ソフトウェア紹介

リアルタイムシステムの形式的手法とその検証ツール

山根 智

1 はじめに

1980 年代より，非同期回路や通信プロトコルなどの分野では，タイミング制約を含む仕様記述と検証のニーズがあった．そのような要請の中で，1989 年に，非同期回路を対象として，ハーバード大学の H.R. Lewis が有限状態機械に時間遅延を記述できるように拡張した仕様記述言語とその検証手法 [1] を開発した．その直後に，スタンフォード大学の D.L. Dill が実数を値とする，リセット可能なクロック変数を導入することにより，H.R. Lewis の手法を簡単化して，初めて，Buchi 受理条件を持つ時間オートマトンと DBMs(Difference Bounds Matrices) を用いた検証手法 [2] を開発した．このクロック変数の導入と DBMs の開発は，時間オートマトンを世の中に流行らせる，重要な道具立てとなった．さらに，1990 年に，R. Alur と D.L. Dill は時間付き言語を受理する時間オートマトンの理論 [3] を開発して，同時に，実時間時相論理に対する時間オートマトンのモデル検査手法 [4] を開発した．一方，1992 年に，ラトビア大学の K. Cerans が時間オートマトンの双模倣検証の決定可能性 [5] を示した．以上の R. Alur と D.L. Dill らの仕事 [3][4][5] により，時間オートマトンの仕様記述と検証の理論が構築されたと言ってよい．

これ以降は，時間オートマトンの効率的な検証方式が盛んに研究されて，有望な検証ツールも開発されてきており，2007 年の時点では，ほぼ落ち着いた感がある．まず，1992 年に，T.A. Henzinger らは時間オートマトン [3] の状態にタイミング制約を追加して，現在標準的に使われている時間オートマトンを開発して，実時間時相論理に対する時間オートマトンの実時間記号モデル検査手法 [6] を開発して，検証ツール KRONOS [7] を実装したが，大規模なシステムの検証は困難であった．その後，二分決定グラフ (BDDs) などを用いた近似検証手法 [8][9][10]，二分決定グラフ (BDDs) を拡張した記号検証手法 [11][12]，抽象化や構成的な検証手法 [13]，on-the-fly 検証手法 [14]，述語抽象化検証手法 [15] などが研究されている．

現在，最も注目されている時間オートマトンの検証ツールとしては，モデル検査ツール UPPAAL [16] がある．UPPAAL が注目されている理由としては，(1) 優れた GUI を備えており，使いやすいこと，(2) 様々な検証技術 (抽象化や構成的な検証，on-the-fly 検証，など) を実装しており，実用的な規模のシステムの検証が可能であること，(3)10 年間以上に渡り，多数の重要な検証事例を開発しており，実問題の適用経験を踏まえた機能拡張を随時行っており，ツールの鮮度が高く，完成度も高いこと，(4)20 名程度の人員が開発と保守などに従事しており，バグ修正などの体制が充実していること，などが考えられる．なお，UPPAAL に関する，仕様記述と検証技術のサーベイ論文 [17] , チュートリアルや多数の事例が HP(http://www.uppaal.com/) にあり，アカデミックユースは無料であるが，日本国内にビジネスユース向けの販売代理店が出来たようである．また，時間オートマトンの形式的検証に関しては，台湾国立大学

Formal Methods and Verification Tools for Real-Time Systems.

Satoshi Yamane, 金沢大学大学院自然科学研究科, Graduate School of Natural Science and Technology, Kanazawa University.

コンピュータソフトウェア, Vol.25, No.3 (2008), pp.81–87.
2008 年 1 月 21 日受付.

の F. Wang 教授の優れたサーベイ論文 [18] がある．

以下では，2 節で UPPAAL における仕様記述として時間オートマトンや時相論理などを紹介する．3 節でモデル検査ツール UPPAAL による検証事例を紹介する．最後に，4 節で周辺の動向を紹介する．

2 UPPAAL における仕様記述

モデル検査ツール UPPAAL は有界な整数などで拡張した時間オートマトン [3] [6] をシステムの仕様記述言語として，分岐時相論理 TCTL (Timed Computation Tree Logic) [4] [6] のサブセットを検証性質の記述言語としている．UPPAAL は時間オートマトンで記述されたシステムが時相論理で記述された検証性質を満たすかどうかを検証する．

