情報処理学会論文誌 Vol.61 No.4 853–862 (Apr. 2020)

活性と同時に保証可能な安全性特定のための ゲーム分析アルゴリズム

相澤 和也[1,a)] 鄭 顕志[1,2,b)] 本位田 真一[1,2,c)]

受付日 2019年8月1日, 採録日 2020年1月16日

概要：イベントベースなシステムにおいて，「起こるべき事象がつねにいつかは起こる」という活性や「起こるべきでない事象はけっして起こらない」という安全性が保証可能かは動作環境に依存する．システムの動作環境のモデルと振舞いに対する要求を入力として，要求が保証された仕様を自動合成する離散制御器合成が研究されてきた．離散制御器合成では環境と要求からゲームを構築し，そのゲームに勝利できるような戦略を仕様として出力するが，保証可能な要求が入力されなければ勝利戦略が立てられず仕様は合成できない．このため開発者には保証可能な要求を特定することが求められた．与えられた環境と要求から保証可能な要求を特定する研究は行われているが，対象のドメインを絞ったり扱う要求を安全性のみに限定したうえでの実現であった．そこで本研究では活性と同時に保証可能な安全性を特定するためのゲーム定義と特定のためのアルゴリズムを提案する．また，提案アルゴリズムが活性と同時に保証可能な安全性を特定できることを証明し，そしてアルゴリズムの計算量が構築されるゲーム空間の状態数 m，遷移数 n，扱う安全性の数 r に対して $O(2nmr)$ と現実的な計算時間となりうることを確認した．

キーワード：離散制御器合成，安全性，活性，2 人対戦型ゲーム

A Game Analysis Algorithm for Identifying Safety Properties Guaranteeable While Satisfying a Liveness Property

Kazuya Aizawa[1,a)] Kenji Tei[1,2,b)] Shinichi Honiden[1,2,c)]

Received: August 1, 2019, Accepted: January 16, 2020

Abstract: Requirements of the event-based systems are guaranteed under environmental assumptions. Our target requirements are safety properties, which means “bad things do not happen”, and liveness properties, which means “good things do always happen eventually”. Discrete controller synthesis is one of the technique acquiring the behavior specification which guarantees the given requirements in the given environment. This technique requires developers to identify the requirements guaranteeable in the environment. Several works identified such requirements, but they limit target requirements or target systems. In this paper, we use a two player game and identify guaranteeable safety properties while satisfying a liveness property. We also proposed an algorithm and proved that the algorithm can identify the guaranteeable safety properties. In addition we confirmed that the time complexity of the algorithm is $O(2nmr)$, where n is the number of states, m is the number of transitions and r is the number of treated safety properties in the game.

Keywords: discrete controller synthesis, safety property, liveness property, two-player game

1. はじめに

イベントベースなシステムが要求を保証しながら振る舞えるかは動作環境に依存する．開発者は動作環境に対する仮定の下で要求を保証する振舞いを策定する．要求とアーキテクチャとの間には相互に依存関係があることが指摘され，ツインピークスモデル [1] のように相互を行き来し

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 162–0042, Japan
[2] 国立情報学研究所
National institute of informatics, Chiyoda, Tokyo 101–8430, Japan
a) k.aizawa@aoni.waseda.jp
b) ktei@aoni.waseda.jp
c) honiden@nii.ac.jp

ながら開発することが提唱されている．相互の依存関係とは，要求を満たすためにアーキテクチャを決定する一方で，アーキテクチャの決定もまた満たせる要求の制約につながるということである．したがって要求やアーキテクチャは固定的でなく複数の候補を持ちうる [2]．要求と振舞いとの間にも同様な関係が存在し，仮定の異なる動作環境とそこで満たせる要求の組を複数扱ったディペンダブルなシステムも研究されている [3]．本論文で扱う要求は「起こるべき事象がつねにいつかは起こる」という活性と「起こるべきでない事象はけっして起こらない」という安全性である．これらはイベントベースなシステム開発において扱われる要求である [4], [5], [6], [7], [8]．

ある動作環境下で要求が保証された振舞い仕様を獲得するために離散制御器合成 [7] が研究されてきた．離散制御器合成ではシステムと環境との相互作用のモデルと振舞いに対する要求を入力とし，与えられた環境下で与えられた要求が満たされることを保証する仕様モデルが出力される．これを用いてそれぞれの環境で合成した可能な限りの要求を満たす仕様を実行時に切り替えていくことでディペンダブルなシステムが実現できる [3]．しかし，与えられた環境下で与えられた要求が保証可能でない場合には仕様は合成されない．保証をともなう仕様を獲得するためには動作環境下で満たせる要求を開発者が特定する必要がある．

