第17回　回路とシステム
軽井沢ワークショップ
The 17th Workshop on Circuits and Systems
in Karuizawa, April 26, 27, 2004

記号モデル検査の並行ソフトウェアシステムへの応用
Symbolic Model Checking Concurrent Software Systems

土屋達弘 †　　菊野亨 †
† 大阪大学 大学院情報科学研究科

Tatsuhiro TSUCHIYA†　　Tohru KIKUNO†
†Osaka University

1　はじめに

モデル検査とは，状態機械でモデル化されたシステムを対象に，その状態空間を探索することで，与えられた性質が満たされるか否かを判定する検証手法である．従来の形式的検証手法は実用的な観点から疑問が呈されることが多かったが，モデル検査は完全自動化が可能なことに加え，様々な手法の発達により扱うことのできる状態数が飛躍的に増加したために，その有効性が特に期待されている．

それらの手法の内，最も重要なものの一つが記号モデル検査 (symbolic model checking) である [17]. 記号モデル検査では，状態集合や状態間の遷移を数式で記号的に表し，それら数式上の処理によって状態探索を実現する．一つ一つの状態ではなく，数式が表す状態集合を単位として探索を行うため，莫大な状態を対象とする場合でも，それを表す数式が簡潔であるならば，非常に高速な探索が可能となる．

元来，記号モデル検査という言葉は，二分決定グラフ (BDD) によって表されたブール式の操作によって検証を行う手法を指すものであった．論理演算を行う高速なアルゴリズムの存在と，デジタル回路の動作が極めて小さい二分決定グラフにより表現できる場合が多いことから，この手法は特にハードウェアシステムの検証に対し広く用いられており，開発者等は，1998 年度に ACM より Paris Kanellakis Theory and Practice Award を受賞している．近年では，二分決定グラフを操作して検証を行うのではなく，一つのブール式を導出して，その充足性判定問題を解くことで，検証を行う手法も普及しつつある．

このように，記号モデル検査はハードウェアの検証には成功を収めて来ているが，ソフトウェアシステムへの適用はまだ試行の段階と言える．本稿では，記号モデル検査手法のソフトウェアシステムへの適用の試みについて，著者等の取り組みと併せて概説する．

2　モデル検査と記号表現

2.1　モデルと検証する性質

例として，以下の並行プロセス P_0, P_1 からなる簡単な相互排除プログラムを考える．

$$
\begin{array}{lll}
P_0 :: & 0: & \textbf{while } \mathit{True} \ \{ \\
& 1: & \quad \textbf{wait } (t = 0); \\
& 2: & \quad t = 1; \}
\end{array}
$$

$$
\begin{array}{lll}
P_1 :: & 0: & \textbf{while } \mathit{True} \ \{ \\
& 1: & \quad \textbf{wait } (t = 1); \\
& 2: & \quad t = 0; \}
\end{array}
$$

各プロセス $P_i (i = 0, 1)$ がどの行を実行しているかを $pc_i (0 \leq pc_i \leq 2)$ で表すと，$pc_i = 2$ の場合が危険領域である．相互排除は，共有変数 t と，その値に対するビジーウェイトによって実現されている．また，初期状態において t の値は 0 か 1 のいずれかであるが不定と仮定する．この時，この並行システムの状態は，(pc_0, pc_1, t) の値の組合せで表現できるので，その動作は図 1 の状態遷移図によって表される．

(狭義の) モデル検査では，検証すべき性質は時相論理 (temporal logic) によって記述する．ここでは時相論理の詳しい説明は行わないが，良く知られた時相論理である計算木論理 (CTL) を用いて，安全性と活性を表した場合を示す．

- **AG**p　常に p が成り立つ（安全性）
- **AF**p　いずれ必ず p が成り立つ （活性）

ここで，**AG**, **AF** は時相演算子と呼ばれる特殊な演算子である．なお，p 自体が計算木論理式であってもよい．

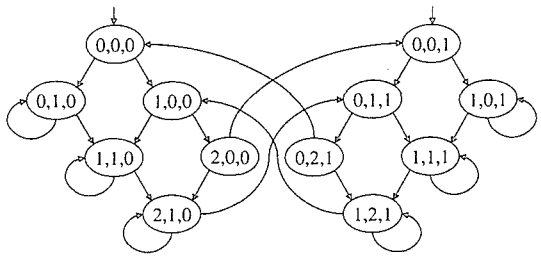

図 1: 状態遷移図.