2.1 時間オートマトン

時間オートマトンは実数値が割り当てられるクロック変数で拡張された有限状態機械である [3] [6]. UPPAAL では，リアルタイムシステムは時間オートマトンの並列合成としてモデル化される．また，モデルの中で，有界な整数の変数が記述できて，その変数は読み込みと書き込みができたり，算術演算ができる．さらに，現時点の最新バージョンでは，チャネルのブロードキャストへの対応やループ文を含むファンクション定義などができるように記述能力がかなり上がっている．

図 2.1(a) はランプをモデル化した時間オートマトンを示す．ここで，y はクロック変数であり，y:=0 は y のリセットを表し，y<5 と y>=5 は条件式を表し，press! はボタンを押すこと (ボタンの出力) を表し，press? はボタンを押されること (ボタンの入力) を表す．Lamp は **off**，**low** と **bright** の 3 つのロケーションを持ち，◎のロケーション **off** は初期状態を意味する．もし User がボタンを押す (press!) ならば，press? と同期して，Lamp はつけられる．もし User が再度，ボタンを押す (press!) ならば，Lamp は消される．しかし，User が速く (y<5)2 度ボタンを押す (press!) ならば，Lamp は明るくなる．User は図 2.1(b) のようにモデル化される．また，Lamp の動作はロケーションとクロック変数の値の順序対 (状態) の時系列により表わされる．以下に，Lamp の動作例を示す．

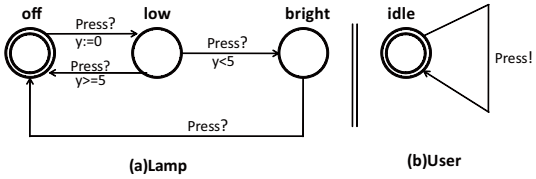

図 2.1 ランプの例

<Lamp.**off**,y=0> → <Lamp.**off**,y=3> $\stackrel{Press?}{\longrightarrow}$
<Lamp.**low**,y=0> → <Lamp.**low**,y=0.5> $\stackrel{Press?}{\longrightarrow}$
<Lamp.**bright**,y=0.5> → <Lamp.**bright**,y=1000>
→ . . . . . .

2.2 時相論理による検証性質記述

UPPAAL での時相論理は TCTL のサブクラスであり，$A\Box$, $E\Box$, $A\Diamond$, $E\Diamond$ のみが記述可能であり，これらのネストを記述できない．ここで，パス限量記号は A(あらゆる計算木で) と E(ある計算木で) であり，時相演算子を含む状態論理式は $\Box p$(常に p が成り立つ) と $\Diamond p$(いつかは p が成り立つ) である．ただし，例外として，*leads to* オペレータがあり，$f - - > g$ を $A\Box(f$ imply $A\Diamond g)$ の略記として記述できる．

まず，代表的な検証性質として，到達可能性，安全性と活性を説明する．

1. 到達可能性の検証では，ある状態論理式 ϕ が到達可能な状態によって満たされるかを検証する．到達可能性の検証性質を $E\Diamond\phi$ と仕様記述して，UPPAAL では，$E <> \phi$ と表記する．
2. 安全性の検証では，何も悪いことは起きないことを検証する．安全性の検証性質の代表的な例として，到達可能なすべての状態で状態論理式 ϕ が成り立つがあり，$A\Box\phi$ と仕様記述できて，UPPAAL では，$A[\,]\phi$ と表記する．
3. 活性の検証では，いつかよいことが起こることを検証する．活性の検証性質の代表的な例として，いつかは状態論理式 ϕ が成り立つがあり，$A\Diamond\phi$ と仕様記述できて，UPPAAL では，$A <> \phi$ と表記する．

