実時間並行ソフトウェアの仕様記述と検証

山根 智†

実時間並行ソフトウェアでは，複雑な並行プロセス記述や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．そこで，我々は時間ステートチャートによる仕様記述と検証の手法について報告した[2]．本手法では，言語包合アルゴリズムにより公平性や正則性の検証を実現したが，可能性の検証はできない．本稿では，時間ステートチャートの可能性や公平性などの検証を考える．可能性を検証するために，以下の手法を提案する．

(1) 時間ステートチャートから意味的に等価な時間 Kripke 構造の生成手法を提案する．

(2) 状態空間の組合せ爆発を回避するために，時間不等式手法を基礎としたモデルチェッキング手法を提案する．

以上の手法により，時間ステートチャートの可能性や公平性などを検証できる検証システムを開発して，本手法の有効性を示した．

Specification and Verification of Real-time Concurrent Software

SATOSHI YAMANE†

It is necessary to specify and verify concurrent processes including timing constraints in real-time concurrent software. For this reason, we have reported the specification and verification method by timed statechart. Using this method, we have verified the properties of fairness and regularity by language inclusion algorithm, but can not verfiy the property of possibilities. In this paper, we propose the verification of both fairness and possibilities using model checking algorithm and language inclusion algorithm. The main feature is as follows.

(1) We generate timed Kripke structure from timed statechart. Timed Kripke structure is semantically equal to timed statechart.

(2) In order to avoid the state explosion problem, model checking algorithm is based on inequalities method.

We have developed the verification system based on the proposed method, and have shown it effective.

1. はじめに

航空機や通信機器などのような実時間並行ソフトウェアは，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，並行プロセス記述や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，実時間並行ソフトウェアを対象として議論する．従来より，並行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．

そこで，我々は時間ステートチャートによる仕様記述と言語包合アルゴリズムによる検証の手法について報告した[2]．一般的に，実時間システムの検証方式としては，モデルチェッキングや言語包合アルゴリズムなどがある[3]．

(1) モデルチェッキングはイベントの存在しない時間グラフ[4]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．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) 言語包合アルゴリズムは仕様の時間オートマトン[5]の受理言語が検証仕様の時間オートマトンの受理言語に包合されるかどうかを判定するも

† 島根大学理学部情報科学科
Department of Computer Science, Faculty of Science, Shimane University

のである．言語包含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．

ゆえに，時間ステートチャートによる手法[2]では，公平性や正則性の検証はできるが，可能性の検証はできない．

そこで，本稿では，時間ステートチャートのモデルチェッキング手法を提案して，可能性や公平性などを検証可能とする．従来の実時間システムのモデルチェッキング手法では，イベントの存在しない時間グラフが検証の対象であったり，時間グラフから大規模なリージョングラフを生成するものが提案されていた[4]．本稿では，時間ステートチャートを効率よく検証するために，以下の手法を提案する．

(1) 時間ステートチャートから意味的に等価な時間 Kripke 構造の生成手法を提案する．これにより，時間ステートチャートのモデルチェッキングが可能となる．

(2) 状態空間の組合せ爆発を回避するために，時間不等式手法[6),7)] を基礎としたモデルチェッキング手法を提案する．

以上の手法を支援する検証システムを実現して，提案する手法の有効性を示す．

以下の本稿は，2 章では状態遷移モデルに基づく並行ソフトウェアの仕様記述の基本概念を述べ，3 章では仕様記述手法を述べ，4 章では検証手法の概要を提案する．5 章では可能性の検証手法を提案して，6 章では公平性の検証手法を述べる．7 章では検証システムとその有効性を示す．最後に，8 章ではまとめと今後の課題を示す．

2. 状態遷移モデルに基づく実時間並行ソフトウェアの仕様記述の基本概念

2.1 実時間並行ソフトウェアの特徴

実時間並行ソフトウェアはデータ駆動型システムと異なり，以下の特徴を有する．

(1) 実行の無限性
入力が無限に続き，動作が停止することなく有限の内部状態が無限に繰り返される．

(2) 動作の並行性
外界よりリアルタイムなイベント列が入力されるので，並行動作を考慮してシステムを構成する必要がある．

(3) システムの複雑性
データの入出力関係だけではシステム構成が決まらず，内部状態が複雑（並行的かつ階層的に）に絡みあう．

(4) 時間制約の厳密性
システムは 2 秒以内に応答せよといったタイミング制約や時間遅延の正しさがシステムの正当性に大きく依存する．

2.2 ステートチャート

従来のオートマトンは 1 つのプロセスの状態遷移のみが記述可能であり，並行プロセスや複雑な状態遷移は記述できなかった．ステートチャートは従来のオートマトンを拡張して，並行プロセスや階層プロセスを記述可能にして，複雑な状態遷移を簡易に記述できるようにした[8]．その主要な特徴は以下のとおりである．

(1) オートマトンの階層性
階層化は状態をクラスタ化して，状態と状態集合を閉じた等高線により記述する．状態の等高線の境界で作られる遷移はすべての状態集合に適用される．図 1 (a) の状態図は，階層化により (b) のようなステートチャートで記述できる．階層化 (b) は，A と C を D に抽象化して，D の境界から 2 つのイベント b を 1 つのイベント b で置換する．A と C の結合 D は exclusive-OR であり，D に存在することは A または C に存在することである．

(2) オートマトンの並行性
並行化はサブ状態のすべてからなり，サブ状態のカルテジアン積として表現される．図 2 (a)

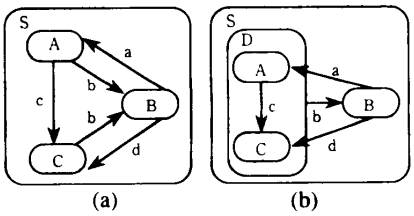


図 1 オートマトンの階層化
Fig. 1 Hierarchy of automaton.

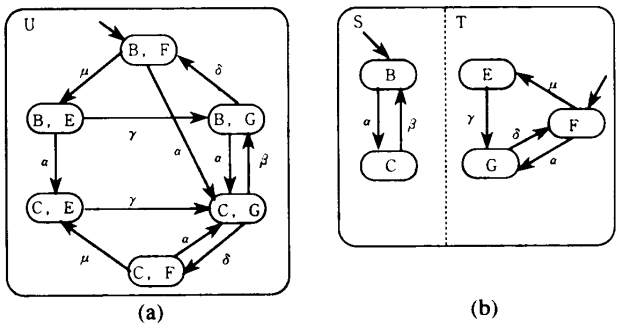


図 2 オートマトンの並行化
Fig. 2 Concurrency of automata.

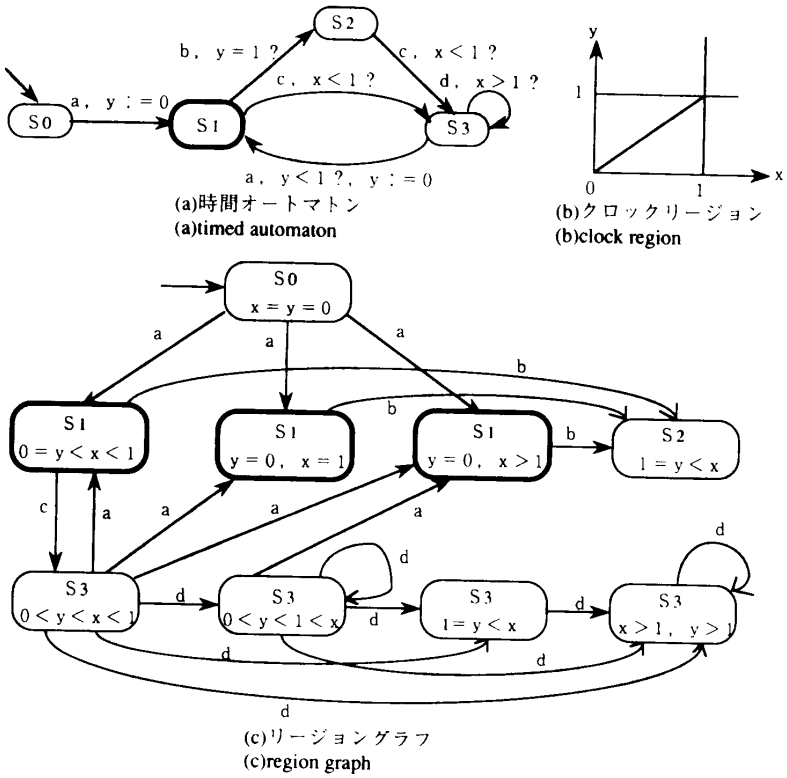


図3 時間オートマトンの例
Fig. 3 Example of timed automaton.