モデル検査とは，時相論理で記述された性質が初期状態で成立するか否かを判定する問題である．検証が可能な時相論理のクラスに応じて，モデル検査問題の計算量も異なってくる．例えば，計算木論理の場合，与えられた状態遷移図に対し，その状態と遷移の総数に比例した時間で判定が可能である [8].

相互排除が満たされるかは $\neg(pc_0 = 2 \wedge pc_1 = 2)$ が常に成立するか，つまり，$\mathbf{AG}\neg(pc_0 = 2 \wedge pc_1 = 2)$ を判定すればよく，この場合成り立つことがいえる．

また，スタベーションが起こらない，すなわち，wait 文において永遠に待たされることはなく，いずれ危険領域を実行できるという性質は，$\mathbf{AG}((pc_0 = 1 \rightarrow \mathbf{AF}pc_0 = 2) \wedge (pc_1 = 1 \rightarrow \mathbf{AF}pc_1 = 2))$ と表すことができる．図 1 のグラフを探索することで，この性質は満たされないことを自動的に示すことができる．これは，危険領域に入る権限を持たないプロセスが，ビジーウェイトを実行し続ける可能性に対応している．

2.2 記号表現

先に，状態遷移図を明示的に扱った場合，その大きさに比例した時間でモデル検査問題が解けることを述べた．しかし，並行システムの様に，システムを構成する要素が複数存在し独立に動作する場合，システム全体での状態数が莫大になり，単に状態遷移図を求めることすらも困難になることが普通である．このことは通常，状態爆発問題 (state explosion problem) と呼ばれる．記号モデル検査は，この問題を，状態や遷移の一つ一つを明示的に区別して扱うのではなく，状態や遷移を数式で記号的に表現し，それらを集合として取り扱うことによって，解決することを意図したものである．ここでは，状態集合や遷移関係に対する記号的表現について説明する．

まず，状態の任意の集合 S は，以下の様に変数上の 2 値関数 $S(s)$ で表現できる．

$$S(s) = \begin{cases} \text{真} & s \in S \\ \text{偽} & s \notin S \end{cases}$$

ここで s は変数のベクトルであり，それらへの付値それぞれが状態を表す．

例えば，上記の例では $s \equiv (pc_0, pc_1, t)$ であり，初期状態は $(0,0,0)$ もしくは $(0,0,1)$ であるから，初期状態の集合は $I(s) \equiv pc_0 = 0 \wedge pc_1 = 0$ と表現できる．同様に相互排除が成り立つ状態すべての集合は (到達不可能な状態も含めて) $M(s) \equiv \neg(pc_0 = 2 \wedge pc_1 = 2)$ と表される．

遷移関係も遷移の集合であるので，同様に 2 値関数として表現できる．具体的には，$s' \equiv (pc'_0, pc'_1, t')$ を次状態における s として，s から s' への遷移が存在する場合，かつその場合のみ真となる s, s' 上の 2 値関数 $T(s, s')$ によって表す．重要なことは，この遷移関係関数は，システムの記述から図 1 の様な状態遷移図を生成することなく，直接に得ることが可能なことである．例の場合であれば，遷移関係関数は以下の様に表現できる．