次に，以下の例を用いて，リアルタイム性の検証性

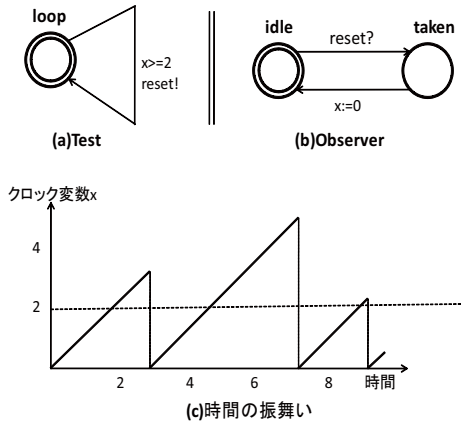

図 2.2 リアルタイム性の検証性質の例

質を説明する．

ここでは，図 2.2(a)Test と (b)Observer の 2 つの時間オートマトンが並列動作して，reset? と reset! が同期している．時間に従ってクロック変数 x がどのように変化するのかの一例を図 2.2(c) に示す．遷移でクロック変数がゼロにリセットされることがあるが，クロック変数の値は常に一定量が稠密で増加して，クロック変数が複数あってもそれらの増加量は同じである．また，遷移は瞬時に起きる．この例におけるリアルタイム性の検証性質の一例は，UPPAAL では，

(1)$A[\ ]$ Observer.**taken** *imply* $x >= 2$,

(2)$E <>$ Observer.**idle** *and* $x > 3$

などと記述される．

3 モデル検査ツール UPPAAL による検証事例

3.1 モデル検査ツール UPPAAL の概要

モデル検査ツール UPPAAL [19] はクライアントサーバアーキテクチャで実装されており，様々なプラットフォーム (Linux, windows, Solaris) 上で動作する．UPPAAL [19] は editor, simulator と verifier の 3 つの機能からなる．ユーザは，まず，editor を用いて，時間オートマトンでシステムを仕様記述して，次に，simulator を用いて，システムが意図どおりに動作するか確認して，最後に，verifier を用いて，時相論理式で記述した検証性質をシステムが満たすかどうかをモデル検査する．これら 3 つの機能は優れた GUI を提供しており，容易に操作できる．その操作方法は，資料 [19] [23] にわかりやすく記述されており，紙面の都合上から，本稿では割愛する．

3.2 モデル化の例

UPPAAL の検証事例は多数報告されており，通信プロトコルから制御ソフトウェア，リアルタイムステートチャートまで多岐に渡り，それらの事例の論文が HP [19] にある．本稿では，初期の事例である Audio/Video プロトコル [20] を取り上げる．この事例を取り上げる理由としては，(1) 優れたモデル化のテクニック (変数，状態や並列構造の決め方など) が内在しており，他のモデルを作成する際に大いに参考となること，(2) 他の有力な検証ツールの中でも適用されており，形式的手法の成功事例として国際的に著名であること，などである．

検証の対象となるシステムは，オーディオコントローラ，テレビ，ビデオなどの構成要素がブロードキャストバスで結合されており，Audio/Video プロトコルで通信する．ここで，信号はフレーム単位に通信して，信号の衝突が起きると，Audio/Video プロトコルはその衝突を検知する．

K. Havelund らが行った検証作業の流れは以下である [20].

1. まず，Audio/Video プロトコルを実装した 2800 行のアセンブリコードと 3 個のフローチャートを使って，UPPAAL のエキスパートが Audio/Video プロトコル開発者とコミュニケートしながら，以下のように，システムの仕様を明確にした．
   (a) まず，システム全体を 4 つのフェーズ，(1) アイドルフェーズ (新しいフレームが伝送可能となるのを待つ)，(2) 初期化フェーズ (バスのリザーブを待っている)，(3) 伝送フェーズ (フレームの伝送中)，(4) 衝突回避フェーズ (衝突検知後に遷移する衝突回避処理)，に分割して，システムを構造化した．
   (b) 次に，適度な抽象度でシステムの仕様を明確にして，バスのリザーブのルール，フレーム間の間隔のルール，フレーム初期化

のルール，バスのサンプルのルール，バスへの出力のルール，衝突検知のルール，衝突対処のルール，伝送停止のルール，検知停止のルール，プロトコルの正しさのルール，を定義した．

ここでは，検証すべきプロトコルの正しさのルールとして，以下を考える:

『送信する構成要素 x(テレビなど) によって伝送されるフレームが衝突で破壊されるならば，x がその衝突を検知して，さらに，送信する構成要素が衝突を検知するならば，瞬時に，他の送信する構成要素もこれを検知すべきである.』

ここで注意すべきことは，検証に無関係のものを捨て去り，仕様を単純化して，モデルを単純にすることである．

2. 次に，上記で明確にされた仕様をもとに，5 回ほど洗練して，変数，状態や並列構造を確定して，9 個の並列動作する時間オートマトンを作成した．また，プロトコルの正しさのルールを時相論理式で記述して検証性質とした．

