マルチメディア通信と分散処理 71−21
（１９９５．７．１４）

実時間分散ソフトウェアの可能性と公平性の検証

山根 智* 川平靖博* 翁 崇恩* 将 海雲+
*島根大学理学部情報科学 +元 南京通信研究所

実時間分散ソフトウェアではタイミング制約記述を含む可能性と公平性の検証の手法が必要である．しかし，従来の検証手法では可能性と公平性を同時に検証できる検証システムの研究事例はない．本稿では，以下の検証の手法により，可能性と公平性を効率的に検証できるシステムを開発した．
(1)モデルチェッキングと言語包合アルゴリズムを同時に実現して，タイミング制約記述を含む可能性と公平性を検証可能とした．
(2)状態空間の組み合わせ爆発を回避するために，時間不等式手法を基礎として検証システムを構築した．

Verification of probabilities and fairness in real-time distributed software

Satoshi Yamane* Yasuhiro Kawahira* Suon Oh* Kaiun Chan+
* Dept. of computer science Shimane University +Nankin Commmunication Lab.

It is necessary to verify both possibilities and fairness including timing constraints in real-time distributed software. But the verification system, which can verify both possibilities and fairness, has not been proposed before. In this paper, we have developed the verification system , which can verify both possibilities and fairness based on the following methods.
(1)Possibilities and fairness can be verified by both model checking and language inclusion algorithm.
(2)Timing verification method is based on inequality method in order to avoid state explosion problem.

１．はじめに

航空機や通信機器などのような実時間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い．つまり，プロセスの論理動作の正当性だけでなく，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．このために，タイミング制約記述を含む適切な仕様記述と検証の手法が必要である[1]．以下の本稿では，実時間分散ソフトウェアを対象として議論する．従来より，分散ソフトウェアの仕様記述手法としては，構造化分析やペトリネット，オートマトン，プロセス代数，時相論理，モードチャートなどが研究されている[1]．しかし，従来の仕様記述手法では，形式性の欠如や検証能力の欠如，記述能力の欠如などの問題点がある．

実時間分散ソフトウェアは時間グラフ[2]や時間オートマトン[3]で仕様記述できる．また，実時間分散ソフトウェアの検証のアプローチとしては，モデルチェッキングや言語包合アルゴリズムがある[4]．モデルチェッキングはKripke構造（仕様）が時相論理式（検証仕様）を充足するかどうかを判定するものである．言語包合アルゴリズムは仕様のオートマトンが検証仕様のオートマトンに包合されるかどうかを判定するものである．しかしながら，従来の仕様記述と検証には以下の問題点がある．
(1)検証時に大規模なリージョングラフ[2),3)]を生成する必要があり，計算時間やメモリ量のコストが大き過ぎる．
(2)モデルチェッキングは∀（すべての計算パス）や∃（ある計算パス）で充足されるといった可能性の検証はできるが，公平性の検証は完全にはできない[2)]．
(3)言語包合アルゴリズムは公平性の検証はできるが，可能性の検証ができない[5)]．

(1)を解決する手法としては，時間不等式による検証方式[6),7)]が提案されている．(2)と(3)により，可能性と公平性を検証できる実時間検証システムは存在しない．(2)に関しては，実時間制約のない場合，CTL*では$F^{\infty}p$（GFp：infinitely often）や$G^{\infty}p$（FGp：almost everywhere）により，公平性（$\wedge_i F^{\infty}executed_i$）や弱公平性（$\wedge_i F^{\infty}(\neg enabled_i \vee executed_i)$），強公平性（$\wedge_i (F^{\infty}enabled_i \rightarrow F^{\infty}executed_i)$）などが検証できる[8)]．しかし，実時間の公平性はモデルチェッキングでは完全には検証できない[2)]．(3)に関しては，オートマトンでは$F^{\infty}p$などが記述可能であり，言語包合アルゴリズムは公平性が検証できる[9)]．しかし，オートマトンでは∀や∃は検証できない．