$$T(s, s') \equiv \bigvee_{1 \le i \le n} T_i(s, s')$$

ただし，$T_i (1 \le i \le n = 8)$ は以下の通りとする．

$$\begin{aligned}
T_1(s,s') \equiv\ & (pc_0 = 0) \wedge (pc'_0 = 1) \wedge (pc'_1 = pc_1) \\
& \wedge (t' = t) \\
T_2(s,s') \equiv\ & (pc_0 = 1) \wedge (pc'_0 = 1) \wedge (pc'_1 = pc_1) \\
& \wedge (t = 1) \wedge (t' = t) \\
T_3(s,s') \equiv\ & (pc_0 = 1) \wedge (pc'_0 = 2) \wedge (pc'_1 = pc_1) \\
& \wedge (t = 0) \wedge (t' = t) \\
T_4(s,s') \equiv\ & (pc_0 = 2) \wedge (pc'_0 = 0) \wedge (pc'_1 = pc_1) \\
& \wedge (t' = 1) \\
T_5(s,s') \equiv\ & (pc_1 = 0) \wedge (pc'_1 = 1) \wedge (pc'_0 = pc_0) \\
& \wedge (t' = t) \\
T_6(s,s') \equiv\ & (pc_1 = 1) \wedge (pc'_1 = 1) \wedge (pc'_0 = pc_0) \\
& \wedge (t = 0) \wedge (t' = t) \\
T_7(s,s') \equiv\ & (pc_1 = 1) \wedge (pc'_1 = 2) \wedge (pc'_0 = pc_0) \\
& \wedge (t = 1) \wedge (t' = t) \\
T_8(s,s') \equiv\ & (pc_1 = 2) \wedge (pc'_1 = 0) \wedge (pc'_0 = pc_0) \\
& \wedge (t' = 0)
\end{aligned}$$

状態一つ一つを明示的に扱うのではなく，このような数式を操作することで，記号モデル検査では検証を行う．

3 二分決定グラフに基づく記号モデル検査

3.1 概要

最も一般的な記号モデル検査手法では，状態集合や遷移関係を二分決定グラフ (BDD) を用いてブール式によって表現，操作することで検証を行う．ここでは簡単に，この手法の基盤となる逆像計算 (preimage computation, inverse image computation) を用いて，$\mathbf{AG}p$ をどのように検査するかについて述べる．(詳細については優れた解説である [11] を参照されたい．) 逆像計算は，遷移関係関数 $T(s, s')$ と，状態集合 S を表す関数 $S(s)$ から，S に属する状態に 1 回の遷移で到達することができる状態集合をブール関数として次の計算によって求める手続きである．

$$\exists s'.(T(s, s') \wedge S(s'))$$

p が成り立つ状態の集合が関数 $p(s)$ として求められている時，$\mathbf{AG}p$ の検証は次のような手順で可能となる．まず，$\neg p(s)$ の逆像を上記の計算にて求める．ここで，得られた逆像に対するブール関数と $\neg p(s)$ の論理和をとると，0 回もしくは 1 回の遷移により $\neg p$ が成り立つ状態に到達する可能性のある状態集合が得られる．次に，この集合に対し再度逆像計算を行うことで，2 回以下の遷移で $\neg p$ が成り立つ状態に到達する状態集合が得られる．この計算を繰り返すことで，最終的に $\neg p$ が成り立つ状態に到達する可能性のあるすべての状態の集合が得られる．この得られた集合の中に初期状態が含まれていなければ，$\neg p$ が成り立つ状態には到達しないこと，すなわち，$\mathbf{AG}p$ が成り立つことが結論付けられる．

上記の手続きで必要な BDD 上の演算アルゴリズムは [4] によって提案されている．この手法を実装したツールとしては，SMV ファミリー (SMV, NuSMV, Cadence SMV) が著名である．図 2 に SMV の入力言語で，例の並行プログラムを記述したものを示す．図 3 は，2.1 で述べたスタベーションに対する検証の結果，SMV が出力した反例であり，状態 (0, 1, 0) における自己ループの存在を示している