の状態図は，並行化により (b) のようなステートチャートで記述できる．並行化 (b) は，S と T の両方がアクティブに並行動作する．状態集合を $\{S,T\}$ で表現すると，初期状態は $\{B,F\}$ であり，イベント α により $\{C,G\}$ に遷移する．このような並行化により，直交状態が記述できて，状態の膨張現象がある程度解決でき独立な並行プロセスが効率よく記述できる．

(3) 通信のブロードキャスト性
図 2 (b) の S と T の並行動作は，イベントが S と T に同時に通信されることにより実現される．これがブロードキャスト通信の概念であり，外部イベントの生起はすべての直交状態の遷移を引き起こす．

2.3 時間オートマトン

時間オートマトン[5]は実時間システムを仕様記述するために，実時間のタイミング制約により ω-オートマトンを拡張したものである．簡単な時間オートマトンの例を**図 3** (a) に示す．有向直線のラベルにはイベントやリセット式，タイミング制約式がある．$S0 \rightarrow S1$ の遷移はイベント a により生起してクロック変数 y がリセットされる．また，$S1 \rightarrow S2$ の遷移はイベント b の生起かつクロック変数 $y = 1$ の条件を満たすとき起きる．受理状態集合は $\{S1\}$ である．しかし，時間オートマトンを検証する場合は，リージョングラフを生成する必要がある．図 3 (a) のオートマトンの場合，クロック変数が x, y で時間定数が各々1 なので，クロック変数の取りうる組合せは，図 3 (b) のクロックリージョンで表される．ゆえに，このオートマトンの取りうるリージョングラフは図 3 (c) になる．しかし，リージョングラフの生成と探索により，検証コストはタイミング制約の長さの指数オーダのメモリ量と計算時間がかかる[5]．

3. 実時間並行ソフトウェアの仕様記述手法

本章では，文献 2) と同様な時間ステートチャートとその展開アルゴリズムを簡単に説明する．

3.1 時間ステートチャート

仕様記述手法としては，実時間記述でステートチャートを拡張した文献 2) の時間ステートチャートを採用する．図形表現はステートチャートの表記を採用して，テキストの構文は以下のように定義する．まず，従来のステートチャートを簡単に定義した後に，時間ステートチャートを定義する．

[定義 1] (ステートチャート)[9]
ステートチャートは $(\Sigma, S, s0, 0, H, E, \rho, \psi)$ の 8 つ組

で定義する．ここで，

Σ: 有限入力イベント集合

S: 有限状態集合

$s0 \subseteq S$: 初期状態集合

0: 有限出力イベント集合

H: ヒストリエントランス（同一レベルの状態集合の中で最も最近実行された状態）

E: $2^S \times \Sigma \times 0 \to 2^{S \cup H}$

ρ: $S \to 2^S$ は階層関数であり，各状態のサブ状態を決定する．

ψ: $S \to \{\mathrm{AND}, \mathrm{OR}\}$ は型関数であり，各状態の型を決定する． □

時間ステートチャートは，ステートチャートの出力イベントやヒストリエントランスを隠蔽して，実時間のタイミング制約記述で拡張したものである．これにより，形式言語理論とオートマトン理論のうえで時間ステートチャートが形式化できる．

[定義 2] （時間ステートチャート）[2]

時間ステートチャートは $(\Sigma, S, s0, C, E, F, \rho, \psi)$ の 8 つ組で定義する．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: $2^S \times \Sigma \times 2^C \times \Phi(C) \to 2^S$: 状態遷移関数

$F \subseteq S$: 受理状態集合

ρ: $S \to 2^S$ は階層関数であり，各状態のサブ状態を決定する．

ψ: $S \to \{\mathrm{AND}, \mathrm{OR}\}$ は型関数であり，各状態の型を決定する．

なお，遷移関数は通常のオートマトンと異なり，並行動作により状態のベキ集合からベキ集合への写像となる．ここで，$\Phi(C)$ 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 と整数の時刻定数 d により再帰的に定義される．

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

状態遷移のノード $(2^S, 2^{S'}, a, \lambda, \delta)$ はイベント $a \in \Sigma$ による状態 2^S から状態 $2^{S'}$ への遷移を表す．$\lambda \subseteq C$ は状態遷移にともなってリセットされるクロックを与える．もし，走査列 r の無限な状態集合が

$\inf(r) \cap F \neq 0$ ならば，走査列 r は受理列である．$\psi(S) = \mathrm{AND}$ ならば $\rho(S)$ は S の AND 分解（並行）であり，$\psi(S) = \mathrm{OR}$ ならば $\rho(S)$ は S の OR 分解（階層）である． □

ここで，時間ステートチャートの動作イメージを簡単に説明する．まず，イベント e が状態 S に到着すると，次の (1) と (2) の規則が状態階層ごとに再帰的に適用されて，状態遷移が決定される．

(1) $\psi(S) = \mathrm{AND}$ ならば，S のすべてのサブ状態 $\rho(S)$ にイベント e がブロードキャスト通信される．

(2) $\psi(S) = \mathrm{OR}$ ならば，どれか 1 つのサブ状態 $\rho(S)$ にイベント e が通信される．

3.2 積時間 Buchi オートマトンの生成

本稿では，時間ステートチャートを検証するために，文献 2) とほぼ同様な手法により，時間ステートチャートから積時間 Buchi オートマトンに変換する．時間ステートチャートは，唯一のルート状態を持つ AND/OR ツリーである．積時間 Buchi オートマトンを生成するためには，ルート状態から各階層ごとに，型関数を適用して，$\psi(S) = \mathrm{AND}$ ならば $\rho(S)$ の積を生成し，$\psi(S) = \mathrm{OR}$ ならば $\rho(S)$ の階層を展開する．以上の操作を階層の最下部まで繰り返す．以下では，時間 Buchi オートマトンと積時間 Buchi オートマトンを定義してから，時間ステートチャートの展開アルゴリズムを定義する．

[定義 3] （時間 Buchi オートマトン）[5]

時間 Buchi オートマトンは $A = (\Sigma, S, s0, C, E, F)$ 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

$E : S \times \Sigma \times 2^C \times \Phi(C) \to S$: 時間付遷移関数

$F \subseteq S$: 受理状態集合

なお，$\Phi(C)$ はクロック C のタイミング制約式 δ であり，クロック変数 X と整数の時刻定数 d により再帰的に定義される．

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

状態遷移のノード $(s, s', a, \lambda, \delta)$ はイベント $a \in \Sigma$ による状態 $s \in S$ から状態 $s' \in S$ への遷移を表す．$\lambda \subseteq C$ は状態遷移にともなってリセットされるクロックを与える．以下では，状態遷移を

$$s \xrightarrow{a,\lambda,\delta} s'$$

と記述する．所与のイベントにより，無限回遷移する状態の集合と受理状態集合 F との共通集合が空集合でないならば，そのイベントは受理される． □

[定義 4] （積時間 Buchi オートマトン）

時間 Buchi オートマトン $A_1 = (\Sigma_1, S_1, s_1 0, C_1, E_1, F_1)$ と $A_2 = (\Sigma_2, S_2, s_2 0, C_2, E_2, F_2)$ に対して，積時間 Buchi オートマトン A を定義する．

