Assume-Guarantee検証による実時間システムの階層的設計支援手法

山根 智[†]

通信システムや制御システムの多くはリアクティブな実時間システムである．実時間性やリアクティブ性のために，実時間システムの設計作業は複雑になりがちであり，また，システムの信頼性保証も一般的には困難である．したがって，実時間システムを階層的に詳細化して設計できて，それらの階層間の正当性が保証できることは重要である．本論文では，要求仕様から設計仕様を得るまでの各階層の仕様を同一の形式的仕様記述言語で記述し，階層間の整合性を構成的に自動検証するための枠組みを提案する．最後に，事例により，提案した階層的な設計支援手法の有効性を示す．

Hierarchical Design Support Method of Real-time Systems Based on Assume-Guarantee Verification

Satoshi Yamane[†]

In many cases, communication systems and control systems are reactive real-time systems. As designing real-time systems is complex for real-time properties and reactivity, it is difficult to guarantee their qualities. In designing real-time systems, the hierarchical design method is important and verifying whether lower levels satisfy upper levels or not is important, too. In this paper, we propose the hierarchical design method based on uniform specification techniques and Assume-Guarantee verification. Finally, we show the proposed method is effective by some example.

1. まえがき

通信システムや制御システムの多くは実時間システムである．さらに，これらは他のシステムと協調して動作するリアクティブシステムであることもしばしばである．したがって，これらのシステムの設計においては，各システムにおける処理時間が他のシステムに及ぼす影響や，システム間の通信遅延といった実時間性をも考慮する必要がある．このことにより，(リアクティブ) 実時間システムの設計作業は複雑になりがちであり，また，システムの信頼性保証も一般的には困難である[1),2)]．したがって，実時間システムを階層的に詳細化して設計できること，また，階層間の整合性の保証によって，最終的に得られるシステムの信頼性保証ができることが重要である[3),4)]．このことをふまえ，本論文では，要求仕様から設計仕様を得るまでの各階層の仕様を同一の形式的仕様記述言語で記述し，階層間の整合性を構成的に自動検証するための枠組みを提案する．具体的にいうと，本論文で提案するのは，仕様記述のためのリアクティブな時間オートマトンの一種，および，この時間オートマトンにおける時間模倣関係の Assume-Guarantee 検証である．Assume-Guarantee 検証形式[5)] は，並列合成プロセス間の整合性検証を効率的に行うための既知の形式[6)] であり，合成プロセスにおける状態の組合せ爆発が抑制できるものである．

従来の実時間システムの形式的検証の研究としては，時間オートマトンの言語包含検証[1)] やモデルチェッキング検証[2)]，時間模倣関係の証明系[3)]，時間模倣関係の自動検証[4)] などがある．階層的設計においては時間模倣関係の自動検証が適当であり，本論文では，時間模倣関係の自動検証手法[4)] を基礎とする．

また，検証を効率化するために，Assume-Guarantee 検証[5)] が注目されている．Assume-Guarantee 検証では，環境の動作を仮定して構成的に検証するものであり，状態の組合せ爆発が抑制できる．しかし，従来の Assume-Guarantee 検証は安全性や活性を対象とするものがほとんどであり，段階的詳細化における階層間の整合性検証に関するものはない．唯一の例外として，時

† 鹿児島大学工学部情報工学科
Department of Information and Computer Science, Kagoshima University

間モジュールの Assume-Guarantee 形式による階層的設計支援がある[6]．この研究では，Assume-Guarantee 検証の健全性を保証するために，receptiveness[7] の自動検証を実現している．しかし，この研究では，本研究と異なり，階層間の整合性関係として言語包含関係を使用しており，さらに Assume-Guarantee 検証手法の部分手続きとなる，階層間の整合性の具体的検証手続きを述べていない．

本論文では，環境との相互作用が表現できる時間 I/O オートマトンを定義して，要求仕様から設計仕様までを，時間 I/O オートマトンで統一的に仕様記述する．そして，時間模倣関係に基づく Assume-Guarantee 検証手法により，仕様の間の整合性を自動検証できる枠組みを定義して階層的な設計手法を提案する．さらに，その手法を実装して，その有効性を計算機実験により示す．

以下の本論文の構成は以下のとおりである．2 章では実時間システムの仕様記述手法を導入して，3 章では時間模倣関係を基礎とした階層的な設計手法を提案する．4 章では設計支援システムと設計事例を示して，5 章ではまとめを述べる．

2. 実時間システムの仕様記述手法と時間模倣関係

2.1 実時間システムの仕様記述手法

実時間システムでは多くのプロセスが並列動作している．実時間システムの要求仕様や設計仕様のプロセスは時間 I/O オートマトンで記述する．時間 I/O オートマトンは環境からの入力イベントに反応したり，環境に出力イベントを発生させながら状態遷移して動作する．また，状態において，時間経過して動作する．ここで，環境とは対象としているプロセス以外のプロセスである．まず，文献 1）を基礎として，時間 I/O オートマトンを形式的に定義する．

Definition 1（時間 I/O オートマトンの定義）

時間 I/O オートマトンは $P = (S, S_0, EVENT, C, E)$ の 5 つ組で定義される．ここで，

S は有限状態集合

$S_0 \subseteq S$ は初期状態集合

$EVENT$ はイベントの有限集合

C はクロックの有限集合

$E \subseteq S \times S \times EVENT \times 2^C \times \Phi(C)$ は遷移関係

ただし，イベントの有限集合は $EVENT = IN \cup OUT$ である．ここで，IN は入力イベントの有限集合，OUT は出力イベントの有限集合である．また，$\Phi(C)$ はクロック集合 C のタイミング制約式 δ であり，クロック集合 $C(x \in C)$ と非負整数の時刻定数 d により以下のように帰納的に定義される：

（1） $x \leq d$ や $d \leq x$ は δ である．

（2） δ_1 と δ_2 が δ ならば，$\neg\delta_1$ や $\delta_1 \wedge \delta_2$ は δ である．

状態遷移 $(s, s\prime, a, \lambda, \delta)$ は入力イベント a または出力イベント a による状態 s から状態 $s\prime$ までの遷移を表す．集合 $\lambda \subseteq C$ は，この状態遷移でリセットされるクロック集合を表す．以下，本論文では，

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

は，状態遷移 $(s, s\prime, a, \lambda, \delta) \in E$ を意味する．

時間 I/O オートマトンは時間付き言語 $(event, \tau)$ を受理する．ここで，$event = event_1, event_2, \ldots, \tau = \tau_1, \tau_2, \ldots$ である．ただし，$event_i \in EVENT$，$\tau_i \in \mathbf{R}$（$\mathbf{R}$：非負の実数）とする．時間 I/O オートマトンの走査列 r は，以下のような無限列である：

$$\langle s_0, \nu_0 \rangle \xrightarrow{event_1,\tau_1} \langle s_1, \nu_1 \rangle \xrightarrow{event_2,\tau_2} \langle s_2, \nu_2 \rangle \xrightarrow{event_3,\tau_3} \ldots$$