本稿では，モデルチェッキングと言語包合アルゴリズムによる検証を同時に支援するシステムを開発したので報告する．この検証システムにより，実時間分散ソフトウェアの可能性と公平性が同時に検証可能となった．また，状態空間の組み合わせ爆発を回避するために，時間不等式手法を基礎として実時間分散ソフトウェアを検証した．

以下の本稿は，２章では仕様記述手法を述べ，３章～５章では検証手法を提案する．６章では検証システムとその有効性を示す．最後に，７章ではまとめと今後の課題を示す．

２．実時間分散ソフトウェアの仕様記述手法

実時間分散ソフトウェアは並行動作する複数のプロセスから構成される．各々のプロセスを時間Buchiオートマトン[3)]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１］（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⊆S：初期状態集合

C：クロックの有限集合

E⊆S×S×Σ×2^{C}×Φ（C）：時間付遷移関数

F⊆S：受理状態集合

Φ（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：

δ：＝C＜d｜d＜C｜¬δ｜$\delta_1 \wedge \delta_2$｜X：＝0．

イベントΣ$^{\omega}$によるオートマトンの無限の状態列inf（r）がinf（r）∩F≠0ならば，inf（r）は受理状態である．■

［定義２］（システム仕様）

システム仕様はプロセスの並行動作で定義する．

S＝P１×P２×…×Pn

ここで，

S：システム仕様

Pi（i＝１，…，n）：プロセス仕様

×：カルテジアン積 ■

［定理１］（時間Buchiオートマトンの積演算閉包性）

時間Buchiオートマトンは積演算に関して閉包性がある．

（証明）

２つの時間BuchiオートマトンM1，M2の受理言語L（M1），L（M2）の積の演算

L（M1）∩L（M2）

を受理する時間Buchiオートマトンが構成できる．これにより，時間Buchiオートマトンは積演算に関して閉じていることが証明できる．■

以上より，システム仕様は時間Buchiオートマトンで定義できる．

３．検証手法の概要

可能性と公平性の概念を検証するために，検証手法はモデルチェッキングと言語包含アルゴリズムから構成する．検証手法は以下の手続きで構成する．

(1)まず，個々のプロセス仕様から積時間Buchiオートマトンを生成する．

(2)可能性の検証のときは，積時間Buchiオートマトンから時間Kripke構造を生成して，検証性質仕様はリアルタイム時相論理式で定義する．リアルタイムモデルチェッキングにより，時間Kripke構造がリアルタイム時相論理式を充足するかどうかを判定する．

(3)公平性のときは，検証性質仕様は決定性時間Muller オートマトンで定義する．言語包含アルゴリズムにより，積時間Buchiオートマトンが決定性時間Mullerオートマトンに包含されるかどうかを判定する．

４．可能性の検証手法

４．１　時間Kripke構造の生成

モデルチェッキングをするためには，積時間Buchi オートマトンから時間Kripke構造を生成する必要がある．時間Kripke構造の生成は積時間Buchiオートマトン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ことにより実現できる[10]．

［定義３］（時間Kripke構造の構成）

時間Kripke構造はT＝（S′，R′，P′，S０′）により以下のように定義できる．

(1)S′＝｛＜s，δ＞｜s∈S，δ∈Φ（C）｝

(2)R′⊆S′×S′

(3)P′：S′→2^{Σ}

(4)S０′＝｛＜s0，δ0＞｜δ0:全てのクロック変数がゼロ｝

ここで，δやΣなどの記号は［定義１］と同様と同様である．■

４．２　リアルタイム時相論理

検証性質記述言語としては，リアルタイム時相論理としてTCTL(Timed CTL)[2]を考える．

［定義４］（TCTLの構文）

TCTLの時相論理式φの構文を次のように帰納的に定義する：

φ：＝ap｜¬φ｜$\phi_1 \rightarrow \phi_2$｜$\exists \phi_1 U_{\sim c} \phi_2$｜$\forall \phi_1 U_{\sim c} \phi_2$