動作環境下で要求が満たせるかを分析する研究は行われてきた [9], [10], [11], [12] が，保証可能な要求を特定する研究 [11], [12] では対象システムを建物の入退室管理システムに特化 [11] したり，分析する要求の種類を安全性に限定 [12] したりして実現させていた．しかしイベントベースなシステムでは安全性と同様に活性も重要な要求としてあげられる．

本論文では分析できる要求の範囲を拡張することを目的とし，活性と同時に保証可能な安全性を特定するための問題の定式化とアルゴリズムの提案を行う．このアルゴリズムは入力された動作環境と要求から構築されたゲーム上で，制御器がゲームの勝利戦略を持てる領域（以下，勝利領域とよぶ）を安全性ごとに特定する．保証可能な安全性の特定は到達可能性ゲーム [13] に変換されるのに対し，活性の保証可否の判定は Büchi ゲーム [13] に変換される．Büchi ゲームの勝利領域を計算するアルゴリズムでは到達可能性ゲームのアルゴリズムが利用される．我々はこの特徴を利用し，保証可能な安全性特定のアルゴリズム [12] と同様な分析を Büchi ゲームの勝利戦略計算アルゴリズムの過程でも実現する．本論文の貢献点は以下のとおりである．

(1) 活性と同時に保証可能な安全性を特定可能なゲームと求める勝利領域の定式化
(2) 勝利領域を求めるためのアルゴリズムの提案
(3) アルゴリズムにより勝利領域が求まることの証明

また，本論文では提案したアルゴリズムの計算量についても検討し，離散制御器合成を安全性の全組合せに対して総あたりに行った場合の計算量と比較することで本アルゴリズムの計算の効率性について論じる．

本論文の構成は以下のとおりである．2 章で関連研究を説明し，3 章で材料加工工場の制御システムを用いて問題の説明を行う．4 章で背景となる技術について述べた後に 5 章で本論文で扱うゲームと求める勝利領域の定義をする．6 章で勝利領域を求めるアルゴリズムを提案したのち，7 章でアルゴリズムの妥当性を確認し 8 章でまとめを述べる．

2. 関連研究

システムのアーキテクチャや振舞いが満たせる要求の分析は様々な研究が行われてきた．Calinescu ら [10] は複数の準備した仕様から，実行環境中で最も要求を満たしうる仕様モデルに切り替える手法を提案した．彼らは仕様を離散時間マルコフ連鎖としてモデル化し実行環境から得られたパラメータ値に基づいて確率モデル検査を行うことでそれぞれの振舞いの保証可否を確率的に分析した．Abushark ら [14] はエージェントベースなソフトウェアの仕様が要求を満たせない箇所を開発早期に発見する手法を提案した．彼らはユースケースやゴールモデルとして与えられた要求をペトリネットに変換し，同様にペトリネットで表現された仕様モデル上での到達可能性を調べることで満たせない要求を特定した．Cámara ら [9] はアーキテクチャ内での要求違反の発生箇所に対する変更計画を自動生成する手法を提案した．彼らはアーキテクチャの変更動作とその影響をマルコフ決定過程でモデル化し，各状態から得られる報酬が最大となるように遷移を選択するための方針を確率モデル検査を用いて合成した．DeVries ら [15] はシステムに対する要求をゴールモデルによって表し，要求の細分化が不完全な箇所や食い違っている箇所を実行時に発見する手法を提案した．彼らは設計時に構築されるゴールモデルの正しさは環境に対する仮定に依存することを指摘し，実行時に各要求の充足に食い違いが起きないかを効用関数を用いて監視した．これらの技術は開発者が作成した仕様が要求を満たすかを分析するものである．しかし仕様を用いるこれらの手法では動作環境下で保証不能な要求が存在することを感知することは難しい．

与えられた動作環境で与えられた要求の保証可否を分析し，振舞い仕様の自動合成を行う研究には次のようなものがある．D'Ippolito ら [7] は与えられた環境下で活性と安全性を保証する制御器を合成するアルゴリズムを提案した．彼らは環境を表す状態遷移モデルと要求を表す時相論理式から環境と制御器との 2 人対戦型ゲームを構築し，制御器の勝利戦略を発見することで離散制御器を合成した．Ciolek ら [16] は安全性と到達可能性を対象として離散制御器を合成するのに必要な計算時間と計算空間の削減を行った．彼らはドメインに非依存なヒューリスティック関数を