3.2 並行システムの特性を利用した効率化

二分決定グラフによって遷移関係関数を表す場合，特にデジタル回路等については極めて小さい二分決定グラフが得られる場合が多いことから，上記の手法は主にハードウェアシステムの検証に広く用いられてきた．一方，並行ソフトウェアシステムの検証に関しては，並行ソフトウェア特有の性質を利用した

```
MODULE main
VAR
  pc_0 : {0, 1, 2};
  pc_1 : {0, 1, 2};
  t : {0, 1};
DEFINE
  T_1 := (pc_0 = 0) & (next(pc_0)=1)
    & (next(pc_1) = pc_1) & (next(t) = t);
  T_2 := (pc_0 = 1) & (next(pc_0)=1)
    & (next(pc_1) = pc_1) & (t = 1) & (next(t) = t);
  T_3 := (pc_0 = 1) & (next(pc_0)=2)
    & (next(pc_1) = pc_1) & (t = 0) & (next(t) = t);
  T_4 := (pc_0 = 2) & (next(pc_0)=0)
    & (next(pc_1) = pc_1) & (next(t) = 1);
  T_5 := (pc_1 = 0) & (next(pc_1)=1)
    & (next(pc_0) = pc_0) & (next(t) = t);
  T_6 := (pc_1 = 1) & (next(pc_1)=1)
    & (next(pc_0) = pc_0) & (t = 0) & (next(t) = t);
  T_7 := (pc_1 = 1) & (next(pc_1)=2)
    & (next(pc_0) = pc_0) & (t = 1) & (next(t) = t);
  T_8 := (pc_1 = 2) & (next(pc_1)=0)
    & (next(pc_0) = pc_0) & (next(t) = 0) ;
INIT
  pc_0 = 0 & pc_1 = 0
TRANS
  T_1 | T_2 | T_3 | T_4 | T_5 | T_6 | T_7 | T_8
SPEC
  AG ((pc_0=1 -> AF pc_0=2) & (pc_1=1 -> AF pc_1=2))
```

図 2: SMV プログラム

```
state 1.1:
T_8 = 0
T_7 = 0
T_6 = 0
T_5 = 0
T_4 = 0
T_3 = 0
T_2 = 0
T_1 = 0
pc_0 = 0
pc_1 = 0
t = 1

state 1.2:
pc_1 = 1

state 1.3:
pc_1 = 2

state 1.4:
pc_1 = 0
t = 0

-- loop starts here --
state 1.5:
pc_1 = 1

state 1.6:
```

図 3: SMV による反例.

検証の効率化が試みられてきている．例えば，並行ソフトウェアの場合，前述の例の様に遷移関係関数が，並行実行可能な各アクションを表す関数T_iの論理和$\bigvee_i T_i$となる場合が多い．この時，逆像計算をTではなく，T_iそれぞれに対して行い，最終的に結果の論理和をとることで処理を分割する方法がある．具体的には，以下の手順で逆像計算を行う．

$$\begin{aligned} & \exists s'.(T(s,s') \vee S(s)) \\ = \; & \exists s'.(T_1(s,s') \vee T_2(s,s') \vee \cdots \vee T_n(s,s') \vee S(s')) \\ = \; & \exists s'.(T_1(s,s') \wedge S(s')) \vee \exists s'.(T_2(s,s') \wedge S(s')) \\ & \vee \cdots \vee \exists s'.(T_n(s,s') \wedge S(s')) \end{aligned}$$

この手法は各T_iに対し小さい二分決定グラフの表現が得られる場合，極めて有効な手法である．

また，記号的な手法を用いないモデル検査アルゴリズムでは，プロセスの並行性を考慮することで検証に不要な状態を探索せず効率化を図るpartial order reductionと呼ばれる手法や，似通ったプロセスが複数存在した場合に，状態の対称性を利用して状態削減を行う手法が良く研究されている．これらの手法は，それぞれ例えばSPIN[12]やSMC[19]といったモデル検査ツールで実装されており，通信プロトコル等の検証で大きな成果を挙げている．二分決定グラフを用いた記号モデル検査においても，これらの手法を組み合わせることで，ある程度，検証効率を高めることが可能なことが示されている[1, 13, 18].