ここで，$s_i \in S$, $\nu_i \in [C \rightarrow \mathbf{R}]$ である． ■

また，時間 I/O オートマトンの意味は，時間付き言語で定義できる．その詳細は文献 1）と同様であり[1]，本論文では紙面の都合上から省略する．

実時間システムは多くの時間 I/O オートマトンから構成されていると考えて，実時間システムの仕様は時間 I/O オートマトンのカルテジアン積（並列合成）で表現する．我々は，プロセス間の相互作用や環境との相互作用を単純にモデル化するために，文献 7）に従って，以下の考え方により，実時間システムの構成を定義する．

（1） あるプロセスの入力イベントと他のプロセスの出力イベントが同じならば，その出力イベントによって入力イベントが生成されて，両方のプロセスは同期する．最終的には，このイベントは出力イベントとして認識する．

（2） プロセスの出力イベントどうしは同期しないと考える．すなわち，構成するプロセスの出力イベント間には，交わりがない．

以下では，時間 I/O オートマトンのカルテジアン積（並列合成）を定義する．

Definition 2（時間 I/O オートマトンのカルテジアン積）

プロセス $P_i = (S_i, S_{0i}, EVENT_i, C_i, E_i)$ $(i = 1, \ldots, n)$ に対して，以下のように実時間システム $P = (S, S_0, EVENT, C, E) = P_1 \parallel P_2 \parallel \ldots \parallel P_n$ を構成する．なお，$\parallel$ は並列合成を意味する．ただ

し，$EVENT_i = IN_i \cup OUT_i$ とする．並列合成できるための条件として，以下がある：すべての $i, j \in \{1, \ldots, n\}, i \neq j$ に対して，$OUT_i \cap OUT_j = \emptyset$ である．

(1) $S = S_1 \times \ldots \times S_n$．ただし，$\times$ はカルテジアン積である．

(2) $S_0 = S_{01} \times \ldots \times S_{0n}$．

(3) $IN = \bigcup_{i=1,\ldots,n} IN_i - OUT_{same}$，ただし，$OUT_{same}$ は入力イベントと等しい出力イベントの集合である．$OUT_{same} = \emptyset$ の場合は，入力イベントが出力イベントとすべて異なるときである．一方，$\bigcup_{i=1,\ldots,n} IN_i = OUT_{same}$ の場合は，入力イベントが出力イベントとすべて等しいときである．

(4) $OUT = \bigcup_{i=1,\ldots,n} OUT_i$．

(5) $C = \bigcup_{i=1,\ldots,n} C_i$．

(6) E は以下の規則で生成する：
以下では，

$$s_i \xrightarrow{a,\lambda_i,\delta_i} s_i'$$

かつ

$$s_j \xrightarrow{b,\lambda_j,\delta_j} s_j'$$

の場合に，生成規則を定義する．

(a) 入力イベントと出力イベントが同期するとき

(i) $a \in IN_i$ かつ $b \in OUT_j$ かつ $a = b$ のとき

$$s_i \times s_j \xrightarrow{b,\lambda_i \cup \lambda_j, \delta_i \wedge \delta_j} s_i' \times s_j'$$

(ii) $a \in OUT_i$ かつ $b \in IN_j$ かつ $a = b$ のとき

$$s_i \times s_j \xrightarrow{a,\lambda_i \cup \lambda_j, \delta_i \wedge \delta_j} s_i' \times s_j'$$

(b) その他のとき

$$s_i \times s_j \xrightarrow{a,\lambda_i,\delta_i} s_i' \times s_j$$

または

$$s_i \times s_j \xrightarrow{b,\lambda_j,\delta_j} s_i \times s_j' \quad \blacksquare$$

Example 1（実時間システムの構成例）

実時間システムの構成例を**図 1** に示す．まず，2 つのプロセスが与えられたとして，並列合成して，実時間システムを構成する．図 1 (1)，(2) は 2 つのプロセスを示す．並列合成の場合，(1) の入力イベント a は (2) の出力イベント a であるので，並列合成 (3) では

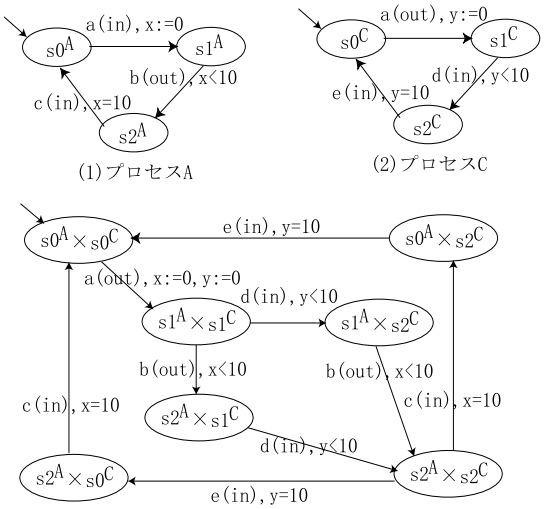

図 1 実時間システムの構成例
Fig. 1 Example of configuration of real-time systems.

a は入力イベントではなく，出力イベントである．図 1 (3) は，2 つのプロセスから並列合成された実時間システムを示す． ■

2.2 時間模倣関係の定義

次に，時間 I/O オートマトンの上で，時間模倣関係[4] を定義する．なお，本論文の時間模倣関係は，引用文献 4) 中では安全性時間模倣関係である．以下に，時間模倣関係を形式的に定義する．

Definition 3（時間模倣関係の定義）

上位レベルの仕様 $P^A = (S^A, S_0^A, EVENT^A, C^A, E^A)$ と下位レベルの仕様 $P^C = (S^C, S_0^C, EVENT^C, C^C, E^C)$ を任意の時間 I/O オートマトンとする．以下の条件を満たす $R \subseteq S^A \times S^C$ を P^A から P^C への時間模倣関係と呼び，そのような関係 R が存在するとき，$P^A \preceq P^C$ と記述する．

(1) 時間模倣関係
$(s_i^A, s_i^C) \in R$ ならば $\forall e^A \in EVENT^A, \lambda^A, \delta^A$ について，

$$s_i^A \xrightarrow{e^A,\lambda^A,\delta^A} s_{i+1}^A$$

である s_{i+1}^A が存在するならば，

$$s_i^C \xrightarrow{e^A,\lambda^A,\delta^A} s_{i+1}^C$$

である s_{i+1}^C が存在して，$(s_{i+1}^A, s_{i+1}^C) \in R$ である．なお，$s_i^A, s_{i+1}^A \in S^A$ かつ $s_i^C, s_{i+1}^C \in S^C$ である．