$A = (\Sigma, S, s0, C, E, F)$

ここで,

Σ: $\Sigma_1 \cup \Sigma_2$

S: $S_1 \times S_2$

$s0$: $s_1 0 \times s_2 0$

C: $C_1 \cup C_2$

E: $(S_1 \times S_2) \times (\Sigma_1 \cup \Sigma_2) \times 2^{C1 \cup C2} \times \Phi(C_1 \cup C_2) \rightarrow (S_1 \times S_2)$

F: $F_1 \times F_2$

すなわち，A_1 と A_2 の状態遷移を各々

$$s_1 \xrightarrow{a1,\lambda 1,\delta 1} s_1', \quad s_2 \xrightarrow{a2,\lambda 2,\delta 2} s_2'$$

とすると，$a1 = a2$ のときの A の状態遷移は

$$s_1 \times s_2 \xrightarrow{a1,\lambda 1 \cup \lambda 2,\delta 1 \wedge \delta 2} s_1' \times s_2'$$

であり，$a1 \neq a2$ のときの A の状態遷移は

$$s_1 \times s_2 \xrightarrow{a1,\lambda 1,\delta 1} s_1' \times s_2, \quad s_1 \times s_2 \xrightarrow{a2,\lambda 2,\delta 2} s_1 \times s_2'$$

である．3個以上の時間 Buchi オートマトンの積も同様に定義できる．また，時間 Buchi オートマトンの積演算は積演算閉包性により保証されている[5]．□

[定義5] (時間ステートチャートの展開アルゴリズム)

```
program statechart;
    var level:integer;      状態の深さレベル
    var counter:integer; 同一レベルの状態通し番号
procedure andor(level:integer,var counter:integer);
  begin
    if (ψ(S) = AND)
      then
        for ρ(S) の全イベントで繰り返す do
          begin
            該当イベントで遷移する状態の積を作る;
            同時にタイミング制約の論理積を作る;
            各状態に (level+1,counter) をラベル付け
                する;
            counter:=counter+1;
          end;
    if (ψ(S) = OR)
      then
        for ρ(S) の全サブ状態について繰り返す do
          begin
            S からサブ状態 ρ(S) に展開する;
            各状態に (level+1,counter) をラベル付け
                する;
            counter:=counter+1;
          end;
  end;
begin
  level:=1;
  for   ルート状態から最下層状態まで繰り返す do   begin
      counter:=1;
      for   同一レベルの全状態を繰り返す do
          begin
            andor(level:integer,var counter:integer);
          end;
      level:= level+1;
  end;
end.                                                    □
```

4. 検証手法の概要

本稿では，文献2)と同様に，時間ステートチャートから生成した積時間 Buchi オートマトンを検証の対象とする．可能性と公平性の概念を検証するために，検証手法は以下のようにモデルチェッキングと言語包合アルゴリズムから構成する．

(1) 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．

(2) 公平性の検証ときは，文献2)とほぼ同様な手法により，言語包合アルゴリズムで実現する．検証性質仕様は決定性時間 Muller オートマトンで定義して，積時間 Buchi オートマトンの受理言語が決定性時間 Muller オートマトンの受理言語に包合されるかどうかを判定する．

以下の5章と6章で各々の検証手法を説明する．

5. 可能性の検証手法

本章では，本稿の中で最も重要な積時間 Buchi オートマトンのモデルチェッキング手法を提案する．

5.1 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 TCTL (Timed CTL)[4]を考える．

[定義6] (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．

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

ここで,

$ap \in AP$ (原始命題)

$c \in N$ (自然数)

$\sim$ は二項関係 $<$, $\leq$, $=$, $\geq$, $>$

また，(1) $\exists\Diamond_{\sim c}\phi = \exists$ (true $U_{\sim c}\phi$) や (2) $\forall\Diamond_{\sim c}\phi = \forall$ (true $U_{\sim c}\phi$), (3) $\exists\Box_{\sim c}\phi = \neg\forall\Diamond_{\sim c}\neg\phi$, (4) $\forall\Box_{\sim c}\phi = \neg\exists\Diamond_{\sim c}\neg\phi$ といった時相表現も可能である．ここで,

$\Diamond$: ある性質がいつかは成り立つ.

$\Box$: ある過去の性質がつねに成り立つ.

∃: ある計算パスで成り立つ.

∀: すべての計算パスで成り立つ.

具体的には, $\exists\phi_1 U_{<C}\phi_2$ は次のように解釈される: 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.

次に, TCTL の意味は時間 Kripke 構造上の充足関係として定義する. 時間 Kripke 構造は Kripke 構造[10]をタイミング制約式で拡張したものである.

[定義 7] (時間 Kripke 構造)

確率時間 Kripke 構造は $T=(S',P',R',\pi')$ により定義する. ここで,

S': 状態の有限集合

P': $S'\to 2^{AP}$ 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)

R': $S'\times S'$ は状態遷移関係

π': $S'\to 2^C\times\Phi(C)$ は各状態にクロック集合 C を割り付ける関数

クロック集合 C はタイミング制約式 δ とリセット式からなる. タイミング制約式 δ はクロック変数 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.

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

リセット式はクロック変数 X により定義する.

$$X:=0$$ □

[定義 8] (TCTL の意味)

時間 Kripke 構造 $T=(S',P',R',\pi')$ に対して, 初期状態 $s0'\in S'$ が TCTL 式 ϕ を充足することを以下のように表現する: $(s0',T)|=\phi$.

以下では, $s0'|=\phi$ と記述する.

(1) $s0'|=ap$ iff $ap\in P'(s0')$

(2) $s0'|=\phi_1\wedge\phi_2$ iff $s0'|=\phi_1$ かつ $s0'|=\phi_2$

(3) $s0'|=\neg\phi_1$ iff $s0'|\neq\phi_1$

(4) $s0'|=\exists\phi_1 U_{\sim C}\phi_2$ iff ある状態列 si' $(i=0,\cdots,n)$ に対して, 以下の (a) と (b) が満たされる.

(a) 初期状態 $s0'$ から状態 sn' に遷移する時間の経過 t が $\sim C$ を充足して,
$sn'|=\phi_2$

(b) 初期状態 $s0'$ から状態 si' $(0\le i<n)$ に遷移する時間の経過 t' が $0\le t'<t$ を充足して,
$si'|=\phi_1$ $(0\le i<n)$

(5) $s0'|=\forall\phi_1 U_{\sim C}\phi_2$ iff すべての状態列 si' $(i=0,\cdots,n)$ に対して, 以下の (a) と (b) が満たされる.

(a) 初期状態 $s0'$ から状態 sn' に遷移する時間の経過 t が $\sim C$ を充足して,
$sn'|=\phi_2$

(b) 初期状態 $s0'$ から状態 si' $(0\le i<n)$ に遷移する時間の経過 t' が $0\le t'<t$ を充足して,
$si'|=\phi_1$ $(0\le i<n)$ □

5.2 時間 Kripke 構造の生成

モデルチェッキングで検証するためには, 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 構造の生成は順序機械から Kripke 構造の生成手法[11]を拡張したものである. 時間 Kripke 構造の生成手法は積時間 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.

[定義 9] (時間 Kripke 構造の生成手法)

積時間 Buchi オートマトン $A=(\Sigma,S,s0,C,E,F)$ に対して, 時間 Kripke 構造 $T=(S',P',R',\pi')$ は以下のように構成できる.

(1) 状態の有限集合 S' は状態遷移関数 E に対応付ける.
$S':E$

(2) ラベリング関数 P' は状態遷移関数 E からイベント集合 Σ への写像に対応付ける.
P': $E\to\Sigma$

(3) 状態遷移関係 R' は状態遷移関数 E の間の関係に対応付ける.
R': $E\times E$

(4) 割付け関数 π' は状態遷移関数 E から各状態にクロック集合 C を割り付ける関数に対応付ける.
π': $E\to 2^C\times\Phi(C)$ □

次に, 時間 Kripke 構造と時間 Buchi オートマトンとの等価性を説明する.

