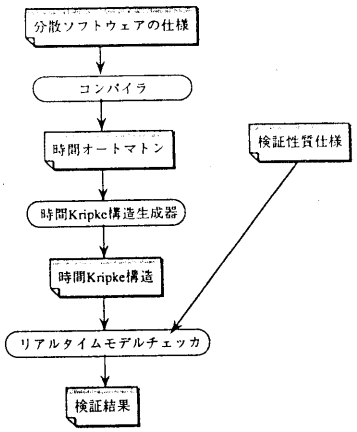


図2　検証システムの構成
Fig.2　Configuration of verification system

検証性質記述は図4に示す．(1)はリアルタイム制約時間含意と呼ばれる活性の代表的な事例である．この性質はfreeze quantifierによる表現であり，(b)～(d)は従来のbounded temporal logicのTCTLでは記述できない．(2)はリアルタイム境界時間含意と呼ばれる活性の代表的な事例である．この性質はbounded temporal logicのTCTLによる表現である．(3)は不変性と呼ばれる安全性の代表的な事例である．

(a) ∃◇ready→(∃◇ x .begin→∃◇ y .(end∧ y ≦ x +20))

(b) ∃◇ x .(send→∃◇ (cd∧∃ y .(end∧ y ≦ x +20)))

(c) ∃◇ x .(send→∃◇ (cd∧∃◇ y .(end∧ y ≦ x +20)))

(d) ∃◇ x .(ready→∃◇ (cd∧∃◇ y .(end∧ y ≦ x +20)))

(1)リアルタイム制約時間含意（活性）
(1)real-time freeze quantifier temporal implication(liveness)

(a) ∀□(send→EF≦30 end)

(b) ∀□ send→∀◇≦15 end

(c) ∃◇(ready→EF≦10 end)

(2)リアルタイム境界時間含意（活性）
(2)real-time bounded temporal implication(liveness)

(a) ∀□(send→ end)

(3)不変性（安全性）
(3)invariance(safety

図4　検証性質仕様
Fig.4　verification property specification

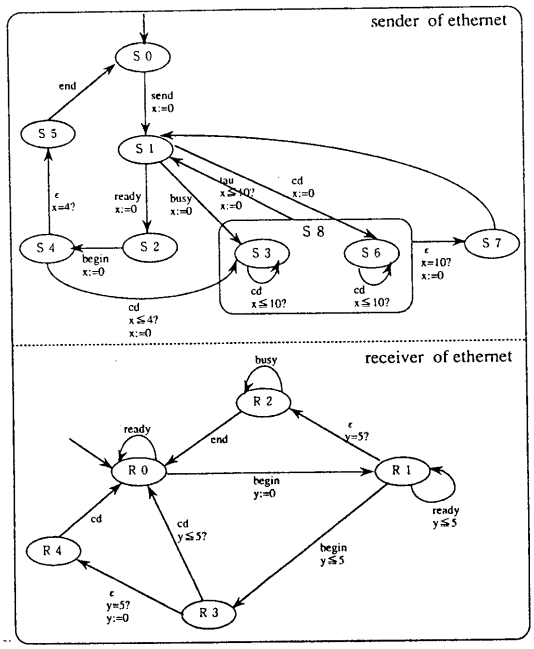


図3　イーサネットの仕様記述
Fig.3　specification of ethernet

(2)検証

検証システムに仕様と検証性質仕様を入力すると，検証結果が得られる．検証システムへの入力形式は仕様のプログラミング言語形式であり，図5に例を示す．

```
System specification ;
System configuration ;
  system=sender×receiver ;
Process specification(sender) ;
State definition part  S0, S1, ···, S7 ;
Event definition part  send, tau, ready, cd, busy, begin, end ;
Initial state definition part  S0 ;
State transition definition part
 S0→send, x:=0→S1 ;
 S1→busy, x:=0→S3 ;
 S1→cd, x:=0→S6 ;
    :
    :
end ;
Process specification(receiver) ;
State definition part  R0, R1, ···, R4 ;
Event definition part  begin, cd, ready, cd ;
Initial state definition part  R0 ;
State transition definition part
 R0→ready→R0 ;
 R0→begin, y:=0→R1 ;
 R1→begin, y≦5!→R3 ;
    :
    :
    :
end ;
```

図5　コンパイラへの入力データ例
Fig.5　Example of input data of compiler

検証コストは以下のように考えられる．モデルチェッキングはラベリングアルゴリズムとDBMによる到達可能解析の融合であり，計算時間は
O（|論理式|×（ノード数＋エッジ数））
　＋O（ラベリングの状態数×（DBMの大きさ）^{3}）
である．メモリ量は
O（ノード数＋エッジ数）
　＋O（ラベリングの状態数×（DBMの大きさ）^{2}）
である．今回はsenderとreceiverを増加させた仕様に対して，図４の検証性質仕様を検証した．時間Kripke構造の状態数は122～1213であり，状態遷移数は338～5472であった．検証コストの実測結果を表１に示す．計算時間とメモリ量は時間Kripke構造のノード数とエッジ数が増えるとともに増加する．時相オペレータ∀と∃に関しては時間Kripke構造の深さや横への広がりに関係しており，∀の論理式が∃の論理式よりも計算時間がかかるがメモリ量はほぼ同じである．