(2) 初期条件
$\forall s_0^A \in S_0^A$ に対して，$(s_0^A, s_0^C) \in R$ を満たす

$s_0^C \in S_0^C$ が存在する．ここで，$(s_0^A, s_0^C) \in R$ とは，上位レベルの初期状態から下位レベルの初期状態への時間模倣関係が存在することを意味する．すなわち，時間模倣関係の中に，初期状態が含まれていることを意味する． ■

一方，時間モジュールの検証の論文 6) では，言語包含関係が使用されている．言語包含関係は時間モジュール間のトレースの包含関係であり，本論文では時間オートマトン間のイベント列の包含関係に対応する．なお，論文 6) では，トレースは外部から観測可能な変数値の移り変わりである．本論文のイベントは，時間モジュールではイベントに対応する変数値への代入で表現できる．また，時間模倣関係は言語包含関係よりも強い関係であることが知られている[4]．言語包含検証は正則性や公平性を含む安全性や活性などが検証できる強力かつシンプルな検証手法であるが，仕様を非決定性時間オートマトンで記述すれば，言語包含検証は不可能である[1]．以上より，本論文では時間模倣関係を使用する．また，言語包含関係は存在するが時間模倣関係は存在しない場合が存在する．これは，言語包含関係が線形時間構造上の関係であり，時間模倣関係が分岐時間構造上の関係であることに起因する．分岐時間構造では，時間の構造は各時点がその直後の時点を複数個持っており，時間の構造は分岐した樹状の構造を持つ．すなわち，分岐時間構造は非決定性の動作をうまく表現できる．ゆえに，非決定性が存在する場合が多い実時間システムには時間模倣関係が適していると考えられる．

3. 時間模倣関係による階層的な設計手法

本論文では，言語包含関係[6] よりも強い関係である時間模倣関係[4] の Assume-Guarantee 検証を基礎として，時間 I/O オートマトンによる統一的な仕様記述による実時間システムの階層的な設計手法を提案する．最初に階層的な設計手法の概要を定義して，次に検証時に必要となるリージョングラフを定義して，最後に receptiveness の検証と Assume-Guarantee 形式の検証を定義する．なお，receptiveness はプロセスが物理的に実装可能な条件であり，プロセスが物理的に実装可能な条件とはプロセスが無限に動作する場合は経過時間が発散することを意味する．Assume-Guarantee 形式検証は対象プロセスと環境との並列合成により，システム全体を検証する手法である．また，Assume-Guarantee 形式検証では，対象プロセスと環境との並列合成で構成されるシステムが物理的に実装可能であることを保証する必要があり，このためにすべてのプロセスが receptive であることを保証する必要がある．

3.1 階層的な設計手法の概要

我々は，実時間システムの抽象度の高い上位レベルの仕様と抽象度の低い下位レベルの仕様を同一の時間 I/O オートマトンで仕様記述して，時間模倣関係の Assume-Guarantee 検証によりそれらの間の整合性を自動検証できる階層的な設計手法を提案する．

Definition 4（階層的設計手法）

実時間システムの階層的設計手法は以下の手順から構成される．

(1) まず，実時間システムの上位レベルのすべてのプロセスの仕様を時間 I/O オートマトンで記述する．
(2) 次に，上位レベルの仕様を基礎として，それを実現する下位レベルのすべてのプロセスの仕様を時間 I/O オートマトンで設計する．
(3) 次に，すべてのプロセスが receptiveness を満たすことを検証する．プロセスが receptiveness を満たすまで修正する．
(4) 次に，上位レベルの仕様から下位レベルの仕様への時間模倣関係の Assume-Guarantee 検証により，時間模倣関係の存在性を確認する．時間模倣関係が存在するまで，下位レベルの仕様を修正する．
(5) 以上のステップを繰り返して，最終的な実現仕様を設計する． ■

なお，receptiveness を満たさないプロセスは物理的に実装できないプロセスである．ゆえに，計算機に実装可能な正しいプロセスはすべて receptiveness を満たす．

3.2 リージョングラフ

時間 I/O オートマトンのタイミング制約記述の時間領域は稠密なので，クロック変数にそのまま時間を割り当てると，無限個の状態とクロック値のペアができてしまう．しかし，タイミング制約式の中の定数部分（たとえば $x < d$ の d の部分）が非負整数なので，クロック値の整数部分と小数部分の順序関係が同じならば，時間 I/O オートマトンの動作は区別されない．この考えから，我々は，動作が区別されないクロック割当ての同値関係により，クロックリージョンと呼ばれる同値類を構成する[1]．そして，クロックリージョンにより，時間 I/O オートマトンから，有限な商構造であるリージョングラフを構成することができる[1]．

まず，このクロック割当ての同値関係[1] を定義する．

Definition 5（クロック割当ての同値関係）

時間 I/O オートマトン P が与えられたとする．任

意のクロック変数 $x \in C$ に対して，P に現れる最大のクロック定数を c_x とする．また，時間 $t \in \mathbf{R}$ に対して，

$fract(t)$：t の小数部分

$\lfloor t \rfloor$：t の整数部分

とする．P のクロック割当ての集合を $\Gamma(P)$ とし，その要素を $\nu, \nu\prime \in [C \to \mathbf{R}]$ とする．以下の条件を満たすときに限り，$\nu \equiv \nu\prime$ と表す．

(1) 任意のクロック変数 $x \in C$ に対して，$\lfloor \nu(x) \rfloor$ と $\lfloor \nu\prime(x) \rfloor$ が同じ，または，$\nu(x)$ と $\nu\prime(x)$ の両方が c_x よりも大きい．

(2) $\nu(x) \le c_x$ と $\nu(y) \le c_y$ であるすべての $x, y \in C$ に対して，$fract(\nu\prime(x)) \le fract(\nu\prime(y))$ のときに限り，$fract(\nu(x)) \le fract(\nu(y))$ である．

(3) $\nu(x) \le c_x$ であるすべての $x \in C$ に対して，$fract(\nu\prime(x)) = 0$ のときに限り，$fract(\nu(x)) = 0$ である． ■

時間 I/O オートマトン P のクロックリージョンは同値関係 $\equiv$ によって導かれるクロック割当ての同値類であり，$[\nu]$ と記述する．

次に，クロックリージョンを基礎として，時間 I/O オートマトンのリージョングラフ[1] を定義する．

Definition 6（リージョングラフ）

時間 I/O オートマトン $P = (S, S_0, EVENT, C, E)$ のリージョングラフは $G = (Q, Q_0, EVENT, E\prime)$ で定義される．