[定理 1] (時間 Buchi オートマトンとの等価性)

時間 Kripke 構造 $T=(S',P',R',\pi')$ と積時間 Buchi オートマトン $A=(\Sigma,S,s0,C,E,F)$, 時相論理式 ϕ に対して,

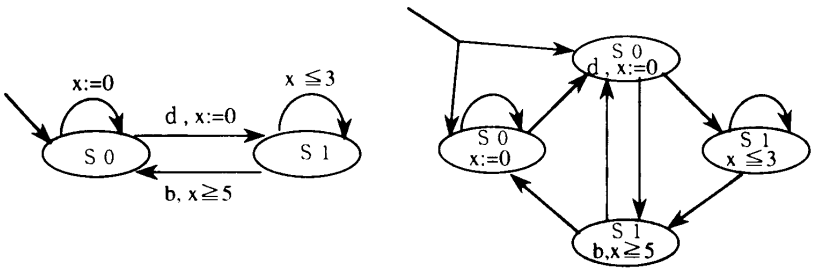
$$|=_A\phi \quad \text{iff} \quad |=_T\phi$$

が成り立つ.

(証明) 時間 Kripke 構造 T と積時間 Buchi オートマトン A の状態列の等価性を導出した後に, 充足の等価性を示す.

(1) 状態列の等価性

時間オートマトン A の状態遷移 $si\xrightarrow{ai,\lambda i,\delta i}si+1$ と時間 Kripke 構造 T の状態 si' が等しいと考える. これより, 時間 Kripke 構造 T と積時間



(1)時間オートマトン
(1)timed automaton

(2)時間Kripke構造
(2)timed Kripke structure

図4 時間 Kripke 構造の生成
Fig. 4 Generation method of timed Kripke structure.

Buchi オートマトン A の状態列は以下のように等価である.

$$s0 \xrightarrow{a0,\lambda 0,\delta 0} s1 \xrightarrow{a1,\lambda 1,\delta 1} s2 \xrightarrow{a2,\lambda 2,\delta 2} \cdots$$

$$s0' \rightarrow s1' \rightarrow s2' \rightarrow \cdots$$

(2) 充足の等価性
$si \xrightarrow{ai,\lambda i,\delta i} si+1$ と si' が等価であり，$\Sigma \ni \phi$ ならば以下が成り立つ.

$$si' \models_A \phi \quad \text{iff} \quad si+1 \models_T \phi$$

ϕ に関する帰納法により，すべての状態遷移列に対して，

$$\models_A \phi \quad \text{iff} \quad \models_T \phi$$

が容易に示せる. □

[例 1] (時間 Kripke 構造の構成例)
図 4 の (1) の時間オートマトンを時間 Kripke 構造に変換したものが (2) である. 時間 Kripke 構造は有向グラフで各状態に原始命題とタイミング制約式，リセット式を割り付けたものである. 時間オートマトンを時間 Kripke 構造に変換すると，最悪の場合は状態数がイベント数の 2 乗のオーダで増加する. □

5.3 リアルタイムモデルチェッキング

5.3.1 モデルチェッキングの概要

リアルタイムモデルチェッキングでは，時間 Kripke 構造 $T=(S',P',R',\pi')$ が TCTL 形式 ψ を充足するかどうかをラベリングアルゴリズム[10]を拡張して判定する. また，リージョングラフ[4]を生成しないで，実時間制約をクロック変数の不等式形式 DBM (Difference Bounds Matrices)[6]で保持して，状態空間の組合せ爆発を抑制する.

リアルタイムモデルチェッキングは以下の処理から構成する.

(1) TCTL 形式 ψ の時間制約を考慮しないラベリング
(2) ラベリング状態列の実時間到達可能性の判定
(状態遷移 $si \xrightarrow{ai,\lambda i,\delta i} si+1$ の可能性)
(3) TCTL 形式 ψ のタイミング制約の判定
(たとえば，$\phi_1 U_{\sim C} \phi_2$ の $\sim C$)

以上の (1)〜(3) が充足されれば，TCTL 形式 ψ で時間 Kripke 構造 T をラベリングする.

以下では，(1)〜(3) を各々説明して，最後に (1)〜(3) をまとめたリアルタイムモデルチェッキングアルゴリズムを説明する.

5.3.2 TCTL 形式 ψ の時間制約を考慮しないラベリング

Clarke らの CTL のラベリングアルゴリズム[10]と同様に，状態列と TCTL 形式の時間制約を考慮しないで，第 1 段階では TCTL 形式 ψ の長さ 1 の部分論理式をラベリングし，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，最後に，時相論理式 ψ の長さの論理式をラベリングして終了する. ただし，以下の 5.3.3 項と 5.3.4 項のラベリング状態列と TCTL 形式の時間制約が充足されなければ，ラベリングしない.

5.3.3 ラベリング状態列の実時間到達可能性の判定

ラベリング状態列にはタイミング制約式により実時間制約が存在する. このために，ラベリング状態列がタイミング制約を充足するかどうかを到達可能解析により判定する. 時間不等式手法による到達可能解析は，以下のように定義できる.

[定義 10] (時間不等式手法による到達可能解析)
到達可能解析は以下の処理から構成する.

(1) 各状態ごとに成立するクロック制約式により，DBM (Difference Bounds Matrices)[6]を生成する.
(2) Floyd-Warshall のアルゴリズムにより，各 DBM の正規形 DBM[6]を求める.
(3) 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，すなわち状態遷移の発生可能性をチェックする. □

以下では，(1)〜(3) を説明する. まず，DBM を

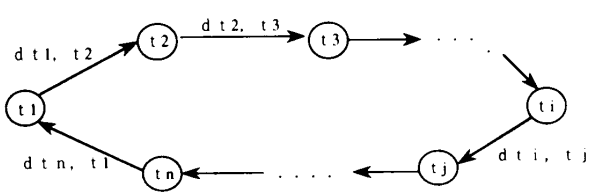


図5 DBMのグラフ表現
Fig. 5 The graph representation of DBM.

定義する.

[定義11] (DBM (Difference Bounds Matrices))
DBMはクロック制約割り当てのマトリックスであり,各状態ごとに作成する.クロック制約割り当ての形式は以下のとおりである.

$$t_i - t_j \leq d_{ij}$$

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.d_{ij} は集合 $\{\cdots, -2, -1, 0, 1, 2, \cdots\} \cup \{\cdots, -2^-, -1^-, 0^-, 1^-, 2^-, \cdots\} \cup \{-\infty, \infty\}$ の要素である.たとえば,2^- は $1 < 2^- < 2$ を充足する限りなく2に近い数である.これにより,$t_i - t_j < d_{ij}$ は $t_i - t_j \leq d_{ij}^-$ と表現する.なお,仮想クロック $t_0 = 0$ と定義する. □

[定義12] (正規形DBM)
DBMの各クロック変数をノードと見なして,各クロック定数をエッジのコストと見なすことにより,DBMがグラフ表現できる.**図5**に,クロック変数 $t1, t2, \cdots, tn$ とクロック定数 $d_{t1,t2}, d_{t2,t3}, \cdots, d_{tn,t1}$ が与えられたときのグラフ表現の例を示す.ただし,$ti - tj \leq d_{ti,tj}$ とする.これにより,Floyd-Warshallのアルゴリズムにより冗長のない最短パスである正規形DBM[6]を求めることができる. □

最後に,DBMを用いたクロック制約の到達可能性の判定法を定義する.

[定義13] (クロック制約の到達関係の判定)
時間Kripke構造では状態遷移元と状態遷移先の時間領域に交わりが存在しないと状態遷移が不可能である.そこで,状態遷移元と状態遷移先の正規形DBMのintersectionを生成して,それらの時間領域に交わりが存在するかどうかをチェックする.

(1) 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DBMのintersection[7]を生成する.

$$\text{intersection} \quad \text{DBM} = \min\{d_{ij}, d'_{ij}\}$$

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

なお,DBMがintersection演算で閉じていることは保証されている.