構築し最良優先探索を行うことで解の発見を高速化した．これら離散制御器合成では保証できない要求が含まれていた場合には制御器を合成できない旨を通知するが，具体的に保証できない要求の特定までは行わない．Tsigkanos ら [11] は入退室管理システムに特化したモデルを構築し，セキュリティに関する安全性と業務遂行に関わる到達可能性の保証可否を分析し，その結果に基づいてシステムの制御方針を決定する手法を提案した．彼らは建造物内の部屋割りと資産，そして人の振舞いをモデル化し，資産の置かれた部屋に対する要求を計算木論理によって与えることで，その要求が脅かされるような状況の特定と対策を状態遷移モデル上で自動化した．相澤ら [12] は扱う要求を安全性に限定することで状態遷移モデルで表された環境上で，保証可能な安全性を特定する手法を提案した．彼らは安全性に限定することで問題を制御器と環境との間で行われる到達可能性ゲームに変換し，制御器が勝利可能な条件となるように安全性の集合を算出するアルゴリズムを構築した．しかし，安全性と活性を対象として保証可能な要求を特定する手法については，我々の知る限り，これまで考えられてこなかった．本論文では活性と同時に保証可能な安全性を特定する手法を提案することで扱える問題の範囲を広げることを目的とする．

3. 材料加工工場の制御システムによる問題例

本論文の説明にあたっては材料加工工場を題材として扱う．材料加工工場は離散制御器合成 [7] でもあげられるイベントベースなシステムのケーススタディ [17] である．工場にはプレス機やドリル，塗装機など加工のための機械，加工前の材料が置かれたトレイ，加工済み材料を置くためのトレイ，そして材料をトレイや機械に運ぶためのロボットアームが存在する．この工場の制御システムは機械による加工作業およびロボットアームによる材料の移動を制御して加工前の材料をトレイから取り出し，加工を完了したうえで加工済みのトレイに置くことを目的とする．この制御システムに対する活性は「材料が加工済みトレイに置かれる」があげられ，安全性は「加工が完了するまで加工済みトレイに材料が置かれない」，「同じ加工を 2 度行わない」，そして「加工の順番を誤らない」があげられる．ただしこれらの安全性は「機械の加工作業は失敗しない」という仮定がなければ保証できない．たとえば塗装作業が失敗した場合，再び行うか加工の完了を諦めて加工済みトレイに置くかのいずれかを行うことになる．仮定の崩れた環境を事前に複数想定し，それぞれで最大限の要求を保証する仕様に切り替える手法 [3] が存在するが，想定した環境下で保証可能な要求は開発者によって特定される必要があった．

4. 背景技術

本論文ではシステムの動作環境を Labeled Transition System (LTS) で与え，活性と安全性を Fluent Linear Temporal Logic [18] で与える．これは離散制御器合成 [7] や保証可能な安全性の特定 [12] と同様である．

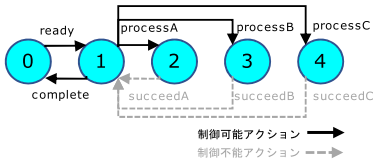


図 1 材料加工工場制御システム環境モデル例
Fig. 1 An example model of the production cell control system.

定義 1. LTS は $E = (S, s_0, A, \Delta)$ で表現される．S は有限の状態集合であり s_0 は初期状態を表す．$A = A_C \cup A_U$ はアクションの集合である．A_C はシステムが制御可能であり，A_U は制御不能である．$\Delta \subseteq S \times A \times S$ は遷移関係を表す．E 上の Δ に従う S と A の列 $t = s_0, a_0, s_1, a_1 \ldots$ をトレースとよび，その集合を T で表す．

図 1 は材料加工工場制御システムの環境モデルの例である．実線は制御可能アクションによる遷移を表し，破線は制御不能アクションによる遷移を表す．ready はロボットアームが加工前の材料を取り出し材料の加工を始められる状態へ遷移する．complete はロボットアームが材料を加工済みトレイに置き加工を終了する．processA/B/C はロボットアームが材料を加工機に移動させ，加工を開始する．succeedA/B/C は各加工の成功通知を受け取り，ロボットアームが加工機から材料を取り出す．ただし，図 1 は加工が必ず成功するという仮定のモデルであり，ロボットアームは加工の失敗通知を受け取る可能性もある．このように環境モデル作成時には仮定に対応する遷移や状態の追加削除が行われる．