ここで，

(1) Q：$\langle s, [\nu] \rangle$ の有限集合
  (a) $s \in S$
  (b) $[\nu] = \{\nu\prime | \nu \equiv \nu\prime\}$，$\nu \in \Gamma(P)$

(2) Q_0：$\langle s_0, [\nu_0] \rangle$ の有限集合
  (a) $s_0 \in S_0$
  (b) $[\nu_0]$：すべてのクロック変数が 0 の順序対の集合

(3) $EVENT$：有限イベント集合

(4) $E\prime \subseteq Q \times EVENT \times Q$：状態遷移関係 ■

3.3 Receptiveness の自動検証

実時間システムのプロセスの動作の正当性を保証する概念として，並列演算閉包性が存在し，Nonzeno より弱い概念である receptiveness[7] を使用する[6]．プロセスは，以下のいずれかの場合に receptive であるという．

(1) プロセスの経過時間が発散する．つまり，プロセスの状態遷移が無限回起きると，プロセスの経過時間も無限に大きくなる．

(2) プロセスの状態遷移が有限回起きる．

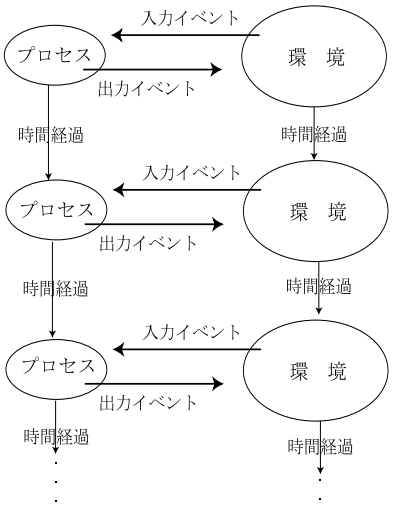

図 2 無限ゲームの概念図
Fig. 2 Concept of infinite games.

Definition 7（Receptiveness の定義）

Receptiveness はプロセスと環境（対象としているプロセス以外のプロセス）との無限ゲームの下で，プロセスに対して定義される．その様子を**図 2** に示す．ここで，プロセスおよび環境は以下のように動作する：プロセスは出力イベントにより状態遷移するか，時間経過する．環境は入力イベントを発生させるか，時間経過する．無限ゲームの戦略により，以下のように，次状態の動作が決定される：

(1) もし環境が入力イベントを提案するならば，環境の入力イベントが選択される．

(2) もし環境が時間経過を提案してプロセスが出力イベントを提案するならば，プロセスの出力イベントが選択される．

(3) もし環境とプロセスがともに時間経過を提案するならば，小さい時間経過が選択される．

上記戦略に従って動作するプロセスの経過時間が発散するか，またはプロセスの状態遷移が有限回ならば，プロセスは receptive である． ■

Definition 8（Receptiveness の自動検証手法）

無限ゲームにおいて，プロセスの経過時間が発散するか，プロセスの状態遷移が有限回ならば，プロセスは receptive である．すなわち，プロセスが無限に動作する場合に，プロセスの経過時間が発散するかどうかを確かめればよい．その検証手順は以下のとおりである．

(1) 経過時間を計測するために，リセットされないクロック変数 now を導入して，プロセスのリージョングラフ[1),4)] を構成する．

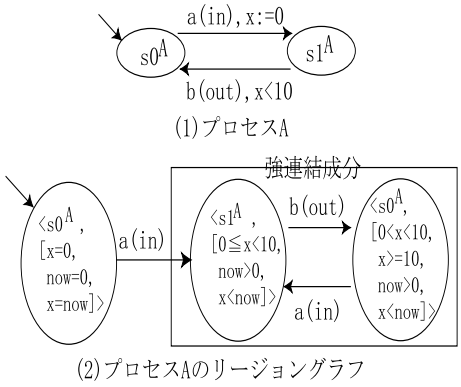

図 3 Receptiveness の事例
Fig. 3 Example of receptiveness.

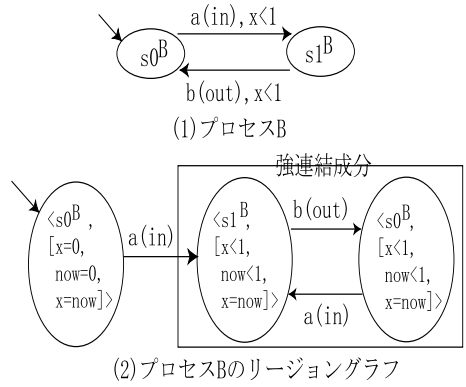

図 4 Receptiveness でない事例
Fig. 4 Example of nonreceptiveness.

(2) リージョングラフにおいて，初期状態集合から到達可能な強連結成分[8] の集合を計算する．強連結成分を計算することは，無限に動作する状態列を計算することに対応する．

(3) 強連結成分ごとに，以下のすべての条件を満たす強連結成分が存在しなければ receptive である．

(a) 強連結成分の枝に出力イベントがある．

(b) 強連結成分の節点の中で，now 以外のクロック変数が 0 でない．

(c) 強連結成分の節点の中で，最大のクロック定数に関するクロック制約式が $<$ または $\leq$ である．

この強連結成分はプロセスが有限時間内で無限に動作することを意味するので，この強連結成分が存在しなければプロセスは receptive である． ■

Example 2（Receptiveness の事例）

Receptiveness の事例を**図 3** に示す．図 3 (1) のプロセス A が receptive であるかどうかを検証する．まず，経過時間を計測するために，リセットされないクロック変数 now を導入して，プロセス A のリージョングラフを構成する．リージョングラフの強連結成分が無限な動作に対応している．この例の強連結成分では，プロセス A と環境が協調動作して，$x = 0$ を含むので，経過時間は発散する．ゆえに，プロセス A は receptive である．

Receptiveness でない事例を**図 4** に示す．まず，経過時間を計測するために，リセットされないクロック変数 now を導入して，プロセス B のリージョングラフを構成する．この例の強連結成分では，プロセス B と環境が協調動作して，$x < 1$ と $x < 1$ でループしており，$x = now$ なので，時間は発散しない．ゆえに，プロセス B は receptive でない． ■

3.4 Assume-Guarantee 形式による階層間の整合性検証

Assume-Guarantee 形式を基礎とする時間模倣関係による階層間の整合性の検証手法を定義する．プロセス 1 の抽象度の高い仕様 ${P_1}^A$ とプロセス 2 の抽象度の高い仕様 ${P_2}^A$ が並列動作する実時間システムがプロセス 1 の抽象度の低い仕様 ${P_1}^C$ とプロセス 2 の抽象度の低い仕様 ${P_2}^C$ が並列動作する実時間システムによって，正当に詳細化されていることを検証する場合を考える．この検証問題は，${P_1}^A \parallel {P_2}^A \preceq {P_1}^C \parallel {P_2}^C$ を示すことであるが，${P_1}^C \parallel {P_2}^C$ の状態数が多くなり，検証が困難である．そこで，以下のように，${P_1}^C \parallel {P_2}^C$ の構成を避けて，構成的に詳細化を検証する方法を提案する．ただし，${P_1}^A \parallel {P_2}^A$ は ${P_1}^A$ と ${P_2}^A$ が並列動作する実時間システムを意味する．