3.3 仕様記述と検証の例

以上のモデル化により得られた並列動作する 9 個の時間オートマトンは，バス (**Bus** の 1 個の時間オートマトン)，送信する構成要素 **A** (**Detector A**,**Sender A**, **Observer A**, **Frame Generator A** の 4 個の時間オートマトン)，送信する構成要素 **B**(**Detector B**, **Sender B**, **Observer B**, **Frame Generator B** の 4 個の時間オートマトン) であり，システムの構成を以下の図 3.1 に示す．ここで，長方形の中の A_Pn などは変数を表す．なお，**A** と **B** は対称である．Audio/Video プロトコルは **Sender A** と **Detector A** によりモデル化されて，**Sender A** はフレームの伝送をつかさどり，**Detector A** は衝突検知のアルゴリズムをつかさどる．これらの並列動作する時間オートマトンは，入力アクションと出力アクションとの同期 (これは制御の通信を意味する) 及び共有変数 (これはデータの通信を意味する) によって通信する．

以下に，各時間オートマトンについて説明する．

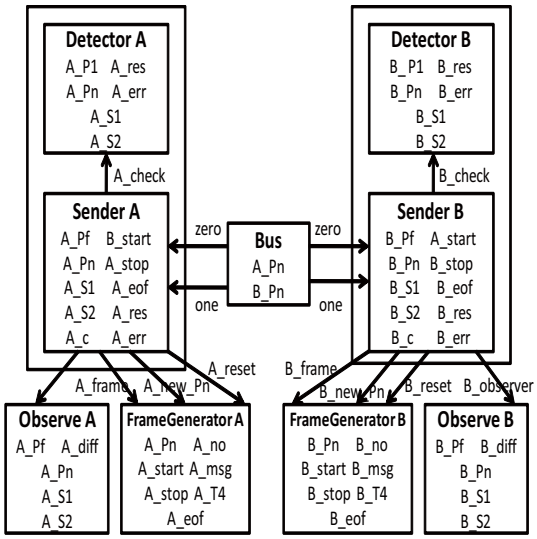

図 3.1 システムの構成図

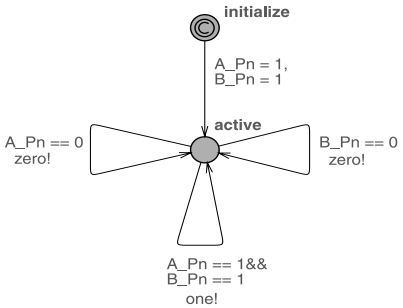

図 3.2 時間オートマトン Bus

まず，時間オートマトン **Bus** を図 3.2 に示す．ここで，変数 A_Pn と B_Pn は 1 ビットのバスのレジスタを表しており，**Sender A** がバスからのデータを読み出しする時点 (W ポイント) で，0 か 1 に設定される．一方，**Sender A** はチャネル zero と one と同期することにより，バスのこの値をサンプルする．ここで，◎または○の中に C のあるロケーションは，そのロケーションに遷移すると，瞬時に優先的に他のロケーションへ遷移することを意味する．

次に，時間オートマトン **Frame Generator A** を図 3.3 に示す．**Frame Generator A** は **Sender A** によってトリガーをかけられたときに (A_new_Pn? を入力したときに)，バス (変数 A_Pn) に 0 か 1 を割り当てて，フレームを生成する．また，**Sender B** が