FLTL 式は $\phi, \psi ::= fl | \neg\phi | \phi \vee \psi | \mathbf{X}\phi | \phi \mathbf{U} \psi$ で再帰的に定義される．$fl = \langle I_{fl}, T_{fl} \rangle_{initially_{fl}}$ は fluent といい，$E = (S, s_0, A, \Delta)$ 上で判定可能な命題である．ここで $I_{fl}, T_{fl} \subseteq A$ かつ $I_{fl} \cap T_{fl} = \emptyset$ であり E 上のあるトレース t において $a_i \in I_{fl}$ が現れてから $a_t \in T_{fl}$ が現れるまでのあいだ fl は真となる．ただし，$initially_{fl} \in \{true, false\}$ は fl の初期値を表しており $true$ なら fl は t の先頭から $a_t \in T_{fl}$ が現れるまでのあいだも真となる．ϕ を構成する論理作用素は $\neg$ (negation), $\vee$ (or), $\mathbf{X}$ (next), $\mathbf{U}$ (until) である．$\mathbf{X}\phi$ では ϕ が真となった次の状態で真となり，$\phi \mathbf{U} \psi$ では ψ がいつかは真となり，ψ が真になるまで ϕ が真であり続けるとき真となる．また，糖衣構文として $\wedge$ (and), $\Diamond$ (eventually), $\Box$ (always), $\mathbf{W}$ (weak until) も用いる．$\Diamond\phi$ は将来的に ϕ が真となる場合，$\Box\phi$ はつねに ϕ が真である場合に真となる．$\phi \mathbf{W} \psi$ は $\phi \mathbf{U} \psi \vee \Box\phi$ である．FLTL では安全性は $\Box\phi$ で表され，活性は $\Box\Diamond\phi$ によって表される．

材料加工工場制御システムにおける活性と安全性の例を以下に示す．fluent 1 は材料に対する加工作業 A が成功し

た状態を表す命題であり，fluent 2 は材料に対する加工作業 B が成功した状態を表す命題である．fluent 3 は材料を加工済トレイに置き，加工が完了したことを表す命題であり，本システムの活性はこの fluent 3 である．安全性 1 は加工作業 A が完了するまでは加工済トレイに置かれないことを表し，安全性 2 は加工 A は加工 B が終わるまで行わないという加工の順番を定めており，安全性 3 は 1 度加工した材料に同じ加工をしないことを表す．安全性を表す式中のアクション名は当該アクションが発火してから他のアクションが発火するまで真となる fluent を表している．

fluent 1：$fl_A = \langle succeedA, complete\rangle_{false}$

fluent 2：$fl_B = \langle succeedB, complete\rangle_{false}$

fluent 3：$fl_{comp} = \langle complete, ready\rangle_{false}$

活性：$\Box\Diamond fl_{comp}$

安全性 1：$\Box(complete \rightarrow \mathbf{X}(\neg complete \mathbf{W} fl_A))$

安全性 2：$\Box(processA \rightarrow fl_B)$

安全性 3：$\Box(processA \rightarrow \mathbf{X}(\neg processA \mathbf{W} complete)$

保証可能な要求を分析するにあたっては LTS による環境モデルと FLTL による安全性と活性から 2 人対戦型のゲームを構築する [7].

定義 2. 2 人対戦型のゲームは $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, \Phi)$ で表される．S_g は有限の状態集合であり $s_{g0} \in S_g$ は初期状態である．$\Gamma^-, \Gamma^+ \subseteq S_g \times S_g$ は状態遷移関係を表しており，Γ^- は環境によって，Γ^+ はシステムの制御器によって制御される．Φ は勝利条件を表す．s_g から $\Gamma^{-/+}$ で遷移される状態の集合を $\Gamma^{-/+}(s_g) = \{s'_g | (s_g, s'_g) \in \Gamma^{-/+}\}$ と表す．このとき，$\Gamma^-(s_g) \neq \emptyset$ であるような s_g は環境の手番であり，それ以外の状態は制御器の手番である．G 上で $i \geq 0$ かつ $\gamma_i = (s_{gi}, s_{gi+1}) \in \Gamma^- \cup \Gamma^+$ であるような列 $\pi = s_{g0}, \gamma_0, s_{g1}, \gamma_1 \ldots$ をプレイとよび π の集合を Π で表す．環境/制御器の手番 s_{gn} で終わる π は $s_{gn+1} \in \Gamma^{-/+}(s_{gn})$ であるような s_{gn+1} によって拡張できる．また，$Occ : \Pi \rightarrow 2^{S_g}$ は π の中に 1 つでも含まれる s_g の集合を返し，$Inf : \Pi \rightarrow 2^{S_g}$ は π の中に無限回含まれる s_g の集合を返す関数である．π において勝利条件 Φ を満たした方が勝利となる．