(2) intersection DBMの正規形の閉ループに負のコストが存在すれば到達関係を充足しない.すなわち,そのDBMをintersectionした状態間には状態遷移が存在しない.もし,負のコストが存在しなければ到達関係を充足する.すなわち,そのDBMをintersectionした状態間には状態遷移が存在する. □

次に,intersection DBMの正規形の閉ループに負のコストが存在すれば到達可能性を充足しないことを証明する.

[定理2] (到達関係の充足性)[6]
intersection DBMの正規形の閉ループに負のコストが存在すれば到達可能性を充足しない.

(証明) クロック変数 $t1, t2, \cdots, tn$ が与えられたとする.閉ループのコスト D は $d_{t1,t2} + d_{t2,t3} + \cdots + d_{tn,t1}$ で定義できる.ここで,$d_{t1,t2}$ や $d_{t2,t3}, \cdots, d_{tn,t1}$ は以下のように定義できる.

$$t1 - t2 \leq d_{t1,t2}$$
$$t2 - t3 \leq d_{t2,t3}$$
$$\vdots$$
$$tn - t1 \leq d_{tn,t1}$$

ゆえに,

$$D = (t1 - t2) + (t2 - t3) + \cdots + (tn - t1)$$
$$= t1 - t1$$

D が負 $(D = t1 - t1 < 0)$ ならば矛盾するので,到達関係を充足しないことが証明できる. □

5.3.4 TCTL形式 ψ のタイミング制約の判定

TCTL形式 ψ のタイミング制約の充足性の判定は,ある状態から別の状態に遷移する時間の経過が論理式のタイミング制約を充足するかどうかをチェックすることである.たとえば,論理式 $\exists\phi_1 U_{\sim C}\phi_2$ のタイミング制約の充足性の判定の場合では,ある状態列 si' $(i = 0, \cdots, n)$ に対して,

(a) 状態 $s0'$ から状態 sn' に遷移する時間の経過 t が $\sim C$ を充足して,

$$sn' \models \phi_2$$

(b) 状態 $s0'$ から状態 si' $(0 \leq i < n)$ に遷移する時間の経過 t' が $0 \leq t' < t$ を充足して,

$$si' \models \phi_1 \quad (0 \leq i < n)$$

である.状態間の時間の経過はDBMの中のあるクロック変数に着目すれば容易に計算できる.

[定義14] (状態間の時間の経過の計算方法)
一般性を失うことなく,状態列 si' $(i = j, \cdots, k)$ に

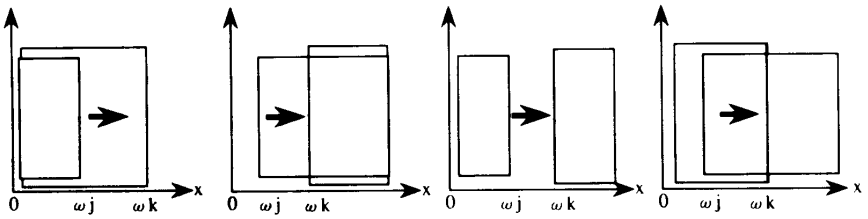


図6 状態遷移の時間の経過
Fig. 6 The time elapsed in state transitions.

対して，状態 sj' から状態 sk' に遷移する時間の経過の計算方法を定義する．

(1) あるクロック変数 x が状態遷移間でリセットされないとき
sj' では $x \leq \omega j$ または $x \geq \omega j$ の形式であり，sk' では $x \leq \omega k$ または $x \geq \omega k$ の形式である（ただし，$j < k, \omega j < \omega k, \omega j$: クロック定数）．**図6**に示すように，以下の(a)～(d)の場合に分類して時間の経過 t を計算する．
(a) $x \leq \omega j$ かつ $x \leq \omega k$ のとき
$t \leq \omega k - \omega j$
(b) $x \geq \omega j$ かつ $x \geq \omega k$ のとき
$t \geq \omega k - \omega j$
(c) $x \leq \omega j$ かつ $x \geq \omega k$ のとき
$t \geq \omega k$
(d) $x \geq \omega j$ かつ $x \leq \omega k$ のとき
$t \leq \omega k - \omega j$

(2) あるクロック変数 x が状態遷移間でリセットされるとき
リセットが状態 sl' $(j < l < k)$ で発生したとする．sj' から sl' の経過時間と sl' から sk' の経過時間を(1)の方法で計算する．そして，(1)の方法と同様に不等号の向きを考慮しながら，2つの経過時間の和を計算する．リセットが複数ある場合も同様に計算できる．

以上の(1)と(2)により状態 sj' から状態 sk' に遷移する時間の経過が計算できる．また，$x < \omega j$ といった等号のない不等式の場合も上記と同様に計算できる． □

5.3.5 リアルタイムモデルチェッキングアルゴリズム

最後に，5.3.2～5.3.4項を組み合わせた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15]（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 $T = (S', P', R', \pi')$ がTCTL形式 ψ を充足するかどうかを判定する．論理式の実時間制約の充足性やDBMの到達可能性を判定しながら，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 ψ 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 ψ でラベリングされたとき，

$(s0', T)| = \psi$

と判定する．以下では，TCTL形式 ψ の各部分論理式 ϕ に対するモデルチェッキングアルゴリズムを定義する．

(1) $\phi = ap$ のとき
$ap \in P'(si')$ ならば，ϕ で si' がラベリングされている．

(2) $\phi = \phi_1 \wedge \phi_2$ のとき
$\phi_1 \in P'(si')$ かつ $\phi_2 \in P'(si')$ ならば，ϕ で si' をラベリングする．

(3) $\phi = \neg\phi_1$ のとき
$\phi_1 \notin P'(si')$ ならば $\neg\phi_1$ で si' をラベリングする．

(4) $\phi = \exists\phi_1 U_{\sim c}\phi_2$ のとき
ある状態列 si' $(i = 0, \cdots, n)$ に対して，以下の(a)～(d)が成り立つならば ϕ で si' $(i = 0, \cdots, n)$ をラベリングする．
(a) si' $(0 \leq i < n)$ に対して ϕ_1 がラベリングされている．
(b) sn' に対して ϕ_2 がラベリングされている．
(c) $s0'$ から sn' への状態遷移の時間の経過が $\sim c$ を充足する．
(d) $s0'$ から sn' への状態遷移がDBMの到達可能性を充足する．

(5) $\phi = \forall\phi_1 U_{\sim c}\phi_2$ のとき
すべての状態列 si' $(i = 0, \cdots, n)$ に対して，以下の(a)～(d)が成り立つならば ϕ で si' $(i = 0, \cdots, n)$ をラベリングする．

(a) si' $(0 \leq i < n)$ に対して ϕ_1 がラベリングされている.

(b) sn' に対して ϕ_2 がラベリングされている.

(c) $s0'$ から sn' への状態遷移の時間の経過が $\sim c$ を充足する.

(d) $s0'$ から sn' への状態遷移がDBMの到達可能性を充足する. □

次に，リアルタイムモデルチェッキングの計算コストを説明する.

[定理3] (リアルタイムモデルチェッキングの計算コスト)

$0[|\psi| \times (|S'| + |R'|)] + 0(|S'| \cdot n^3)$

ここで，n: クロック変数の数.

(証明の概要)

(1) $0[|\psi| \times (|S'| + |R'|)]$

CTLのモデルチェッキングの計算コスト[10]より自明である.

(2) $0(|S'| \cdot n^3)$

Floyd-Warshallのアルゴリズムの計算コスト[12]より自明である. □

次に，リアルタイムモデルチェッキングの例を示す.

[例2] (リアルタイムモデルチェッキングの事例)

TCTL形式 $\forall pU_{<10}q$ のラベリングアルゴリズムにより，**図7**(1)の時間Kripke構造が生成できたとする.