Theorem 1（Assume-Guarantee 検証）

もし，${P_1}^A \parallel {P_2}^C \preceq {P_1}^C$ と ${P_1}^C \parallel {P_2}^A \preceq {P_2}^C$ ならば，${P_1}^A \parallel {P_2}^A \preceq {P_1}^C \parallel {P_2}^C$ である．ただし，${P_1}^A$ と ${P_2}^A$，${P_1}^C$，${P_2}^C$ はすべて receptiveness を充足する．ここで，

${P_1}^A$：プロセス 1 の抽象度の高い仕様
${P_2}^A$：プロセス 2 の抽象度の高い仕様
${P_1}^C$：プロセス 1 の抽象度の低い仕様
${P_2}^C$：プロセス 2 の抽象度の低い仕様．

Proof 1

${P_1}^A = ({S_1}^A, {S_0}_1{}^A, EVENT, {C_1}^A, {E_1}^A)$ および
${P_2}^A = ({S_2}^A, {S_0}_2{}^A, EVENT, {C_2}^A, {E_2}^A)$,
${P_1}^C = ({S_1}^C, {S_0}_1{}^C, EVENT, {C_1}^C, {E_1}^C)$,
${P_2}^C = ({S_2}^C, {S_0}_2{}^C, EVENT, {C_2}^C, {E_2}^C)$ とする．
証明すべき条件は以下である：

もし，$({s_1}^A \times {s_2}^C, {s_1}^C) \in R_1$ かつ $({s_1}^C \times {s_2}^A, {s_2}^C) \in R_2$ ならば，$({s_1}^A \times {s_2}^A, {s_1}^C \times {s_2}^C) \in R$ である．
ただし，${s_1}^A \in {S_1}^A$，${s_2}^A \in {S_2}^A$，${s_1}^C \in {S_1}^C$，${s_2}^C \in {S_2}^C$ であり，R_1，R_2，R はそれぞれの時間模倣関係である．

1. すべてのプロセスが同期する場合を考える．
(1) $({s_1}^A \times {s_2}^C, {s_1}^C) \in R_1$ であり，すべてのプロセスは receptive なので，receptive の並列演算閉包性により，receptive なプロセス ${P_1}^A \parallel {P_2}^C$ と ${P_1}^C$ に関して，以下が成り立つ：
すべての ${e_1}^A \in EVENT$，${\lambda_1}^A$，${\delta_1}^A$，${\lambda_2}^C$，${\delta_2}^C$ について，

$$ {s_1}^A \times {s_2}^C \xrightarrow{{e_1}^A, {\lambda_1}^A \cup {\lambda_2}^C, {\delta_1}^A \wedge {\delta_2}^C} {s_1}^{A\prime} \times {s_2}^C $$

である ${s_1}^{A\prime} \times {s_2}^C$ が存在するならば，

$$ {s_1}^C \xrightarrow{{e_1}^A, {\lambda_1}^C, {\delta_1}^C} {s_1}^{C\prime} $$

である ${s_1}^{C\prime}$ が存在して，$({s_1}^{A\prime} \times {s_2}^C, {s_1}^{C\prime}) \in R_1$ である．
(2) $({s_1}^C \times {s_2}^A, {s_2}^C) \in R_2$ であり，すべてのプロセスは receptive なので，receptive の並列演算閉包性により，receptive なプロセス ${P_1}^C \parallel {P_2}^A$ と ${P_2}^C$ に関して，以下が成り立つ：
すべての ${e_2}^A \in EVENT$，${\lambda_2}^A$，${\delta_2}^A$，${\lambda_1}^C$，${\delta_1}^C$ について，

$$ {s_1}^C \times {s_2}^A \xrightarrow{{e_2}^A, {\lambda_1}^C \cup {\lambda_2}^A, {\delta_1}^C \wedge {\delta_2}^A} {s_1}^C \times {s_2}^{A\prime} $$

である ${s_1}^C \times {s_2}^{A\prime}$ が存在するならば，

$$ {s_2}^C \xrightarrow{{e_2}^A, {\lambda_2}^C, {\delta_2}^C} {s_2}^{C\prime} $$

である ${s_2}^{C\prime}$ が存在して，$({s_1}^C \times {s_2}^{A\prime}, {s_2}^{C\prime}) \in R_2$ である．

以上の (1) と (2)，receptiveness の並列演算閉包性により，以下が成り立つ：
すべての $e^A \in EVENT$，${\lambda_1}^A$，${\delta_1}^A$，${\lambda_2}^A$，${\delta_2}^A$ について，

$$ {s_1}^A \times {s_2}^A \xrightarrow{e^A, {\lambda_1}^A \cup {\lambda_2}^A, {\delta_1}^A \wedge {\delta_2}^A} {s_1}^{A\prime} \times {s_2}^A $$

である ${s_1}^{A\prime} \times {s_2}^A$ が存在するならば，

$$ {s_1}^C \times {s_2}^C \xrightarrow{e^A, {\lambda_1}^C \cup {\lambda_2}^C, {\delta_1}^C \wedge {\delta_2}^C} {s_1}^{C\prime} \times {s_2}^C $$

である ${s_1}^{C\prime} \times {s_2}^C$ が存在して，$({s_1}^{A\prime} \times {s_2}^A, {s_1}^{C\prime} \times {s_2}^C) \in R$ である．
すなわち，$({s_1}^A \times {s_2}^A, {s_1}^C \times {s_2}^C) \in R$ が成り立つ．
2. プロセスが同期しない場合も同様に証明できる．
以上により，定理の成り立つことが証明できた．■

時間模倣関係を Assume-Guarantee 形式で自動検証するために，時間模倣関係をリージョン模倣関係として検証する．なお，時間模倣関係をリージョン模倣関係として検証する手法は文献 4) で提案されている手法と同じである．このために，時間 I/O オートマトン上の時間模倣関係の存在性問題はリージョングラフ上のリージョン模倣関係の存在性問題に帰着できることを示す[4)]．

まず，上位レベルの仕様 $P^A = (S^A, {S_0}^A, EVENT, C^A, E^A)$ と下位レベルの仕様 $P^C = (S^C, {S_0}^C, EVENT, C^C, E^C)$ に対して，以下を定義する．