定義 2 では勝利条件に数学的定義を与えていない．これは扱うゲームによってその勝利条件の形式が変わるためである．たとえば到達可能性ゲーム [13] であれば勝利条件はゲームの状態 S_g の部分集合となり，本論文で扱う後述の安全性付き Büchi ゲームであれば勝利条件は S_g の部分集合と S_g の部分集合族との組となる．また，ゲームの勝利条件は分析対象となる要求ごとに異なる．安全性のみであれば到達可能性ゲーム [13] に，活性のみであれば Büchi ゲーム [13] の勝利条件にそれぞれ変換可能である．離散制御器合成 [6], [7] ではこれらのゲームにおいて制御器がつねに勝利できる戦略を求めることで制御器を自動合成する．この勝利戦略を持てる状態の集合を勝利領域とよぶ．

定義 3. ある $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, \Phi)$ の s_{g0} を $s_w \in S_g$ に置換したとき，環境の手番での遷移に非依存に制御器が勝利できる遷移を選択する戦略を勝利戦略，勝利戦略を持つ状態 s_w の集合 W_c を勝利領域とよぶ．

勝利領域特定の計算量は状態数に対して安全性では線形であり，活性では二乗のオーダである．計算量の複雑さから保証可能な要求の特定では安全性のみが扱われてきた [12]．本論文では活性と同時に保証可能な安全性を特定する手法を提案する．

5. 活性と同時に保証可能な安全性特定のためのゲームと条件付き勝利領域の定義

本章では活性と同時に保証可能な安全性を特定するためのゲームと保証可能な安全性を特定するための勝利領域を定義する．安全性の分析では，安全性を表す FLTL 式を違反する状態を環境側の勝利条件として持つゲームが構築され，その状態に到達すれば環境の勝利となる．一方で活性の分析では，活性を表す FLTL 式の作用素 “$\Box\Diamond$” が作用する式を満たす状態を勝利条件として持ち，その状態に無限に到達できれば制御器の勝利となる．これより活性と安全性を扱うゲームを以下のように定義する．

定義 4. 2 人対戦型ゲーム $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, \Phi)$ が与えられたとき，勝利条件 Φ が $\lambda \subseteq S_g$ かつ $\Sigma \subseteq 2^{S_g}$ であるような組 (λ, Σ) であるとき，そのようなゲームを安全性付き Büchi ゲームとよぶ．安全性付き Büchi ゲームでは $Inf(\pi) \cap \lambda \neq \emptyset$ かつ，すべての $\sigma \in \Sigma$ に対して $Occ(\pi) \cap \sigma = \emptyset$ であるようなプレイ π は制御器の勝利となり，それ以外は環境の勝利となる．

安全性付き Büchi ゲームの勝利条件 λ は活性に対応し，Σ は安全性に対応する．λ が状態の集合なのに対して Σ が集合族であるのは活性を保証することを前提にしており複数の活性を識別する必要がないのに対して，安全性は保証可否を個々に特定するためである．活性をゲームに変換する過程は離散制御器合成の手法 [6] を用い，安全性をゲームに変換する過程は保証可能な安全性特定のためのゲーム構築手法 [19] を用いる．

活性と安全性の保証可否は，それらから構築されたゲームの勝利領域に初期状態が含まれているかで判定することが可能である．これより，保証可能な安全性を特定することは初期状態が勝利領域に含まれるような安全性の組合せを，入力される全安全性から発見することと同じである．本研究ではこれを次のような勝利領域としてとらえる．

定義 5. $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, (\lambda, \Sigma))$ が与えられたとき，$\Sigma' \subseteq \Sigma$ であるような $G_{(\lambda,\Sigma')} = (S_g, \Gamma^-, \Gamma^+, s_{g0}, (\lambda, \Sigma'))$ の勝利領域 $w_{(\lambda,\Sigma')}$ を Σ' に限定した勝利領域とよび，すべての $\Sigma' \subseteq \Sigma$ に限定した勝利領域の集合を条件付き勝利領域とよぶ．

図 2 は安全性付き Büchi ゲームの一例を示している．黒

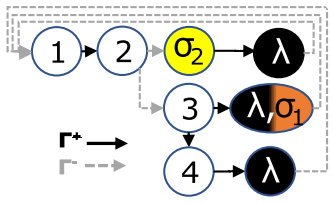


図 2 安全性付き Büchi ゲームの例
Fig. 2 An example of safety-Büchi game.