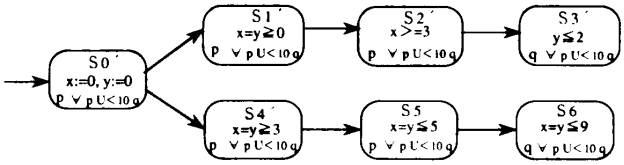
まず，ラベリング状態列のタイミング制約を計算する必要がある．ラベリングするときに，不等式を生成してDBMを作成する．たとえば，初期状態 $s0'$ のタイミング制約は $x = y = 0$ であるので,

$x = y$より, $x - y \leq 0$ かつ $y - x \leq 0$

$x = 0$ より, $x - t0 \leq 0$ かつ $t0 - x \leq 0$

$y = 0$ より, $y - t0 \leq 0$ かつ $t0 - y \leq 0$

の不等式ができる．以上より，初期状態 $s0'$ のDBMのD1が得られる．そして，個々のDBMの正規形をFloyd-Warshallのアルゴリズムに計算する．次に，DBMによる到達可能解析を行う．図7(2)に示すように，正規形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 $s2' \rightarrow s3'$ はタイミング制約を充足しないことが分かる．なお，$s0'$ から $s3'$ の状態遷移の時間の経過はクロック変数yに着目すると，D4の要素 $y - t0 \leq 2$ より $y \leq 2$ であり，$U_{<10}$ を充足する．以上より，TCTL形式 $\forall pU_{<10}q$ 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

*00	000
DBM D1=0*0	正規形 DBM=000
00*	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7 リアルタイムモデルチェッキングの例
Fig. 7 Example of real-time model checking.

6. 公平性の検証手法

公平性の検証手法は，文献2)とほぼ同様であるので，簡単に説明する.

6.1 検証性質仕様記述言語

検証性質記述の時間オートマトン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[5]を考える.

[定義16] (検証性質記述言語)

決定性時間Mullerオートマトンは $(\Sigma, S, s0, C, E, F)$ 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: $S \times \Sigma \times 2^C \times \phi(C) \rightarrow S$: 状態遷移関数

$F \subseteq 2^S$: 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，$\Phi(C)$ 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：

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

状態遷移のノード $(s, s', a, \lambda, \delta)$ はイベント $a \in \Sigma$ による状態 $s \in S$ から状態 $s' \in S$ への遷移を表す．$\lambda \subseteq C$ は状態遷移にともなってリセットされるクロックを与える．もし，走査列 r の無限な状態集合 $\inf(r)$ が $\inf(r) \in F$ 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 17] （公平性の定義）[13]

公平性は形式的に以下のように定義できる．

$$SF(L,U) = \{\inf(r) | \inf(r) \cap L \neq \phi \rightarrow \inf(r) \cap U \neq \phi\} = \inf(r) | \inf(r) \cap L = \phi \text{ または } \inf(r) \cap U \neq \phi\}$$

ここで，

$SF(L,U)$: 公平性を満たす受理状態集合のクラス

$\inf(r)$: 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 $SF(L,U)$ を求めて受理状態集合のクラスを定義すればよい．

[例 3] （検証性質記述言語による公平性の記述例）

Kurshan の定義[13]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 $x \rightarrow$ infinitely often $\neg x$ の記述（時相論理では $\Box\Diamond x \rightarrow \Box\Diamond \neg x$，ここで，□: always, ◇: eventuality）[14]

公平性のオートマトンは図 8 (1) で定義できる．次に，$L = \{S0\}$, $U = \{S1\}$ と定義して受理状態集合のクラスを求める．

(a) $\inf(r) \cap \{S0\} = \phi$ のとき
$\inf(r) = \{S1\}$

(b) $\inf(r) \cap \{S1\} \neq \phi$ のとき
$\inf(r) = \{S1\}$ または $\{S0, S1\}$

以上より，受理状態集合のクラスは $\{\{S1\}, \{S0, S1\}\}$ である．

(2) infinitely often $x \rightarrow$ infinitely often y の記述（時相論理では $\Box\Diamond x \rightarrow \Box\Diamond y$，ここで，□: always, ◇: eventuality）[14]

公平性のオートマトンは図 8 (2) のように infinitely often $x \rightarrow$ infinitely often $\neg x$ と infinitely often $y \rightarrow$ infinitely often $\neg y$ のオートマトンのカルテジアン積で定義できる．次に，$L = \{S2, S4\}$ と $U = \{S2, S3\}$ を定義して受理状態集合のクラスを求める．

(a) $\inf(r) \cap \{S2, S4\} = \phi$ のとき
$\inf(r) = \{S1\}$ または $\{S3\}$ または $\{S1, S3\}$

(b) $\inf(r) \cap \{S2, S3\} \neq \phi$ のとき
$\inf(r) = \{S2\}$ または $\{S3\}$ または $\{S1, S2\}$ または $\{S1, S3\}$ または $\cdots \{S1, S2, S3, S4\}$

以上より，受理状態集合のクラスは $\{\{S1\}, \{S2\}, \cdots, \{S1, S2, S3, S4\}\}$ である． □

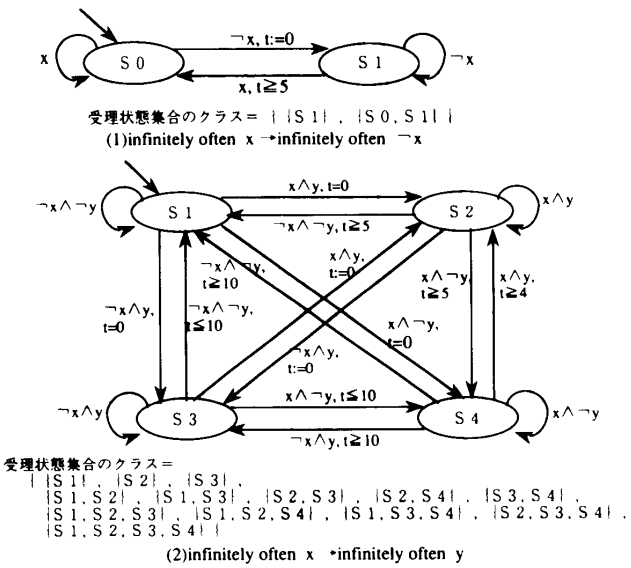


図 8 決定性時間 Muller オートマトンによる公平性の記述例
Fig. 8 Example of fairness by deterministic timed Muller automaton.

以上より，受理状態集合のクラスは $\{\{S1\}, \{S2\}, \cdots, \{S1, S2, S3, S4\}\}$ である． □

6.2 言語包合アルゴリズム

言語包合アルゴリズムは，積時間 Buchi オートマトンが決定性時間 Muller オートマトンに包合されるかどうかを判定する．まず，言語包合問題を定義する．

[定義 18] （言語包合問題の定義）[2]

言語包合問題は次のように定義できる．

$$L(M1) \subseteq L(M2)$$

また，これは以下と等価である．

$$L(M1) \cap \overline{L(M2)} = \phi$$

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，言語包合問題の決定性に関する定理を以下に説明する．

[定理 4] (言語包合問題の決定性)
時間ステートチャートと決定性時間 Muller オートマトンとの言語包合問題は決定性手続きが存在する.
(証明) 時間ステートチャートから積時間 Buchi オートマトンへの変換では, AND 分解は積を生成し, OR 分解は階層展開を行いながら状態階層をフラットにする. ステートチャートの状態階層は有限なので, 決定性手続きは存在する. また, 時間 Buchi オートマトンは積演算で閉じているので, 時間ステートチャートから時間 Buchi オートマトンが生成できる. 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合問題は決定性手続きが存在することが知られている[5]. 以上より決定性手続きの存在性が証明できた. □

言語包合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[5]に帰着できて, 以下のように定義できる.

[定義 19] (言語包合アルゴリズム)
言語包合アルゴリズムは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を Tarjan の深さ優先探索[15]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[6]を生成する
(3) Floyd-Warshall のアルゴリズムにより, 正規形 DBM[6]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, 言語包合アルゴリズムの計算コストを説明する.