ここで，ap∈AP（原始命題）

c∈N（自然数）

～は二項関係＜，≦，＝，≧，＞

また，以下の時相表現も可能である．

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

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

(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，

◇：ある性質がいつかは成り立つ

□：ある過去の性質が常に成り立つ ■

具体的には，$\exists \phi_1 U_{<c} \phi_2$は次のように解釈される：$\phi_2$が最後に成立してその間は$\phi_1$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．

４．３　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はリアルタイム時相論理式の時間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り，状態空間の組み合わせ爆発を抑制する．リアルタイムモデルチェッキングは以下の手続きで構成する．

(1)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(2)～(4)を行う．

(2)各状態毎に成立するクロック制約により，DBM (Difference Bounds Matrices)[6]を生成する．

(3)Floyd-Warshallのアルゴリズムにより，正規形DBM[7]を求める．

(4)状態遷移元と状態遷移先の正規形DBMのintersection[7]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．

まず，DBMを形式的に定義する．

［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)intersection DBMの正規形に負のパスが存在すれば到達関係を充足しない．負のパスが存在しなければ到

達関係を充足する．■

以下にリアルタイムモデルチェッキングアルゴリズムを定義する．

［定義７］（リアルタイムモデルチェッキングアルゴリズム）

時間Kripke構造T＝（S′，R′，P′，S０′）をTCTLの時相論理式ϕでラベリングする．CTL[11]の場合と同様に，第１段階ではϕの長さ１の部分論理式をラベリングして，第２段階ではϕ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．初期状態が時相論理式ϕでラベリングされるならば，時相論理式ϕが充足されるとする．各部分論理式ψ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)ψ＝ａｐ（原始命題）のとき

$\psi \in$P′（ａｐ)ならば，ψでTをラベリングする．そうでないならば$\neg\psi$でラベリングする．

(2)$\psi = \neg\phi$のとき

Tを$\neg\phi$でラベリングする．

(3)$\psi = \phi_1 \rightarrow \phi_2$のとき

Tが$\neg\phi_1$またはϕ_2でラベリングされればψでラベリングする．そうでなければ$\neg\psi$でラベリングする．

(4)$\psi = Q\,\phi_1 U_{\sim c}\,\phi_2$（Qは∃又は∀）のとき

Tはϕ_1または$\neg\phi_1$とϕ_2または$\neg\phi_2$でラベリングされる．もし，∃の場合はある公平（∀の場合はすべての公平）な状態列S'_1，S'_2，…，S'_nがあり，

(a)S'_i（$1 \leq i < n$）がϕ_1でラベリングされる

(b)S'_nがϕ_2でラベリングされる．さらに，ϕ_1でラベリングされる可能性がある．

(c)υ_n｜＝ｕ～ｃが成り立つ（S'_nの時のクロック変数の割り当て）

(d)DBMが状態遷移関係を充足する

ときは，Tはψでラベリングされる．そうでなければTは$\neg\psi$でラベリングされる．ここで，ｕはタイミング制約を計測するためのリセットされないクロック変数である．■

［例１］（リアルタイムモデルチェッキングの例）

TCTL式$\forall\, p\, U_{<10} q$などの可能性の概念の検証を考える．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，ラベリング状態列がタイミング制約を充足して状態遷移可能かどうかを検証する必要がある．まず，Ｓ０→Ｓ１→Ｓ２→Ｓ３の状態列が状態遷移可能かどうかを検証する．図１(2)のように各状態のDBMを作成する．次に，各DBMの正規形DBMを求めて到達可能解析を行う．図１(2)に示すように，D3とD4のintersection DBMの正規形に負のパスが存在するために，状態遷移関係Ｓ２→Ｓ３は状態遷移不可能なことがわかる．また，同様な手法によりＳ０→Ｓ４→Ｓ５→Ｓ６の到達可能解析を行うと，状態遷移可能であった．ゆえに，TCTL式$\forall\, p\, U_{<10} q$は充足されない．しかし，TCTL式$\exists\, p\, U_{<10} q$は充足される．■