(1) $RG_{P^A \parallel P^C} = (Q, Q_0, EVENT, E)$ は，$P^A \parallel P^C$ のリージョングラフとする．ここで，
  (a) Q：$< s^A \times s^C, [\nu] >$ の有限集合
    (i) $s^A \in S^A$，$s^C \in S^C$
    (ii) $[\nu] = \{\nu\prime | \nu \equiv \nu\prime\}$，$\nu \in \Gamma(P^A \parallel P^C)$
  (b) Q_0：$< {s_0}^A \times {s_0}^C, [\nu_0] >$ の有限集合
    (i) ${s_0}^A \in {S_0}^A$，${s_0}^C \in {S_0}^C$
    (ii) $[\nu_0]$：すべてのクロック変数が 0 の順序対の集合
  (c) $EVENT$：有限イベント集合
  (d) $E \subseteq Q \times EVENT \times Q$：状態遷移関係
(2) $RG(s^A, s^C)$ は，$\langle s^A \times s^C, [\nu] \rangle$ または $\{\langle s^A, [\nu] \rangle, \langle s^C, [\nu] \rangle\}$ であり，状態の対 (s^A, s^C) が属するリージョングラフの同値類を表す．

以下に，リージョングラフ上のリージョン模倣関係を定義する．

Definition 9（リージョン模倣関係の定義）
任意の $RG({s_i}^A, {s_i}^C) \in \chi$ に対して，以下の条件が満たされるときに限り，$\chi \subseteq RG_{P^A \parallel P^C}$ は P^A から P^C へのリージョン模倣関係であるという：
(1) 任意の $event \in EVENT$ に対して，

$$ \langle {s_i}^A, [\nu_i] \rangle \xrightarrow{event} \langle {s_{i+1}}^A, [\nu_{i+1}] \rangle $$

ならば，$RG({s_{i+1}}^A, {s_{i+1}}^C) \in \chi$ である $\langle {s_{i+1}}^C, [\nu_{i+1}] \rangle$ に対して，

$$ \langle {s_i}^C, [\nu_i] \rangle \xrightarrow{event} \langle {s_{i+1}}^C, [\nu_{i+1}] \rangle $$

である．
(2) $\forall {s_0}^A \in {S_0}^A$ に対して，$RG({s_0}^A, {s_0}^C) \in \chi$ を満たす ${s_0}^C \in {S_0}^C$ が存在する．■

Theorem 2（時間模倣関係とリージョン模倣関係）
$RG({s_i}^A, {s_i}^C) \in \chi$ に対して，$R_\chi = \{({s_i}^A, {s_i}^C) | RG({s_i}^A, {s_i}^C) \in \chi\}$ とする．χ が P^A から P^C へのリージョン模倣関係である必要十分条件は，R_χ が P^A か

ら P^C への時間模倣関係 $P^A \preceq P^C$ である．

Proof 2

文献 4）の **Theorem 1** の証明と同様なので，本論文では省略する． ■

以上より，時間模倣関係の存在性は，リージョン模倣関係の存在性に帰着して，検証する．ゆえに，時間 I/O オートマトンの間の時間模倣関係の存在性の検証アルゴリズムは以下の手続きから構成される[4]．まず，下位レベルの仕様と上位レベルの仕様の時間 I/O オートマトンのカルテジアン積を構成して，次に，カルテジアン積のリージョングラフを構成して，最後に，上位レベルの仕様から下位レベルの仕様へのリージョン模倣関係の存在性を検証する．

Definition 10（時間模倣関係のアルゴリズム）

時間 I/O オートマトンの間の時間模倣関係に基づく検証アルゴリズムは以下の手続きから構成する．

(1) 上位レベルの仕様と下位レベルの仕様の時間 I/O オートマトンのカルテジアン積を構成する．

(2) カルテジアン積のリージョングラフを構成する．
  (a) タイミング定数の組合せによって，クロックリージョンを構成する．
  (b) 各状態をクロックリージョンに分割して，リージョングラフの状態集合を構成する．
  (c) リージョングラフの状態集合をイベントで結ぶ．

(3) リージョングラフ上のリージョン模倣関係を次のような手続きでチェックする．基本的考え方はリージョン模倣関係 R を $R(0), \ldots, R(k), R(k+1)$ と帰納的に計算する．そして，$R(k) = R(k+1)$ となる $R(k)$ を $R = R(k)$ とする．なお，上位レベルの仕様 $P^A = (S^A, {S_0}^A, EVENT, C^A, E^A)$ と下位レベルの仕様 $P^C = (S^C, {S_0}^C, EVENT, C^C, E^C)$ に対して，$P^A \parallel P^C$ のリージョングラフを $RG_{P^A \parallel P^C} = (Q, Q_0, EVENT, E)$ とする．ここで，

  Q：$\langle {s^A} \times {s^C}, [\nu] \rangle$ の有限集合
    ${s^A} \in S^A$，${s^C} \in S^C$
    $[\nu] = \{\nu\prime | \nu \equiv \nu\prime\}$，$\nu \in \Gamma(P^A \parallel P^C)$
  Q_0：$\langle {s_0}^A \times {s_0}^C, [\nu_0] \rangle$ の有限集合
    ${s_0}^A \in {S_0}^A$，${s_0}^C \in {S_0}^C$
    $[\nu_0]$：すべてのクロック変数が 0 の順序対の集合
  EVENT：有限イベント集合
  $E \subseteq Q \times EVENT \times Q$：状態遷移関係

$R(k)$ を計算するアルゴリズムは文献 4）の Definition 12 と同様なので，本論文では省略する． ■

4. 実時間システムの階層的な設計の事例

4.1 設計支援システム

本手法を支援する設計支援システムは**図 5** のように receptiveness 検証器と Assume-Guarantee 検証器から構成した．

(1) receptiveness の検証は，receptiveness の並列演算閉包性により，プロセスごとに実現できる．receptiveness 検証器では，プログラミング言語形式で入力したプロセスから，時間 I/O オートマトンのリージョングラフを生成して，receptiveness の条件をチェックする．

(2) 時間模倣関係の Assume-Guarantee 検証は，リージョングラフ上の模倣関係として自動検証する．なお，リージョングラフは隣接リスト構造で表現する．

本設計支援システムにより，上位レベルの仕様 ${P_1}^A$，${P_2}^A$ から下位レベルの仕様 ${P_1}^C$，${P_2}^C$ への時間模倣関係が存在するかどうかを Assume-Guarantee 形式で検証する．まず，プロセスごとに，receptiveness の充足性を検証して，receptiveness を充足するまでプロセスを修正する．次に，Assume-Guarantee 形式により時間模倣関係の充足性を検証して，充足するならば，さらに下位レベルの仕様を具体化して，詳細な下位レベルの仕様を設計する．そうでなければ，下位レベルの仕様を再設計して，時間模倣関係の存在性を調べる．そして，同様に，下位レベルの仕様から詳細な下位レベルの仕様への時間模倣関係が存在するかどうかを検証する．以上の設計作業を続けて，最終的な実現仕様を設計する．