い実線は Γ^+ を表し灰色の破線は Γ^- を表す．色付きの状態は勝利条件を表しており，黒は活性 λ，橙と黄は安全性 σ_1，σ_2 に属する状態を表す．2 色持つ状態は 2 つの勝利条件に属している．図 2 における制御器の手番は 1，3，4，σ_2 であり環境の手番は 2 と λ に属する状態すべてである．環境の手番である 2 から σ_2 への遷移があり，λ に属するいずれの状態も 2 を避けて無限に到達することは不可能であることからこのゲーム自体の勝利領域は存在しない．しかし σ_1 は 3 が制御器の手番であることから避けることが可能である．したがって $\Sigma' = \{\sigma_1\}$ に限定した勝利領域は存在する．また，このとき σ_1 も σ_2 も避けないことを許容する $\Sigma'' = \emptyset$ に限定した勝利領域も存在する．

この条件付き勝利領域の要素を 1 つずつ特定する場合，安全性の冪集合の数だけ勝利領域特定のアルゴリズムを適用する必要がある．この要素の特定には Büchi ゲームの勝利領域を求めるアルゴリズムが利用できる [7] ため，1 つの要素にかかる計算量は，状態数を n としたときに $O(n^2)$ である [20]．これより全体の計算量は安全性の個数を r とすると $O(n^2 \times 2^r)$ となり，扱う要求の数に対して指数関数的に増加する．本論文ではこの条件付き勝利アルゴリズムをより効率的に特定するアルゴリズムを提案する．

6. 条件付き勝利領域特定アルゴリズム

Büchi ゲームの勝利領域を求めるアルゴリズムは到達可能性ゲームの勝利領域を求めるアルゴリズムを繰り返し適用するという特徴がある．本論文ではこの特徴に着目し，Büchi ゲームの勝利領域を求めるアルゴリズムの形を保ちながら，到達可能性ゲームの勝利領域を求める部分で複数の領域を扱うことで条件付き勝利領域を特定する．

Algorithm 1 は条件付き勝利領域を求めるアルゴリズムの全体像である．安全性付き Büchi ゲームを入力として条件付き勝利領域を出力とする．条件付き勝利領域の特定は λ を起点に始まる．まず，λ に $\Sigma' \subseteq \Sigma$ を避けて到達可能な領域の集合 W' を求め（6 行目），各領域 $w \in W'$ に含まれる λ の要素のすべてで w 自身に到達可能であるかを調べ，到達可能であればそれを条件付き勝利領域 W に加える（7–11 行目）．すべての λ が自身に到達可能でなかった w については，到達可能な $Z_n \subseteq \lambda$ のみを用いて再び領域を求め直す．12 行目の $|Z_n|$ は Z_n の要素数を表す．これ

Algorithm 1 条件付き勝利領域の特定

```
1: INPUT: G = (S_g, Γ^-, Γ^+, s_g0, (λ, Σ))
2: OUTPUT: W
3: W ⇐ {}, Z_0 ⇐ {}, Z_1 ⇐ λ, n ⇐ 1
4: while Z_n ≠ Z_{n-1} do
5:     n ⇐ n + 1, Z_n ⇐ {}
6:     W' ⇐ Algorithm2(S_g, Γ^-, Γ^+, Z_{n-1})
7:     for all w ∈ W' do
8:         Z' ⇐ Pre(w) ∩ Z_{n-1}
9:         if Z' = Z_{n-1} then
10:            W ⇐ W ∪ {w}
11:        end if
12:        if |Z_n| < |Z'| then
13:            Z_n ⇐ Z'
14:        end if
15:    end for
16:    if Z_n = {} then
17:        break
18:    end if
19: end while
20: return W
21: Pre(X) = {s_g ∈ S_g | Γ^-(s_g) = ∅, Γ^+(s_g) ∩ X ≠ ∅}
            ∪ {s_g ∈ S_g | Γ^-(s_g) ⊆ X}
```