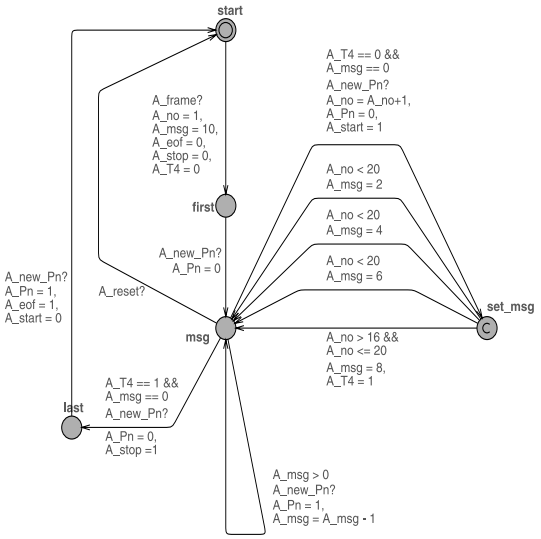

図 3.3 時間オートマトン Frame Generator A

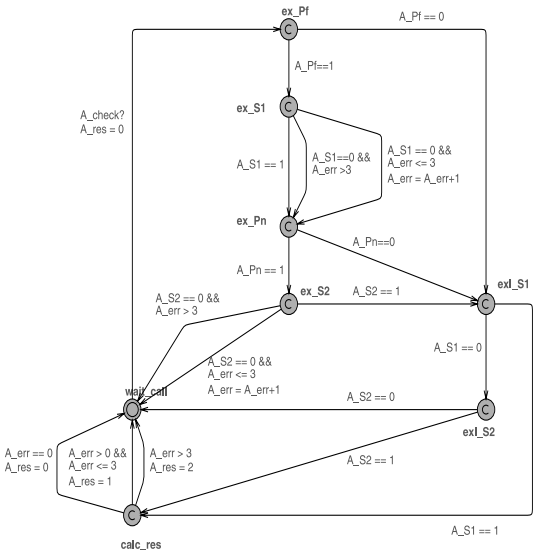

図 3.4 時間オートマトン Detector A

新しいフレームを伝送する前提条件として，**Sender A** がメッセージを出力していないことであるので，**Sender B** はバスをサンプルする．これを直接仕様記述するのは複雑であるので，以下のような抽象化を行う:

『**Sender A** がメッセージを伝送するときに，変数 A_start をセットして，メッセージ伝送後に，A_start をリセットする．これにより，**Sender B** は A_start

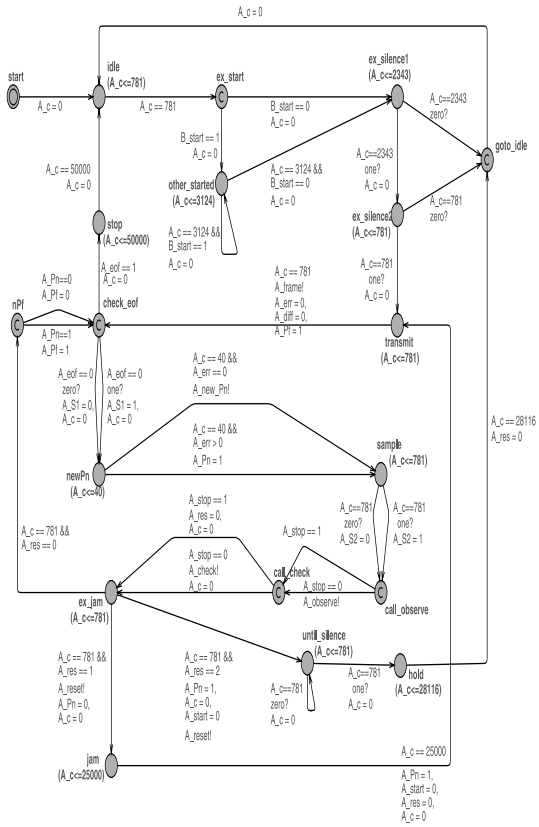

図 3.5 時間オートマトン Sender A

を読むことによって，**Sender A** がメッセージ伝送中かどうかを知る.』

次に，時間オートマトン **Detector A** を図 3.4 に示す．**Detector A** は衝突検知のアルゴリズムであり，**Sender A** からの A_check! により呼び出される．2 つのサンプリングポイントの値 (A_S1, A_S2) が 2 つの出力された値 (A_Pn, A_Pf) と等しく (A_Pf=A_S1 ∧ A_Pn=A_S2) なければ，衝突と見なす．

次に，時間オートマトン **Sender A** を図 3.5 に示す．原論文 [20] には分岐漏れの誤りがあったが，K.G. Larsen 教授らが迅速に正しい時間オートマトンのファイルを送ってくれた [21]．時間オートマトン **Sender A** は **Bus** へ出力するのをトリガーするものであり，(1) 初期フェーズ，(2) 伝送フェーズ，(3) 衝突応答フェーズに分類される．(1) 初期フェーズでは，**Sender B** がフレーム送信中であれば **Sender A** が待つことを表現している．(2) 伝送フェーズで