５．公平性の検証手法

５．１　検証性質仕様記述言語

言語包含アルゴリズムでは，検証性質記述のオートマトンは補集合演算閉包性が必要となる．本稿では補集合演算閉包性を満たす決定性時間Mullerオートマトンを検証性質記述言語として考える．

［定義８］（検証性質記述言語）

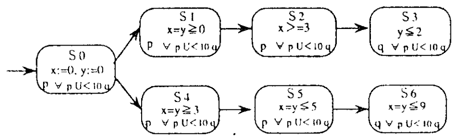
決定性時間Mullerオートマトンは（Σ，Ｓ，S0，C，Ｅ，Ｆ）の６つ組で定義される．

ここで，

Σ：有限イベント集合

Ｓ：有限状態集合

S0⊆Ｓ：初期状態集合



(1)時間Kripke構造の状態列

(1)State sequences of timed Kripke structure

At S0, from x=y=0

	*00		000
DBM D1=	0**	正規形 DBM=	000
	0**		000

At S1, from x=y≧0.

	*00		*00
DBM D2=	**0	正規形 DBM=	*00
	0		*00

At S2, from x=y≧3.

	*-3-3		*-3-3
DBM D3=	**0	正規形 DBM=	*00
	0		*00

At S3, from x=y≦2.

	***		***
DBM D4=	2*0	正規形 DBM=	200
	20*		200

At S2→S3, from x=y≦2 after x=y≧3.

	*-3-3		-5-8-11
intersection D3∩D4 DBM=	2*0	正規形 DBM=	-3-6 -9
	20*		-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

Ｃ：クロックの有限集合

E⊆Ｓ×Ｓ×Σ×2^C×Φ（Ｃ）：状態遷移関数

F⊆2^S：受理状態集合のクラス

なお，｜S０｜＝１であり，状態遷移は決定的である．ここで，Φ（Ｃ）はクロックＣのタイミング制約式δであり，クロック集合Ｘと時刻定数ｄにより再帰的に定義される：

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

もし，走査列ｒの無限な状態集合inf(ｒ)がinf(ｒ)∈Ｆならば，走査列ｒは受理列である．■

次に，決定性時間Mullerオートマトンによる公平性の記述を説明する．

［定義９］（公平性の定義）

公平性は形式的に以下のように定義できる．

$$SF(L,U) = \{inf(r) \mid inf(r) \cap L \neq \phi \rightarrow inf(r) \cap U \neq \phi\}$$
$$= \{inf(r) \mid inf(r) \cap L = \phi \text{ 又は } inf(r) \cap U \neq \phi\}$$

ここで，

inf(r)：走査列ｒの無限な状態集合

Ｌ：enabled条件を持つ状態集合

Ｕ：executed条件を持つ状態集合　■

決定性時間Mullerオートマトンで公平性を記述するためには，ＬとＵの状態集合よりＳＦ(L,U)を求めて受理状態のクラスを定義すればよい．

［例２］（検証性質記述言語による公平性の記述例）

Kurshanの定義[19]に従って，決定性時間Mullerオートマトンで公平性を記述する．

(1)infinitely often ｘ →infinitely often ¬x

公平性のオートマトンは図２(1)で定義できる．次に，Ｌ＝｛Ｓ０｝，Ｕ＝｛Ｓ１｝と定義して受理状態のクラスを求める．

(a)inf(r)∩｛Ｓ０｝＝ϕのとき

inf(r)＝｛Ｓ１｝

(b)inf(r)∩｛Ｓ１｝≠ϕのとき

inf(r)＝｛Ｓ１｝又は｛Ｓ０，Ｓ１｝

以上より，受理状態のクラスは｛｛Ｓ１｝，｛Ｓ０，Ｓ１｝｝である．

(2)infinitely often ｘ →infinitely often ｙ

公平性のオートマトンは図２(2)のようにinfinitely often ｘ →infinitely often ¬xとinfinitely often ｙ →infinitely often ¬yのオートマトンのカルテジアン積で