なお，SUN ULTRA（メインメモリ 24 MB, 143 MHz）上にインプリメントした設計支援システムの全体規模は C 言語で 7.3 Kstep である．

4.2 階層的な設計の事例

本論文では，イーサネットの CSMA/CD[9] を基礎とした事例により，提案した階層的な設計手法の有効性を示す．

4.2.1 CSMA/CD の仕様記述

イーサネットの CSMA/CD は LAN で広く使われており，送信局と受信局からなり，以下に送信局と受信局の各々を詳細に説明する．

1. 送信局はデータを送信する ($send_i$) 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 ($begin_i$)．しかし，チャネ

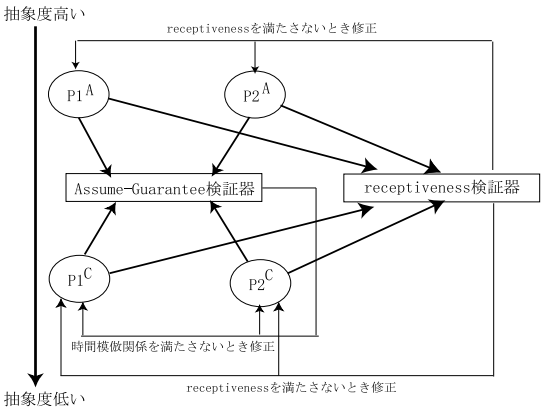

図 5 設計支援システムの構成
Fig. 5 Configuration of a design support system.

ルが busy であったり ($busy_i$) データが破壊されたら (cd_i)，10 時刻未満待って再送する (tau_i)．このイーサネットの CSMA/CD プロトコルの送信局の仕様は以下のように記述できる．

(1) まず，**図 6** (1) のように，CSMA/CD プロトコルの i-送信局の上位レベルの仕様を設計して，時間 I/O オートマトンで仕様記述する．この仕様は，データを送信すると，チャネルの busy やデータ破壊を繰り返して，正常にデータを送信することを意味する．

(2) 次に，図 6 (2) のように，CSMA/CD プロトコルの i-送信局の下位レベルの仕様を設計して，時間 I/O オートマトンで仕様記述する．下位レベルの仕様では，初期状態から非決定的に動作する．一方の動作はチャネルの busy やデータ破壊を繰り返した後，正常に動作して，他方の動作はネットワーク上の何らかのトラブルにより，データの送信が失敗する可能性があることを意味する．

2. 受信局はデータ受信を開始し ($send_i$)，データが送信されると ($begin_i$)，10 時刻以内でデータを受信する (end_i)．もし，データ受信中にチャネルが busy になったら ($busy_i$)，10 時刻以内のデータの受信待ちをして ($fail_i$)，その後データの受信を再開する．このイーサネットの CSMA/CD プロトコルの受信局の仕様は以下のように記述できる．

(1) まず，**図 7** (1) のように，CSMA/CD プロトコルの i-受信局の上位レベルの仕様を設計して，時間 I/O オートマトンで仕様記述する．この仕様は，データを受信すると，チャネルの busy やデータ破壊を繰り返したり，正常にデータが受信できたりすることを意味する．

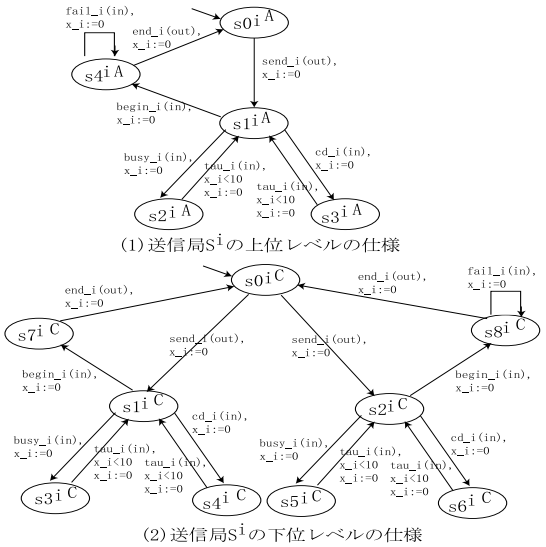

図 6 送信局の仕様記述例
Fig. 6 Example of specification of sender.

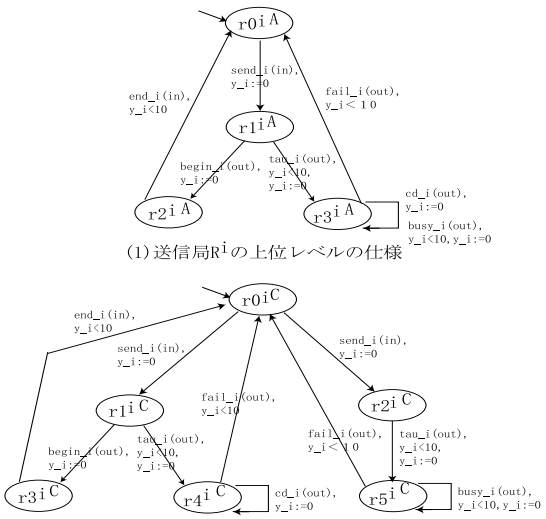

図 7 受信局の仕様記述例
Fig. 7 Example of specification of receiver.

(2) 次に，図 7 (2) のように，CSMA/CD プロトコルの i-受信局の下位レベルの仕様を設計して，時間 I/O オートマトンで仕様記述する．下位レベルの仕様では，初期状態から非決定的に動作する．一方の動作はチャネルの busy を繰り返した後，データ受信を失敗し，他方の動作はネットワーク上のトラブルがまったくなく，データの受信を正常に行えたり，データ破壊によりデータ受信が失敗したりすることを意味する．

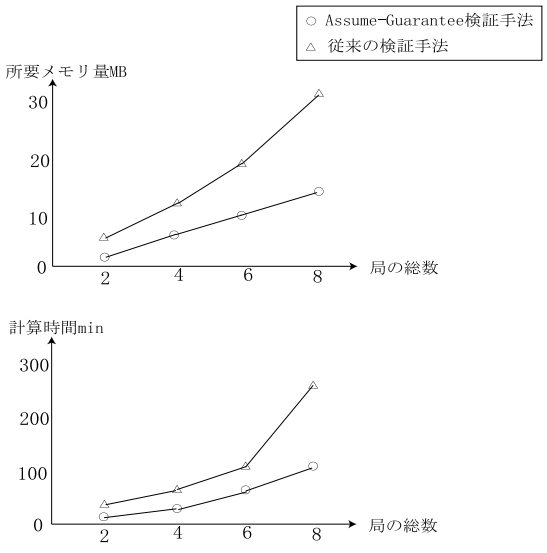

図 8 時間模倣関係の検証コスト
Fig. 8 Verification costs of timed simulation relations.