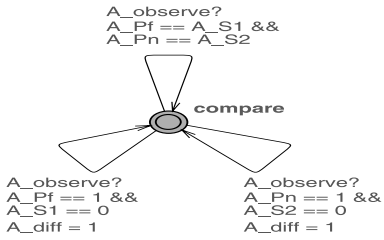

図 3.6　時間オートマトン Observer A

は，伝送中に A_Pf, A_S1, A_Pn, A_S2 を設定したり，**Detector A** の衝突検知の結果でジャム動作を行ったりする．(3) 衝突応答フェーズでは，**Sender A** が衝突検知したときは 25ms の間ジャム動作を行い，**Sender B** がジャム動作を行っているときは 18 周期後に (2) 伝送フェーズに移る．

次に，時間オートマトン **Observer A** を図 3.6 に示す．**Observer A** は衝突が発生したら変数 A_diff をオンして，**Sender A** と **Sender B** が衝突を検知できるようにするものである．

最後に，検証性質を時相論理式で記述する．

$A[\,]$(A_eof==1 imply (A_diff==0 and B_res==0))

ここで，A_eof==1 は **A** のフレームが送信されたことを表し，A_diff==0 は **A** のフレームが衝突していないことを表し，B_res==0 は **B** が衝突を見つけないことを表す．UPPAAL により，以上の 9 個の時間オートマトンが検証性質を満たすことが検証できる．

また，正しいモデルでは，以下の検証性質は成り立たない．

$A[\,]$(A_eof==1 imply (A_diff==1 and B_res==0))

この場合，UPPAAL の反例出力オプション Diagnotic Trace をオンにして検証すると，反例をシミュレーションの画面で確認できる．

さらに，検証式で表現できない性質についても，M.V. Vardi らのアイディア [22] により，テストオートマトンを併用すれば検証可能となる．

なお，著者のホームページ [23] に，本事例の xml ファイル及びプレゼンテーション資料を公開している．

4　おわりに

本稿では，リアルタイムシステムの形式的手法とその検証ツール UPPAAL を概観して，UPPAAL による検証事例を紹介した．実数値のクロック変数の導入により，時間オートマトンは簡潔な仕様記述を実現できる有望な言語である．また，UPPAAL は優れた GUI を備えており，時間オートマトンのモデル検査を実現する検証ツールであり，今後，産業界で大いに使われることが予想される．

これら以外の重要なリアルタイムシステムの形式的手法とその検証ツールとしては，(1) スタンフォード大学の Z. Manna らの時間オートマトンの演繹的検証ツール STeP [24]，(2)T.A. Heinzinger らのハイブリッドオートマトンの記号モデル検査ツール HyTech [25]，(3) ウップサラ大学の Wang Yi らの時間オートマトンを用いたスケジューラビリティ検証器 TIMES [26]，(4) バーミンガム大学 (現在，オックスフォード大学) の M. Kwiatkowska らの CTMC(連続時間マルコフ連鎖)，DTMC(離散時間マルコフ連鎖)，MDP(マルコフ決定プロセス) のモデル検査器 PRISM [27] などがある．組込みシステム開発や最近のプロトコル開発には，スケジューリング解析や確率の導入が重要であり，今後，TIMES [26] と PRISM [27] が機能拡張されて，適用されることが期待される．なお，UPPAAL のサイト [19] から，TIMES [26] を含む UPPAAL の関連するサブプロジェクトをたどることができる．

謝辞　本稿の執筆機会をくださり，貴重な助言をくださった，国立情報学研究所の中島 震 教授に感謝いたします．また，UPPAAL に関する，著者の問い合わせに快く応答くださった，BRICS の K.G. Larsen 教授らに感謝いたします．

参 考 文 献

[1] Lewis, H. R.: *Finite-State Analysis of Asynchronous Circuits with Bounded Temporal Uncertainty*, Harvard University TR-15-89,1989. (Lewis, H. R.: *A Logic of Concrete Time Intervals*, LICS, 1990, pp. 380–389.)

[2] Dill, D. L.: *Timing Assumptions and Verification of Finite-State Concurrent Systems*, LNCS 407, 1989, pp. 197–212.

[3] Alur, R. and Dill, D. L.: *Automata For Modeling Real-Time Systems*, LNCS 443, 1990, pp. 322–335.