定義できる．次に，L＝｛S 2,S 4｝U＝｛S 2S 3｝と定義して受理状態のクラスを求める．

(a)inf(r)∩｛S 2，S 4｝＝ϕのとき
　inf(r)＝｛S 1｝又は｛S 3｝又は｛S 1，S 3｝

(b)inf(r)∩｛S 2，S 3｝≠ϕのとき
　inf(r)＝｛S 2｝又は｛S 3｝又は｛S 1，S 2｝又は｛S 1，S 3｝又は…｛S 1，S 2，S 3，S 4｝

以上より，受理状態のクラスは｛｛S 1｝，｛S 2｝，…，｛S 1，S 2，S 3，S 4｝｝である．■

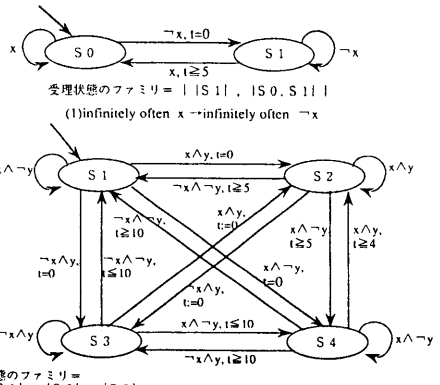


図2　決定性時間Mullerオートマトンによる公平性の記述例
Fig.2　Example of fairness by deterministic timed Muller automaton

5．2　言語包含アルゴリズム

言語包含アルゴリズムは，積時間Buchiオートマトンが決定性時間Mullerオートマトンに包含されるかどうかを判定する．まず，言語包含問題[4]を定義する．

［定義１０］（言語包含問題の定義）
言語包含問題は次のように定義できる．

　L（M１）⊆L（M２）

また，これは以下と等価である．

　L（M１）∩$\overline{\text{L（M２）}}$＝ϕ

ここで，

　M１：仕様の時間オートマトン
　L（M１）：時間オートマトンM１の受理言語
　M２：検証性質記述の時間オートマトン
　L（M２）：時間オートマトンM２の受理言語

なお，L（M２）は検証コストの都合により，補集合演算閉包性が必要である．■

次に，言語包含問題の決定性に関する定理を以下に説明する．

［定理２］（言語包含問題の決定性）
決定性時間Mullerオートマトンとの言語包含問題は決定性手続きが存在する．

（証明）
時間Buchiオートマトンは積演算で閉じているので，システム仕様は時間Buchiオートマトンである．時間Buchiオートマトンと決定性時間Mullerオートマトンとの言語包含問題は決定性手続きが存在することが知られている[3]．■

次に，時間Buchiオートマトンと決定性時間Mullerオートマトンの補集合との積時間オートマトンの生成を定義する[3]．

［定義１１］（積時間オートマトンの生成）
時間Buchiオートマトン（Σ，S1，S01，C1，E1，F1）と決定性時間Mullerオートマトン（Σ，S2，S02，C2，E2，F2）の補集合との積時間オートマトンを定義する．積時間オートマトンは（Σ，S1×S2，S01×S02，C1∪C2，E，F）で定義できる．すなわち，状態遷移関数Eは２つの状態遷移関数の積であり，状態集合はS1×S2である．また，初期状態はS01×S02であり，クロック変数は論理和C1∪C2であり，クロック制約は論理積δ1∧δ2である．ここで，２つのオートマトンの状態遷移を＜s1，t1，a，λ1，δ1＞と＜s2，t2，a，λ2，δ2＞とする．s1は遷移元の状態であり，t1は遷移先の状態である．a∈Σであり，λ1∈C1，λ2∈C2である．また，δ1とδ2は各々のクロック制約である．E⊆＜s1×s2，t1×t2，a，λ1∪λ2，δ1∧δ2＞である．また，決定性時間Mullerオートマトンの補集合は受理状態集合の補集合なので，受理状態集合のクラスFはF1×（2^{S2}－F2）である．■