[定理 5] (言語包合アルゴリズムの計算コスト)

$$0(|S|+|E|)+0(|AS|\cdot n^3)$$

ここで,
S: 積時間オートマトンの状態数
E: 積時間オートマトンの状態遷移数
AS: 受理ループの状態数 ($AS \leq S$)
n: クロック変数の数

(証明の概要)
(1) $0(|S|+|E|)$
深さ優先探索の計算コスト[16]より自明である.
(2) $0(|AS|\cdot n^3)$
Floyd-Warshall のアルゴリズムの計算コスト[12]より自明である. □

7. 検証システムと検証事例

7.1 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 9 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, 0.3 kstep, 1 kstep, 2.5 kstep の C 言語で実現した. コンパイラは実時間並行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.

7.2 検証事例

本節では, 自動車制御システムとイーサネットの検証を行った.

(1) 自動車制御システム

自動車制御システムはドライバや外部環境とインタフェースして, アクセルやエンジン, トランスミッション, ブレーキから構成される. ノーマルブレーキと

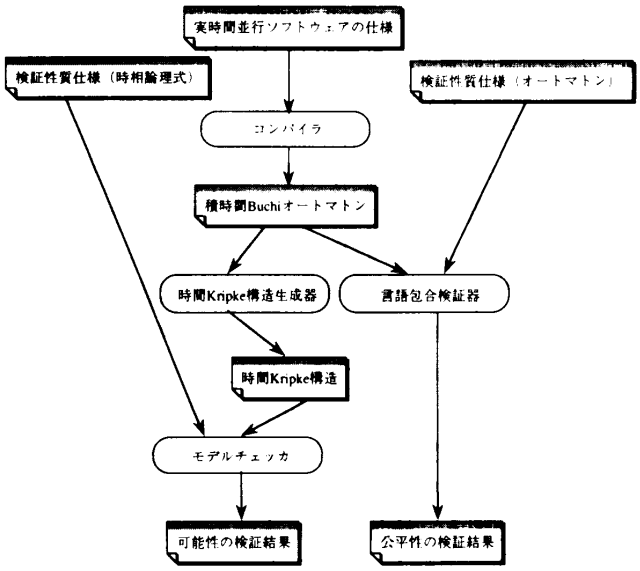


図 9 検証システムの構成
Fig. 9 Configuration of vertification system.

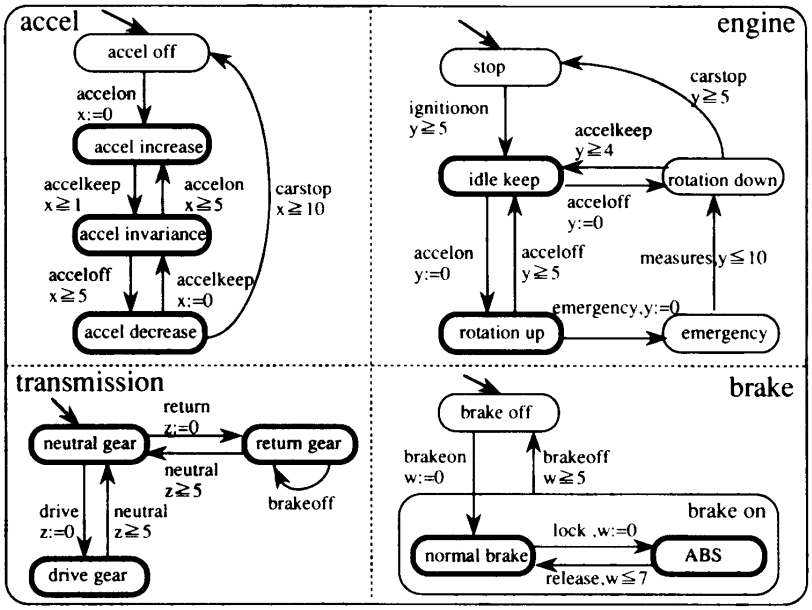
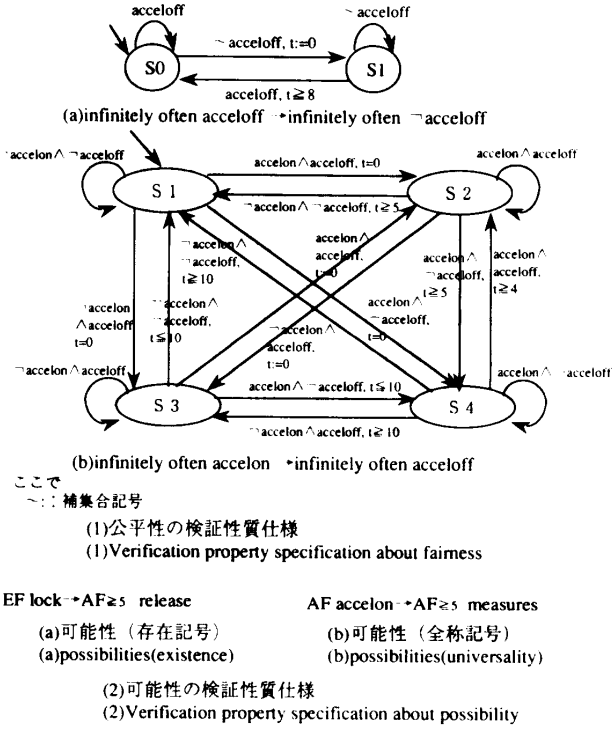


図10 自動車制御システムの仕様記述
Fig. 10 Specification of automobile control system.

ABS (Antilock Brake System) は brake on 状態に抽象化できる．ノーマルブレーキはドライバのブレーキ操作により作動して，ABS はスリップ率の上昇により車輪がロックしないように油圧を制御する．以上の自動車制御システムは**図10**の時間ステートチャートにより仕様記述できる．検証性質記述は**図11**のように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．公平性は infinitely often acceloff → infinitely often ¬ acceloff と infinitely often accelon → infinitely often acceloff であり，可能性は EF lock → $AF \geq 5$ release と AF accelon → $AF \geq 5$ measures である．

(2) イーサネット

イーサネットの CSMA/CD[17] 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12**の時間ステートチャートにより仕様記述できる．この図では，sender と receiver は並行の関係にあり，状態 S3 と状態 S6 を状態 S8 に抽象化する．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13**に示す．



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

EF lock→AF≧5 release
(a)可能性（存在記号）
(a)possibilities(existence)

AF accelon→AF≧5 measures
(b)可能性（全称記号）
(b)possibilities(universality)

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

図11 検証性質仕様
Fig. 11 Verification property specification.

(3) 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14**に例を示す．各々の状

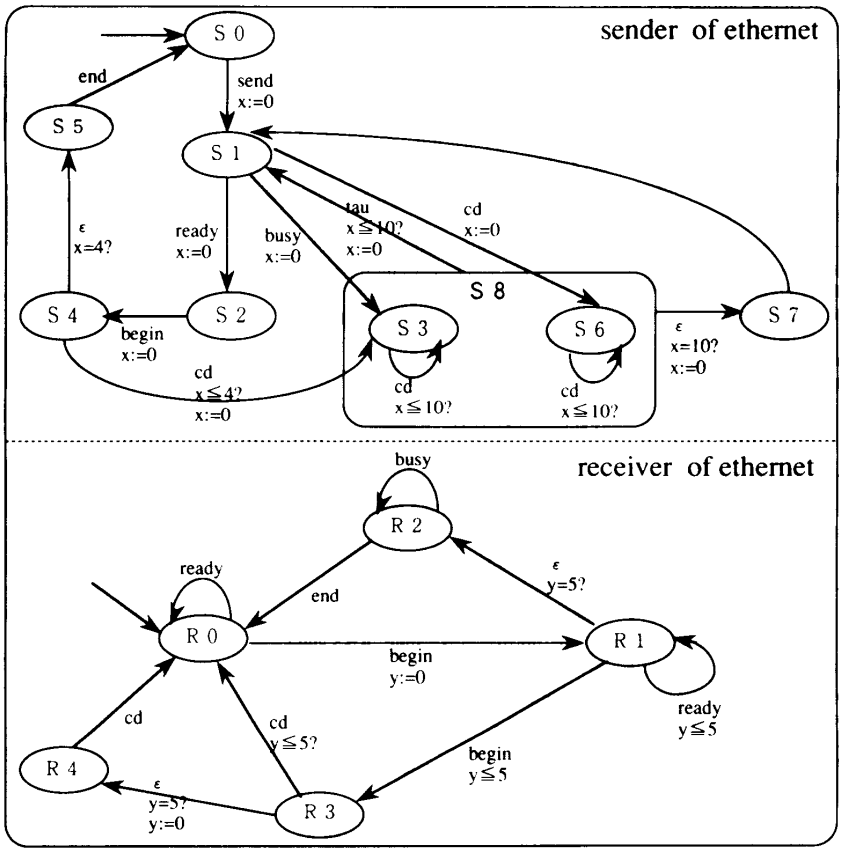
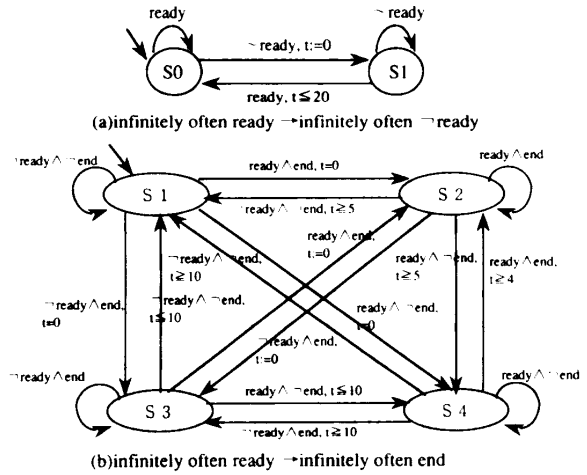