3.3 適用例

3.3.1 航空機搭載電子システム

文献[6, 7]では，航空機搭載電子システムである衝突防止システムTCAS II，及び，電力分配システムEPDに対する，記号モデル検査の適用に関して報告している．この二つシステムは，共に，並行性を有する大規模なソフトウェアシステムであり，ステートチャートによって記述された厳密な設計仕様が存在する．記号モデル検査の適用は，SMVを用いてステートチャート特有の階層構造，並行性，遷移を表現することによって実現されている．また，外部イベントの排他性を利用して遷移関係関数を論理和の形に分割し，3.2で述べた効率化を図るといった，様々な最適化手法も提案されている．非決定的遷移の有無や，各種機能に関し検証を行った結果，シミュレーションでは発見できなかった幾つかの誤りの検出に成功している．

3.3.2 鉄道連動装置

鉄道連動装置(interlocking system)とは，鉄道における信号機器を効率的に制御し，列車の安全を確保する信号保安装置のことである．鉄道連動装置は検証の対象として頻繁に取り上げられるシステムであり，連動図等によって記述された厳密な仕様が存在するという特徴を持つ．記号モデル検査の適用に関しては，例えば，文献[9]では，離散時間の概念を明示的に扱うことの可能なモデル検査ツールVerusによる検証について報告がなされている．また文献[14]では，模倣関係を利用した場合分けを行い，必要なシナリオのみに対しNuSMVを用いた検証を行っている．

3.3.3 自己安定アルゴリズム

自己安定アルゴリズムは，どのようなシステム状態からでも正常な状態に回復することが可能な分散アルゴリズムである．このような性質は自己安定性と呼ばれ，耐故障性の面から注目されている．文献[20]では，知られている幾つかの自己安定アルゴリズムを，SMVを用いて，それらが実際に自己安定性を満たしているかを検証している．自己安定性の検証で問題になるのが，任意の状態について正常状態への復帰を確認しなければならない点であり，状態数が膨大になるため明示的に状態を考慮することが困難である．しかし，記号的表現を用いた場合，初期状態の集合は，単に恒真であるようなブール関数として表現できるので，容易に検証が可能となる．検証の結果，あるアルゴリズムにおいて，正常状態への到達に関して活性が満たされない可能性があることが分かった．

4 限定モデル検査 (Bounded Model Checking)

4.1 概要

ブール式の充足可能性判定の技術が目覚しい進歩を遂げているのに呼応して，近年従来の二分決定グラフを用いず，充足可能性判定によって検証を行う手法が注目されている．この手法は限定モデル検査(有界モデル検査，bounded model checking)と呼ばれ[3]，その概念は二分決定グラフを用いる場合よりもむしろ簡単である．

もっとも単純な$\mathbf{AG}p$の場合の検証の場合であれば，以下の式の充足可能性判定を行うことで検証が

可能となる．

$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigvee_{0 \le i \le k} \neg p(s_i)$$

ここで，最後の $\neg p(s_i)$ の論理和を除いた部分が真になる必要充分条件は，s_0 が初期状態であり，かつ全ての $0 \le i \le k-1$ に対して，状態 s_i から状態 s_{i+1} へ遷移可能であることである．また $\bigvee_{0 \le i \le k} \neg p(s_i)$ は $s_0, \cdots, s_k$ のいずれかの状態で p が成り立たない場合かつその場合のみ真となる．従って，式全体が充足する必要充分条件は，初期状態から k 回以下の遷移で p が成り立たない状態に到達することである．従って，充足可能であれば，$\mathbf{AG}p$ は成り立たないことが結論できる．しかし，充足不能の場合，初期状態から $k+1$ 回以上の遷移によって p が成り立たない状態に到達する可能性は排除できない．限定モデル検査という呼称は，このように限定された遷移回数内で反例を探すことに由来する．

4.2 並行性の活用

限定モデル検査によって，従来の二分決定グラフを用いる方法に比べて大幅に検証効率を改善できる場合があることが報告されている．しかし，並行プログラムのような非同期システムでは，遷移関係関数 T の簡潔な表現が得られず検証時間が大きくなるため，限定モデル検査の有用性が殆ど利用できない．この問題を解決するため，著者等は新しい動作表現について研究している．