Algorithm 2 λ に $\Sigma' \subseteq \Sigma$ を避けて到達可能な領域の特定

```
1: INPUT: S_g, Γ^-, Γ^+, λ', Σ
2: OUTPUT: W_n
3: W_0 ⇐ {}, W_1 ⇐ {}
4: for s_λ ∈ λ do
5:     Σ' ⇐ Σ\{σ ∈ Σ | s_λ ∈ σ}
6:     w_(λ',Σ') ⇐ w_(λ',Σ') ∪ {s_λ}
7:     W_1 ⇐ W_1 ∪ {w_(λ',Σ')}
8: end for
9: n = 1
10: while W_n ≠ W_{n-1} do
11:     n ⇐ n + 1
12:     W_n ⇐ Algorithm 3(W_{n-1}, Γ^-, S_g, Σ)
13:     W_n ⇐ Algorithm 4(W_n, Γ^+, S_g, Σ)
14: end while
15: return W_n
```

を繰り返すことで条件付き勝利領域の各要素を求めることが可能である．ここで，ある状態の集合 X に到達可能な状態とは X に含まれる状態への遷移を持つ制御器の手番またはすべての遷移が X に含まれているような環境の手番のことをいい，Pre はこれを求める関数である（21 行目）．

Algorithm 2 は入力された λ' に $\Sigma' \subseteq \Sigma$ を避けて到達可能な領域の集合を出力する．このアルゴリズムでは λ' に到達可能な領域を λ' から逆伝播させる（10–14 行目）．この伝播対象の状態が環境の手番か制御器の手番かで伝播方法が異なり，環境の手番における領域の伝播を **Algorithm 3** で，制御器の手番における領域の伝播を **Algorithm 4** で行っている．すべての $w \in W'$ について領域がそれ以上伝播されなくなれば終端となる．

Algorithm 3 はすでに特定されている領域の集合 W' 内のいずれかへの遷移を持つ環境の手番に W' を伝播する

Algorithm 3 環境の手番における到達可能性判定

```
1: INPUT:W, Γ⁻, S_g, Σ
2: OUTPUT:W′
3: W′ ⇐ {}
4: for all s_g ∈ S_g|∀y ∈ Γ⁻(s_g), y ∈ ⋃_{w(λ,Σ′)∈W} w_(λ,Σ′) do
5:    F^{Σ′} ⇐ ∏^∩_{s′_g∈Γ⁻(s_g)} {Σ′ ∈ 2^Σ | s′_g ∈ w_(λ,Σ′), w_(λ,Σ′) ∈ W}
6:    for all Σ′ ∈ F^{Σ′} do
7:       Σ″ ⇐ Σ′
8:       for all σ ∈ Σ′|s_g ∈ σ do
9:          Σ″ ⇐ Σ″\{σ}
10:      end for
11:      w_(λ,Σ″) ⇐ w_(λ,Σ″) ∪ w_(λ,Σ′) ∪ {s_g}
12:      W′ ⇐ W′ ∪ {w_(λ,Σ″)}
13:   end for
14: end for
15: return W′
16: ∏^∩ : 2^Σ → 2^Σ such that
    ∏^∩_{ϕ∈Φ} A_ϕ = {⋂_{ϕ∈Φ} a_ϕ | a_ϕ ∈ A_ϕ, ∀ϕ ∈ Φ}
```

Algorithm 4 制御器の手番における到達可能性判定

```
1: INPUT: W, Γ⁻, Γ⁺, S_g, Σ
2: OUTPUT W′
3: W′ ⇐ {}
4: for all s_g ∈ S_g|Γ⁻(s_g) = ∅, ∃y ∈ Γ⁺(s_g),
          y ∈ ⋃_{w(λ,Σ′)∈W} w_(λ,Σ′) do
5:    for all w_(λ,Σ′) ∈ W|Γ⁺(s_g) ∩ w_(λ,Σ′) ≠ ∅ do
6:       Σ″ ⇐ Σ′
7:       for all σ ∈ Σ′|s_g ∈ σ do
8:          Σ″ ⇐ Σ″\{σ}
9:       end for
10:      w_(λ,Σ″) ⇐ w_(λ,Σ″) ∪ w_(λ,Σ′) ∪ {s_g}
11:      W′ ⇐ W′ ∪ {w_(λ,Σ″)}
12:   end for
13: end for
14: return W′
```