[4] Alur, R., Courcoubetis, C. and Dill, D. L.: *Model-Checking for Real-Time Systems*, LICS, 1990, pp. 414–425.
[5] Cerans, K.: *Decidability of Bisimulation Equivalences for Parallel Timer Processes*, LNCS 663, 1992, pp. 302–315.
[6] Henzinger, T. A., Nicollin, X., Sifakis, J. and Yovine, S.: *Symbolic Model Checking for Real-time Systems*, LICS 1992, 1992, pp. 394–406.
[7] Daws, C., Olivero, A., Tripakis, S. and Yovine, S.: *The tool Kronos*, LNCS 1066, 1996, pp. 208–219.
[8] Dill, D. L. and Wong-Toi, H.: *Verification of Real-Time Systems by Successive Over and Under Approximation*, LNCS 939, 1995, pp. 409–422.
[9] Balarin, F.: *Approximate reachability analysis of timed automata*, RTSS 1996, 1996, pp. 52–61.
[10] Yamane, S. and Nakamura, K.: *Symbolic Model-Checking Method Based on Approximations and BDDs for Real-Time Systems*, LNCS 1281, 1997, pp. 562–582.
[11] Asarin, E., Bozga, M., Kerbrat, A., Maler, O., Pnueli, A. and Rasse, A.: *Data-Structures for the Verification of Timed Automata*, LNCS 1202, 1997, pp. 346–360.
[12] Wang, F.: *Efficient Data Structure for Fully Symbolic Verification of Real-Time Software Systems*, LNCS 1785, 2000, pp. 157–171.
[13] Jensen, H. E., Larsen, K. G. and Skou, A.: *Scaling up Uppaal — Automatic Verificationof Real-Time Systems using Compositionality and Abstraction*, LNCS 1926, 2000, pp. 19–30.
[14] Larsen, K. G., Larsson, F., Pettersson, P. and Yi, W.: Compact Data Structure and State-Space Reduction for Model-Checking Real-Time Systems, *J. of Real-Time Systems*, Vol. 25, Kluwer Academic Publisher, 2003, pp. 255–275.
[15] Moller, M. O., RueB, H. and Sorea, M.: Predicate Abstraction for Dense Real-Time System, *ENTCS* 65(6), 2002.
[16] Bengtsson, J., Larsen, K. G., Larsson, F., Pettersson, P. and Yi, W.: *UPPAAL — a Tool Suite for Automatic Verification of Real-Time Systems*, LNCS 1066, 1995, pp. 232–243.
[17] Bengtsson, J. and Yi, W.: *Timed Automata: Semantics, Algorithms and Tools*. LNCS 3098, 2003, pp. 87–124.
[18] Wang, F.: Formal verification of timed systems: a survey and perspective, in *Proceedings of the IEEE*, Vol. 92, No. 8, 2004, pp. 1283–1305.
[19] UPPAAL: http://www.uppaal.com/.
[20] Havelund, K., Skou, A., Larsen, K. G. and Lund, K.: *Formal modeling and analysis of an audio/video protocol: an industrial case study using UPPAAL*, RTSS 1997, 1997, pp. 2–13.
[21] Skou, A. and Larsen, K. G.: Personal communications.
[22] Vardi, M. Y. and Wolper, P.: *An Automata-Theoretic Approach to Automatic Program Verification*, LICS 1986, 1986, pp. 332-344.
[23] 山根智: チュートリアル：リアルタイムシステムの仕様記述と検証, 組込みシンポジウム 2007. (http://csl.ec.t.kanazawa-u.ac.jp/)
[24] Bjorner, N., Manna, Z., Sipma, H. and Uribe, T. E.: *Deductive Verification of Real-Time Systems Using STeP*, LNCS 1231, 1997, pp. 22–43.
[25] Henzinger, T. A., Ho, P. H. and Wong-Toi, H.: HyTech: A model checker for hybrid systems, Software Tools for Technology Transfer 1, 1997, pp. 110–122.
[26] Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P. and Yi, W.: *TIMES — A Tool for Modelling and Implementation of Embedded Systems*, LNCS 2280, 2002, pp. 460–464.
[27] Kwiatkowska, M., Norman, G. and Parker, D.: *PRISM: Probabilistic Symbolic Model Checker*, LNCS 2324, 2002, pp. 200–204.