先に述べたように，並行ソフトウェアの場合，全体の遷移関係は，並行実行可能なアクションそれぞれの動作によって規定される遷移の和として表現される．このことを利用して，以下の $\hat{T}(s_0, s_1, \cdots, s_n)$ によって遷移を表現し，遷移関係関数 $T(s, s')$ の代わりに用いることで，検証の効率化を達成することができる．

$$\begin{aligned} \hat{T}(s_0, \cdots, s_n) \equiv\ & (T_1(s_0, s_1) \vee (s_0 = s_1)) \\ & \wedge (T_2(s_1, s_2) \vee (s_1 = s_2)) \\ & \wedge \cdots \\ & \wedge (T_n(s_{n-1}, s_n) \vee (s_{n-1} = s_n)) \end{aligned}$$

この時，$\hat{T}(s_0, \cdots, s_n)$ が真となる必要充分条件は，各 $i (0 \le i < n)$ について，$s_i = s_{i+1}$ もしくは s_i から s_{i+1} へ遷移が存在することである．$\hat{T}(s_0, \cdots, s_n)$ を用いて以下の式を構成する．

$$\begin{aligned} & I(s_0) \wedge \hat{T}(s_0, \cdots, s_n) \wedge \hat{T}(s_n, \cdots, s_{2*n}) \wedge \\ & \cdots \wedge \hat{T}(s_{(k-1)*n}, \cdots, s_{k*n}) \wedge \neg p(s_{k*n}) \end{aligned}$$

この式の充足可能性を判定することで $\mathbf{AG}p$ の検証が可能となる．式が充足可能ならば，p が成り立たない状態に初期状態から $k*n$ 回以下の遷移で到達する可能性があり，充足不能ならば，p が成り立たない状態には k 回以下の遷移では到達できないことが言える．つまり，この手法は，p を満たさない状態に k 回以下の遷移で到達する可能性があれば，その可能性を必ず検出するのと同時に，k より多くの遷移回数が必要な場合でも，それが $k*n$ 回以下であれば検出できる場合があるという特徴を持っている．

4.3 適用例

前述した理由により，限定モデル検査を並行ソフトウェアの検証へ用いることは困難であり，適用例は多くない．著者等は，上記の手法を電話システムの仕様における機能競合検出に適用し，競合が存在する場合は，明示的な状態探索や二分決定グラフを用いた記号モデル検査に比べ，極めて高速な検出が可能であることを示した [21]．

逐次プログラムが対象であるが，文献 [15] では，C プログラムの動作を記号表現を用いて表し，限定モデル検査を行うことで，仕様との整合性を判定する手法が提案されている．また，この手法を改変し，クロスサイトスプリクティング脆弱性の検出を，限定モデル検査で行う手法が開発されている [16]．

5 無限状態システムの検証

3, 4 節で説明したブール式による記号表現を用いた手法では，変数の値がブール変数で表現できるように，値の取り得る範囲を予め定める必要がある．従って，変数値の範囲が無限である場合，抽象化が避けられないという問題があった．この問題に対し，無限状態を正確に取り扱う様々な手法が研究されている．

例えば，時間オートマトンやその拡張であるハイブリッドオートマトンは，有限状態機械を連続値を取る変数を付加することで拡張したものでり，変数が連続値を取るため，その状態空間は無限個の状態を含む．このようなシステムのモデル検査には，等価な状態の集合を上位レベルにおいて一つの状態としてみなし，システムの動作を有限状態機械として扱うことが必要である．この上位レベルの状態 (region, zone 等と呼ばれる) も当然無限個の状態を含んでおり，システム変数上の制約式を論理結合することによって表す必要があるため，状態集合の記号的な表現という概念が必須となる．また，このようにして得られる有限状態機械 (例えば region graph) を明示

的に生成し検証を行うのではなく，記号的な表現のみを利用してモデル検査を行う手法もよく研究されている [2].