言語包含アルゴリズムは積時間オートマトンの受理ループ発見問題に帰着できて，以下の手続きから構成する．

(1)タイミング制約を考慮しない積時間オートマトンの受理ループ発見[3]をTarjanの深さ優先探索[12]で行う．もし，受理ループが存在しなければ検証性質は充足される．もし，受理ループが存在すれば，モデルチェッキングの場合と同様な以下の(2)～(4)により到達可能解析を行って，受理ループの存在性を判定する．

(2)受理ループの状態列に成立するクロック制約割り当てにより，DBM(Difference Bounds Matrices)[6]を生成する

(3)Floyd-Warshallのアルゴリズムにより，正規形DBM[7]を求める．

(4)状態遷移元と状態遷移先の正規形DBMのintersection[7]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，該当受理ループは存在しない．もし，そうでなければ該当受理ループは存在する．もし，受理ループが存在しなければ検証性質は充足される．

次に，受理ループの発見アルゴリズムの概要を定義する．

［定義１２］（受理ループの発見アルゴリズム）

```
program closedpath ;
  var counter : integer ;        頂点の順序数
  var i : integer ;
/*  状態探索サブルーチン開始  */
 procedure visit(v : vindex) ;
  var z : vindex ;                頂点のリスト構造
  begin
     counter:=counter+1 ;
     with vertex[v] do
       begin seq:=counter ; searching:=true ; end ;
     for  v を始点とする辺各々について do
       begin
          z:=辺の行き先の頂点 ;
          if  vertex[z].seq=0
             then  visit(z) ;                          未訪問
             else  if vertex[z].seq>vertex[v].seq
                      else if vertex[z].searching      訪問済
                        then
                          begin
                            if 受理条件を満たすか
                             then
                               受理状態列のDBMを作る
                               if DBMの到達可能性があるか
                                  then
                                     受理ループが存在する
                          end ;
       end ;
    vertex[z].searching:=false ;
  end ;
/*  状態探索サブルーチン終了  */
begin
/*  初期化  */
  for  i:=1  to  n  do
    with  vertex[i]  do
      begin  seq:=0 ; searching:=false ; end ;
  counter:=0 ;
/*  探索開始  */
  初期状態で全クロックをゼロに割り当てる ;
    if  vertex[1].seq=0  then
      visit(1) ;
end.          ■
```

ここで，言語包含アルゴリズムの例を示す．

[例３]（言語包合アルゴリズムの例）
仕様と検証性質仕様が各々図３(1)(2)で与えられた場合の言語包合アルゴリズムによる公平性の検証の例を示す．検証性質仕様はinfinitely often a→infinitely often ¬a，すなわちinfinitely often a→infinitely often cであることを意味する．仕様の受理状態集合は｛S０，Ｓ１｝であり，検証性質仕様の受理状態集合のクラスは｛｛P１｝，｛Ｐ０，Ｐ１｝｝である．ゆえに，検証性質仕様の補集合の受理状態集合のクラスは｛｛Ｐ０｝｝である．

次に，図３(3)のように，仕様と検証性質仕様の補集合との積時間オートマトンを生成する．積時間オートマトンの受理状態集合のクラスは｛｛Ｓ０×Ｐ０，Ｓ１×Ｐ０｝｝である．深さ優先探索により，積時間オートマトンの受理ループを探索すると，以下の受理ループが発見できた．

（Ｓ０,Ｐ０）→（Ｓ１,Ｐ０）→（Ｓ０,Ｐ０）→…

次に，受理ループの到達可能解析により，受理ループが実時間制約を充足するかどうかを判定する．受理ループの到達可能解析の例を図３(4)に示す．まず，受理ループの各状態のDBMと正規形DBMを求める．次に，正規形DBMのintersectionを生成して，intersectionのDBMの正規形に負のパスが存在するかどうかを判定する．（Ｓ１,Ｐ０）→（Ｓ０,Ｐ０）のintersectionのDBMは負のパスが存在しないので，上記受理ループは実時間制約に矛盾しないで状態遷移が発生する．以上より，受理ループは存在するので，検証性質仕様は充足されない．■