図 12 イーサネットの仕様記述
Fig. 12 Specification of ethernet.



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)possibilities(existence)　(b)possibilities(universality)

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

図 13 検証性質仕様
Fig. 13 Verification property specification.

態数と状態遷移数は数百であり，SUN4/IP（30 MB）での検証コストの実測結果を**表 1** に示す．表 1 では，検証に要した計算時間とメモリ量を示す．以上より，状態数が千未満の小規模な実時間並行ソフトウェアの検証は実用的なコストで実現できることが分かった．

8. ま と め

本稿では，実時間並行ソフトウェアを対象として，時間ステートチャート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り可能性と公平性を効率的に検証する手法を提案した．本手法を支援する検証システムを開発して，提案した手法の有効性を示した．これにより，数百の状態数を持つ中小規模の家電製品や自動車などの組込み型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分かった．しかし，本検証システムでは，バグの存在は分かるが，バグの具体的な場所は分からない．これは今後の検討課題である．また，現在は大規模システムを検証するために，状態遷移関係を BDD 表現するといった実時間並行ソフトウェアのシンボリック検証技術[18]を開発中である．

```
System specification ;
System configuration ;
  system=accel×engine×transmission×brake ;
Process specification(accel) ;
State definition part  acceloff state, accel increase state, accel invariance state, accel decrease state;
Event definition part  acceloff, accelkeep, accelon ,carstop;
Initial state definition part  acceloff ;
Acceptance state definition part  accel increase state, accel invariance state, accel decrease state  ;
State transition definition part
 accel off state →accelon, x:=0 →accel increase state ;
 accel increase state →accelkeep, x≧1→accel invariance state ;
 accel invariancestate→accelon, x≧5 →accel increase state ;
 accel invariance state→acceloff, x≧0→accel decrease state ;
 accel decrease state →accelon, x:=0→accel increase state ;
 accel decrease state →carstop, x≧10→accel off state ;
end ;
      :
      :
      :
Process specification(brake) ;
State definition part  brake off state, normal brake state, ABS state, brake on state ;
Substate definition part
  brake on state=normal brake state + ABS state ;
Event definition part  brakeon, brakeoff, lock, lockrelease ;
Initial state definition part  brake off state ;
Acceptance state definition part  normal brake state, ABS state ;
State transition definition part
 brake off state →brakeon, w:=0→brake on state ;
 brake on state →brakeoff, w≧5 →brake off state ;
 normal brake state →lock, w:=0 →ABS state ;
 ABS state→lockrelease, w≦7→normal brake state ;
end ;
```

図 14 コンパイラへの入力データ例
Fig. 14 Example of input data of compiler.

表 1 検証のメモリ量と計算時間の評価
Table 1 Evaluation of space and time complexity.

検証性質仕様		計算時間	メモリ量
自動車制御システム	公平性の検証性質仕様 (a)	133 sec	5500 kb
	公平性の検証性質仕様 (b)	226 sec	12333 kb
	可能性の検証性質仕様 (a)	111 sec	4512 kb
	可能性の検証性質仕様 (b)	122 sec	4213 kb
イーサネット	公平性の検証性質仕様 (a)	27 sec	552 kb
	公平性の検証性質仕様 (b)	55 sec	1321 kb
	可能性の検証性質仕様 (a)	23 sec	455 kb
	可能性の検証性質仕様 (b)	21 sec	432 kb

参考文献

1) Kavi, K.M.: *Real-time Systems, Abstraction, Language, and Design Methodologies*, IEEE Computer Society, p.660 (1992).
2) 山根:時間ステートチャートに基づくリアルタイムシステム検証方式, 情報処理学会論文誌, Vol.35, No.12, pp.2640–2650 (1994).
3) Alur, R. and Henzinger, T.A.: Logics and Models of Real Time: A Survey, *LNCS*, Vol.600, pp.74–106 (1992).
4) Alur, R., Courcoubetis, C.and Dill, D.: Model-Checking for Real Time Systems, *Proc. 5th IEEE Symp. on Logic in Computer Science*, pp.414–425 (1990).
5) Alur, R. and Dill, D.: The theory of Timed Automata, *LNCS*, Vol.600, pp.45–73 (1992).
6) Dill, D.: Timing Assumptions and Verification of Finite-state Concurrent Systems, *LNCS*, Vol.407, pp.197–212 (1989).
7) Alur, R., Courcoubetis, C., Dill, D., Halbwachs, N. and 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) Harel, D.: Statecharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, Vol.8, pp.231–274 (1987).
9) Harel, D., Pnueli, A., Schmidt, J.P. and Sherman, R.: On the Formal Semantics of Statecharts, *IEEE Logic in Computer Science*, pp.54–64 (1987).
10) Clarke, E.M., Emerson, E.A. and Sistla, A.P.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, *ACM Trans. on Programming Languages and Systems*, Vol.8, No.2, pp.244–263 (1986).
11) Hiraishi, H.: Design Verification of Sequential

Machines Based on ϵ-free Regular Temporal Logic, *Computer Hardware Description Languages and Their Applications*, Elsevier Science, pp.249–263 (1990).
12) 浅野：情報の構造（下），日本評論社，p.396 (1994).
13) Kurshan, R.P.: *Computer-aided Verification of Coordinating Processes: The Automata-Theoretic Approach*, Princeton University Press, p.270 (1994).
14) Emerson, E.A. and Lei, C.: Modalities for Model Checking: Brancing Time Logic Strikes Back, *Science of Computer Programming*, Vol.8, pp.275–306 (1987).
15) Tarjan, R.: Depth-first Search and Linear Graph Algorithms, *SIAM Journal of Computing*, Vol.1, No.2, pp.146–160 (1972).
16) 石畑：アルゴリズムとデータ構造，岩波書店，p.486 (1989).
17) IEEE ANSI/IEEE 802.3, ISO/DIS 8802/3. IEEE Computer Society Press (1985).
18) McMillan, K.L.: *Symbolic Model Checking*, Kluwer, p.194 (1993).

(平成7年 5 月25日受付)
(平成7年11月 2 日採録)

山根 智（正会員）

山口県出身．1984年京都大学工学研究科修士課程修了．同年富士通(株)入社．通信ソフトウェアの研究開発に従事．その後，島根大学理学部情報科学科勤務，現在に至る．通信ソフトウェア等の実時間システムの形式的な仕様記述や検証，自動生成に関する研究に従事．ソフトウェア工学や人工知能等に興味を持つ．EATCS，IEEE Computer Society，ACM，人工知能学会等，各会員．