また実数変数ではなく，上下限のない整数変数を持つ並行プログラムを直接検証する手法として，整数の加減・比較演算と論理演算のみを持つ一階述語論理式である Presburger 式を用いる手法が開発されている [5]. Presburger 式によって状態集合，遷移関係を表し，逆像計算を適用することで，変数値を離散化することなくモデル検査を実現することができる.

6 おわりに

本稿では，記号モデル検査手法のソフトウェアシステムへの適用について概説した．抽象度の高いアルゴリズムの段階ではなく，実用レベルの設計へモデル検査を適用するには，本稿で挙げた航空機搭載電子システムと鉄道連動装置の例に典型的に見られるように，モデル化が正確に行えるだけの厳密な仕様が存在することが重要になる.

この問題に対する一つのアプローチとして，Bandera[10] のように，並行プログラムのソースコードから直接モデルを抽出する手法が研究されており，今後一層注目を集めるものと思われる.

参考文献

[1] R. Alur, R. Brayton, T. Henzinger, and S. Qadeer. Partial-order reduction in symbolic state space exploration. In *Proceedings of the Ninth International Conference on Computer-aided Verification (CAV'97), LNCS 1254*, pages 340–351, 1997.

[2] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Engineering*, pages 181–201, 1996.

[3] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), LNCS 1579*, pages 193–207, 1999.

[4] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, Aug 1986.

[5] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Trans. Program. Lang. Syst.*, 21(4):747–789, 1999.

[6] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. D. Reese. Model checking large software specifications. *IEEE Trans. Software Eng.*, pages 498–520, 1998.

[7] W. Chan, R. J. Anderson, P. Beame, D. H. Jones, D. Notkin, and W. E. Warner. Optimizing symbolic model checking for statecharts. *IEEE Trans. Software Eng.*, pages 170–190, 2001.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[9] V. Hartonas-Garmhausen, S. Campos, A. Cimatti, E. Clarke, and F. Giunchiglia. Verification of a safety-critical railway interlocking system with real-time constraints. *Sci. Comput. Program.*, 36(1):53–64, 2000.

[10] J. Hatcliff and M. B. Dwyer. Using the Bandera tool set to model-check properties of concurrent java software. In *Proceedings of CONCUR 2001*, pages 39–58, 2001.

[11] 平石裕実, 浜口清治. 論理関数処理に基づく形式的検証手法. 情報処理, 35(8):710–717, 1994.

[12] G. J. Holzmann. The model checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[13] S. Jha. *Symmetry and Induction in Model Checking*. Ph.D. Dissertation, 1996.

[14] 亀山幸義, 中島一. ケーススタディ：モデル検査と定理証明を用いた鉄道信号制御システムの検証. 「システム検証の科学技術」予稿集, 82–91, 2004.

[15] D. Kroening, E. Clarke, and K. Yorav. Behavioral consistency of C and Verilog programs using bounded model checking. In *Proceedings of DAC 2003*, pages 368–371. ACM Press, 2003.

[16] S.-Y. Kuo, Y.-W. Huang, F. Yu, C. Hang, and D.-T. Lee. Detecting web application vulnerabilities by model checking. In *Proceedings of DSN 2004*. IEEE CS Press, 2004.

[17] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[18] M.Tomisaka and T.Yoneda. Partial order reduction in symbolic state space traversal using zbdds. *IEICE Trans. Information and Systems*, E82-D(3):704–711, 1999.

[19] A. P. Sistla, V. Gyuris, and E. A. Emerson. SMC: A symmetry-based model checker for verification of safety and liveness properties. *ACM Trans. Softw. Eng. Methodol.*, 9(2):133–166, 2000.

[20] T. Tsuchiya, S. Nagano, R. B. Paidi, and T. Kikuno. Symbolic model checking for self-stabilizing algorithms. *IEEE Trans. Parallel and Distributed Systems*, pages 81–95, 2001.

[21] T. Tsuchiya, M. Nakamura, and T. Kikuno. Detecting Feature Interactions in Telecommunication Services with a SAT solver. In *Proc. of 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02)*, pages 131–134, 2002.