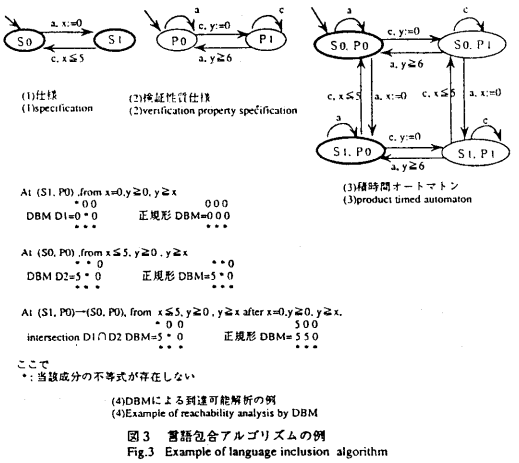


図３　言語包合アルゴリズムの例
Fig.3　Example of language inclusion algorithm

６．検証システムと検証事例

６．１　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４に示すようにコンパイラや時間Kripke構造生成器，言語包合検証器，モデルチェッカから構成して，各々約１kstep，0.3kstep，1kstep，2kstep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積時間Buchiオートマトンを生成する．時間Kripke構造生成器は積時間Buchiオートマトンから時間Kripke構造を生成する．言語包合検証器は積時間Buchiオートマトンが検証性質仕様（オートマトン）を充足するかどうかを検証する．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．

６．２　検証事例

本稿では，イーサネットの検証を行った．

(1)仕様記述

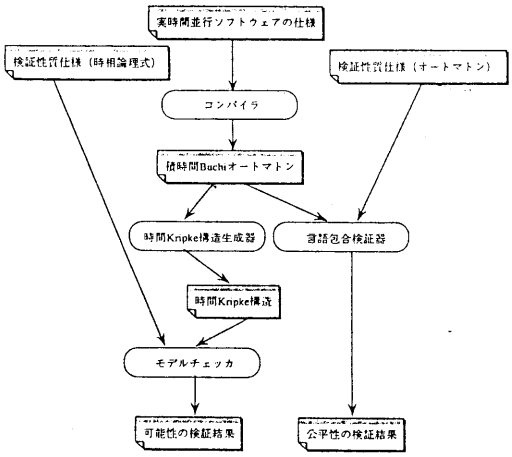


図４　検証システムの構成
Fig.4　Configuration of verification system

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５の時間オートマトンの積により仕様記述できる．

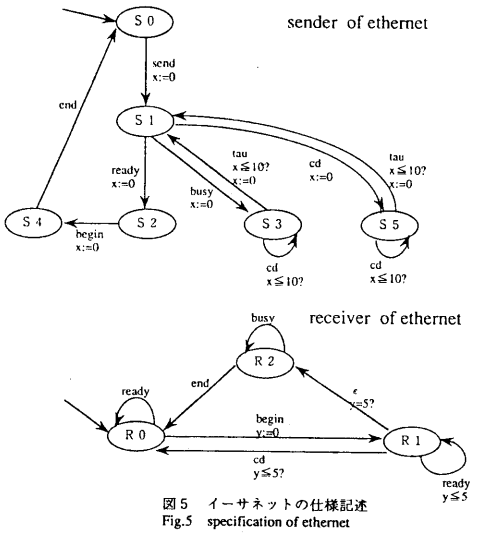


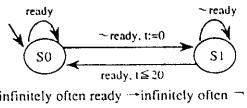
図５　イーサネットの仕様記述
Fig.5　specification of ethernet

検証性質記述は公平性を決定性時間Mullerオートマトンで記述して，可能性をリアルタイム時相論理で記述した．検証性質記述を図６に示す．

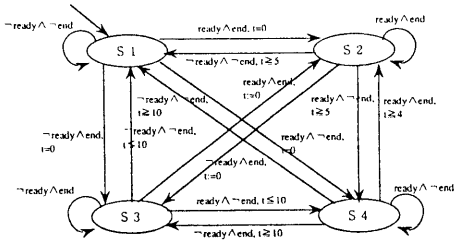
(2)検証

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テキスト形式であり，図７に例を示す．

検証コストは以下のように考えられる．言語包合アルゴリズムは深さ優先探索とDBMによる到達可能解析の融合であり，計算時間はO（オートマトンのノード数＋エッジ数）＋O（受理ループの状態数×（DBMの大きさ）^{3}）である．メモリ量はO（オート