表１　検証システムによるethernetの検証コスト
Table 1 Verification cost of ethernet by verification system

検証性質	結果	状態数	遷移数	CPU時間	メモリ量
∃◇ready→(∃◇ x .begin→∃◇ y .(end∧ y ≦ x +20))	true	122	338	5.9sec	585kb
	true	870	3801	16.7sec	747kb
	true	942	3915	19.5sec	751kb
	true	1213	5472	27.0sec	817kb
∃◇ x .(send→∃◇ (cd∧∃ y .(end∧ y ≦ x +20)))	true	122	338	6.0sec	585kb
	true	870	3801	17.0sec	720kb
	true	942	3915	18.4sec	758kb
	true	1213	5472	26.8sec	748kb
∃◇ x .(send→∃◇ (cd∧∃◇ y .(end∧ y ≦ x +20)))	true	122	338	6.3sec	585kb
	true	870	3801	23.8sec	742kb
	true	942	3915	23.6sec	760kb
	true	1213	5472	37.9sec	813kb
∃◇ x .(ready→∃◇ (cd∧∃◇ y .(end∧ y ≦ x +20)))	true	122	338	6.1sec	585kb
	true	870	3801	23.8sec	744kb
	true	942	3915	23.5sec	759kb
	true	1213	5472	38.0sec	782kb
∀□(send→∃◇≦10 end)	false	122	338	6.0sec	585kb
	true	870	3801	21.9sec	745kb
	false	942	3915	19.6sec	759kb
	true	1213	5472	38.5sec	818kb
∀□ send→∀◇≦15 end	true	122	338	6.4sec	587kb
	true	870	3801	17.2sec	732kb
	true	942	3915	18.8sec	759kb
	true	1213	5472	27.3sec	818kb
∃◇(ready→∃◇≦10 end)	true	122	338	5.7sec	585kb
	true	870	3801	15.9sec	744kb
	true	942	3915	18.1sec	760kb
	true	1213	5472	26.8sec	818kb
∀□(send→ end)	true	122	338	5.8sec	586kb
	true	870	3801	22.5sec	745kb
	true	942	3915	18.6sec	760kb
	true	1213	5472	38.4sec	818kb

５．むすび

本稿では，分散ソフトウェアの新しいタイミング検証方式により，隣接しない命題論理間の効率的なタイミング検証が実現できた．主要な特徴は以下のとおりである．
(1)稠密時間モデル上のfreeze quantificationとbounded temporal operatorの融合により，簡潔で表現力の高いタイミング制約表現が可能となった．
(2)システム仕様は時間オートマトンの並行合成により記述して，時間Kripke構造を自動生成した．
(3)モデルチェッキングは不等式手法とラベリングアルゴリズムの融合により実現した．

今後の主要な課題は以下が考えられる．
(1)本手法の実問題への適用により，検証コストや表現力などを検討する
(2)公平性が記述できるように時相論理の表現力を向上させる

参考文献

1)Alur R. Henzinger H.A.:"Logics and Models of Real Time:A Survey",LNCS(Lecture Notes in Computer Science)600,pp.74-106(1992)
2)Alur R. Dill D:"The Theory of Timed Automata",LNCS 600,pp.45-73(1992)
3)Emerson E.A. Mok A.K. Sistla A.P. Srinivasan J. : "Quantitative Temporal Reasoning",LNCS531,pp.136-145(1990)
4)Alur R. Henzinger H.A.:"A Really Temporal Logic",Journal of ACM,Vol.41,No.1,pp.181-204(1994)
5)Harel E. Lichtenstein O. Pnueli A. :"Explicit Clock Temporal Logic",Proc. 5th Logic in Computer Science (LICS),pp.402-413(1990)
6)Alur R. Courcoubetis C. Dill D:"Model-Checking for Real Time Systems",Proc. 5th IEEE Symp. on Logic In Computer Science,pp.414-425(1990)
7)Rokicki T.G.:Representing and Modeling Circuits,PhD thesis,Stanford University(1993)
8)Rokicki T.G. Myers C.J.:"Automatic Verification of Timed Circuits",LNCS 818,pp.468-480(1994)
9)Clarke E.M. Emerson E.A. Sistla A.P.:"Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications",ACM Trans. on Programming Languages and Systems,Vol.8,N0.2,pp.244-263 (1986)
10)Hiraishi H.:"Design verification of sequential machines based on ϵ-free regular temporal logic",Computer hardware description languages and their applications", pp.249-263,Elsevier science publishers(1990)
11)Dill D.:"Timing assumptions and verification of finite-state concurrent systems",LNCS 407,pp.197-212(1989)
12)Alur R. Courcoubetis C. Dill D. Halbwachs Wong-Toi H.:"An Implementation of Three Algorithms for Timing Verification Based on Automata Emptiness",Proc. Real-Time Systems Symposium,pp.157-166(1992)
13)IEEE ANSI/IEEE 802.3, ISO/DIS 8802/3. IEEE Computer Society Press(1985)