4.2.2 階層間の整合性の検証実験

次に，Assume-Guarantee 形式の時間模倣関係により，設計した上位レベルと下位レベルの仕様の整合性を検証する．本論文では，通信システムは複数の送信局と受信局の仕様から構成されるものと考える．すなわち，通信システムは送信局と受信局の仕様のカルテジアン積である．整合性検証問題では，これらの通信システムの下位レベルの仕様が上位レベルの仕様に対して，時間模倣関係が存在するかどうかを判定する．すなわち，$S^A \parallel R^C \preceq S^C$ と $S^C \parallel R^A \preceq R^C$ により，$S^A \parallel R^A \preceq S^C \parallel R^C$ を検証する．ただし，S^A と R^A，S^C，R^C は，すべて receptiveness を充足する．ここで，S^A と R^A は送信局および受信局の上位レベルの仕様，S^C と R^C は送信局および受信局の下位レベルの仕様である．

まず，プロセスごとに，receptiveness の充足性を検証した．図 6 と図 7 のすべてのプロセスは receptiveness を充足していることが分かった．次に，送信局と受信局の総数が 2 個，4 個，6 個，8 個の場合の整合性の検証実験を行って，Assume-Guarantee 検証手法と従来検証手法との比較評価を行った．UNIX の time コマンドで計測した検証の所要メモリと計算時間の比較結果を**図 8** に示す．実験結果より，Assume-Guarantee 検証により，大幅に検証コストが削減できることが分かった．

5. むすび

本論文では，要求仕様から設計仕様を得るまでの各階層の仕様を同一のリアクティブな時間オートマトンの一種で記述し，階層間の整合性を時間模倣関係の Assume-Guarantee 形式で自動検証するための枠組みを提案した．そして，計算機実験により，その有効性を実証した．類似研究として，時間モジュールの Assume-Guarantee 形式による階層的設計支援[6] が存在するが，本研究と異なり，階層間の整合性関係として言語包含関係を使用しており，さらに Assume-Guarantee 検証手法を述べていない．また，時間制約を考慮しないリアクティブシステムの階層的な設計を支援するシステムとしては，Cuncurrency workbench[10] や SMC[11] など多く存在する．一方，実時間システムの階層的な設計を支援するシステムとしては，Real-time Step[12] や KRONOS[13]，Real-time COSPAN[14]，UPPAAL[15] などが存在する．Real-time Step は実時間時相論理の証明系であり，KRONOS と UPPAAL は実時間シンボリックモデルチェッカであり，階層設計にはあまり有効ではない．また，Real-time COSPAN は抽象度の異なる仕様間に準同形写像を定義して，階層的設計を支援しているが，手作業で準同形写像を定義する必要があり，あまり実用的ではない．以上により，世界ではじめて，実時間システムの Assume-Guarantee 形式の自動階層設計支援システムが構築でき，その有効性が確認できた．

今後の課題としては以下が考えられる．

(1) BDDs（Binary Decision Diagrams）[16] などを基礎とするシンボリック検証技術による検証の所要メモリ量の削減
(2) 抽象実行などによる所要メモリ量と計算時間の削減

謝辞 懇切丁寧に，洞察力の豊富なコメントをいただきました査読者各位に感謝いたします．本研究は文部省科研費基盤 C「時相論理と並行計算，オートマトンの統合化による自律性のある分散システムの設計支援（代表：山根智）」の援助の下で実施されました．

参考文献

1) Alur, R. and Dill, D.: The theory of timed automata, LNCS, Vol.600, pp.45–73 (1992).
2) Alur, R., Courcoubetis, C. and Dill, D.: Model checking for real-time systems, *Proc. 5th LICS*, pp.414–425 (1992).
3) Lynch, N.A. and Attiya, H.: Using mapping to prove timing properties, *Distributed Computing*, No.6, pp.121–139 (1992).
4) 山根 智：時間模倣関係による実時間システムの階層的設計手法，情報処理学会論文誌，

Vol.40, No.7, pp.3001–3015 (1999). (preliminary versions appeared in FM-Trends 98, LNCS, Vol.1641, pp.151–167, 1998).
5) Pnueli, A.: *Logics and Models of Concurrent Systems*, Transition From Global to Modular Temporal Reasoning about Programs, NATO ASI Series, pp.123–144 (1985).
6) Alur, R. and Henzinger, T.A.: Modularity for timed and hybrid systems, LNCS, Vol.1243, pp.74–88 (1997).
7) Gawlick, R., Segala, R., Sogaard-Andersen, J.F. and Lynch, N.: Liveness in Timed and Untimed Systems, *Information and Computation*, Vol.141, pp.119–171 (1998).
8) Cormen, T.H., Leiserson, C.E. and Rivest, R.L.: *Introduction to algorithms*, MIT Press (1990).
9) IEEE Computer Society: IEEE ANSI/IEEE 802.3, ISO/DIS 8802/3. IEEE Computer Society Press (1985).
10) Cleaveland, R., Parrow, J. and Steffen, B.: The concurrency workbench, LNCS, Vol.407, pp.24–37 (1989).
11) Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D. and Hwang, L.J.: Symbolic Model Checking: 10^{20} States and Beyond, *Proc. 5th LICS*, pp.428–439 (1990).
12) Kesten, Y., Manna, Z. and Pnueli, A.: Verifying clocked transition systems, LNCS, Vol.1066, pp.13–40 (1996).
13) Daws, C., Olivero, A., Tripakis, S. and Yovine, S.: The tool KRONOS, LNCS, Vol.1066, pp.208–219 (1996).
14) Alur, R. and Kurshan, R.: Timing analysis in COSPAN, LNCS, Vol.1066, pp.220–231 (1996).
15) Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P. and Wang, Y.: UPPAAL-a tool suite for automatic verification of real-time systems, LNCS, Vol.1066, pp.232–243 (1996).
16) Bryant, R.E.: Graph-based algorithms for boolean function manipulation, *IEEE Trans. Comput.*, Vol.C-35, No.8, pp.677–691, IEEE Computer Society (1986).

(平成 12 年 1 月 31 日受付)
(平成 12 年 10 月 6 日採録)

山根 智（正会員）

昭和 59 年京都大学大学院修士課程修了．現在，鹿児島大学工学部助教授．工学博士．リアルタイムシステムの演繹的検証と自動演繹的検証の研究に従事．EATCS や ACM，IEEE 等会員．