(a)infinitely often ready →infinitely often ¬ready



(b)infinitely often ready →infinitely often end

ここで
~:補集合記号

(1)公平性の検証性質仕様
(1)Verification property specification about fairness

A G send→E F ≧5 end　　E F cd→E F ≦10 end

(a)可能性（存在記号）　(b)可能性（全称記号）
(a)probability(existence)　(b)probability(universality)

(2)可能性の検証性質仕様
(2)Verification property specification about possibility

図6　検証性質仕様
Fig.6　verification property specification

```
System specification ;
System configuration ;
  system=sender×receiver ;
Process specification(sender) ;
State definition part  S0, S1, ···, S5 ;
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, ···, R3 ;
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→cd, y≦5→R0 ;
     :
     :
     :
end ;
```

図7　コンパイラへの入力データ例
Fig.7　Example of input data of compiler

マトンのノード数＋エッジ数）＋0（受理ループの状態数（DBMの大きさ）[2]）である．また，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0（論理式の長さ×（時間Kripke構造のノード数＋エッジ数））＋0（ラベリングの状態数×（DBMの大きさ）[3]）である．メモリ量は0（時間Kripke構造のノード数＋エッジ数）＋0（ラベリングの状態数×（DBMの大きさ）[2]）である．各々の状態数と状態遷移数は百程度であり，実用的なコストで検証できた．

7．まとめ

本稿では，実時間分散ソフトウェアを対象として，時間オートマトンによる仕様記述手法を採用して，モデルチェッキングと言語包合アルゴリズムによる検証を同時に支援する検証システムを開発した．このシステムにより，実時間分散ソフトウェアの可能性と公平性が同時に検証可能となった．また，状態空間の組み合わせ爆発を回避するために，時間不等式手法を基礎として実時間並行ソフトウェアを検証した．数百の状態数を持つ小規模の通信システムの実時間タイミング検証は実用的な検証コストで可能であることがわかった[14]．大規模システムを検証するために，メモリ量の削減や状態空間の探索時間の削減が必要である．これらを解決するために，現在，我々は状態遷移関係をBDD表現する実時間シンボリック検証技術やヒューリスティックス検証技術を開発中である．

参考文献

1)Kavi K.M.:Real-time Systems,Abstraction,Language,and Design Methodologies,IEEE Computer Society(1992)

2)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)

3)Alur R. Dill D.:"The theory of Timed Automata",LNCS 600,pp.45-73(1992)

4)Alur R. Henzinger T.A.:"Logics and Models of Real Time:A Survey",LNCS 600,pp.74-106(1992)

5)Alpern B. Schneider F. B. :"Verifying temporal properties without temporal logic",ACM Trans. on Programming Languages and Systems,Vol.11,N0.1,pp.147-167 (1989)

6)Dill D.:"Timing assumptions and verification of finite-state concurrent systems",LNCS 407,pp.197-212(1989)

7)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)

8)Emerson E.A. LEI C. :"Modalities for model checking:brancing time logic strikes back",Science of computer programming 8,pp.275-306(1987)

9)KurshanR.P.:Computer-aided verification of coordinating processes:The automata-Theoretic Approach,Princeton University Press(1994)

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)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)

12)Tarjan R."Depth-first search and linear graph algorithms", SIAM Journal of Computing 1(2),pp.146-160(1972)

13)IEEE ANSI/IEEE 802.3, ISO/DIS 8802/3. IEEE Computer Society Press(1985)

14)Yamane S.:"Formal timing verification techniques for distributed system", Proc 5th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, IEEE Computer Society(1995.8)(to appeare)