アルゴリズムである．対象となる環境の手番を特定（4 行目）し，その環境の手番が含まれる領域を特定する（5 行目）．環境の手番では環境が任意に遷移を選択できるため，遷移先のいずれかが $\sigma \in \Sigma$ を避けられない領域に属する場合，環境の手番でも同様に避けることができない．これより環境の手番が属する領域は遷移先のいずれの領域でも避けることのできる安全性の集合 $\Sigma' \subseteq \Sigma$ である．ただし，遷移先が複数の領域に属する場合，遷移先でどの $\sigma \in \Sigma$ を避けることを諦めるかの選択肢が生まれる．これを考慮すると環境の手番が属する領域は遷移先が属する領域の組合せ数分の領域に属することになる．また，環境の手番自身がいずれかの $\sigma \in \Sigma$ に属する場合はそのような σ を Σ' から取り除く（7–10 行目）．最後に特定した Σ' に対応する領域に環境の手番を含めて領域の集合自体も更新する（11–12 行目）．

Algorithm 4 はすでに特定されている領域の集合 W' 内のいずれかへの遷移を持つ制御器の手番に W' を伝播するアルゴリズムである．制御器の手番では制御器が遷移を選択できるため，遷移先の状態が属する領域のいずれにも属することができる．対象となる制御器の手番を特定し（4 行目），それらの遷移先が属する領域に加えていく（10 行目）．このとき制御器の手番自身がいずれかの $\sigma \in \Sigma$ に含まれる場合は **Algorithm 3** と同様の処理を行う．

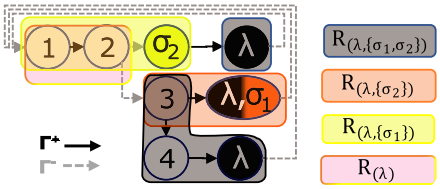


図 3 図 2 の条件付き勝利領域

Fig. 3 The winning region of safety-Büchi game in Fig. 2.

図 3 は **Algorithm 2** を図 2 に適用して特定できる領域を表している．黒の領域は σ_1 も σ_2 も避けて λ に到達でき，橙は σ_2 を，黄は σ_1 をそれぞれ避けて λ に到達できる．複数色の領域は σ_1 も σ_2 も避けない領域である．ただし，黒の領域は他のすべての領域に，橙と黄色の領域は複数色の領域にも含まれるものとする．この領域の特定は λ に属するすべての状態を起点に行われる．このとき σ_1 にも属する状態は黒でなく橙の領域を伝播させる．1 や 3，4 のように制御器の手番であれば **Algorithm 4** によって遷移先の領域がそのまま伝播される．σ_2 のようにいずれかの $\sigma \in \Sigma$ に属する状態であれば領域にその情報を付加して伝播させる（黄の領域）．2 のように環境の手番であれば **Algorithm 3** によって遷移先のすべてが避けることのできる領域を計算し伝播させる．2 では 2 つある遷移先の一方が黄色の領域に属し，もう一方が黒と橙の領域に属している．このとき黒と黄から黄の領域が算出され，橙と黄からは複数色の領域が算出される．**Algorithm 2** によって特定されたこの領域は，**Algorithm 1** において λ から到達可能か調べて勝利条件に十分かを確認される．黒と橙の領域は λ に属する状態が唯一到達可能な 1 まで伝播されていないため，λ に到達できない．複数色の領域は λ のすべてを持ち，かつ 1 も含んでいるため条件付き勝利領域に含まれる．黄色の領域は 1 を含んでいるが，σ_1 にも λ にも属する状態が領域に含まれないため，これを除外して再特定することになる．**Algorithm 1** はこのように条件付き勝利領域を特定する．

7. アルゴリズムの妥当性確認

本章では前章で提案したアルゴリズムが条件付き勝利領域を特定できることの証明とアルゴリズムの計算量の検討をもって妥当性の確認を行う．

7.1 アルゴリズムが条件付き勝利領域が特定できることの証明

Algorithm 1 が条件付き勝利領域を特定することを証

明するために (1) **Algorithm 3** と **Algorithm 4** が到達可能な領域を遷移元に伝播させられること, (2) **Algorithm 2** が入力された活性を満たす状態の集合 $\lambda' \in \lambda$ に安全性の違反を表す状態の集合族 $\Sigma' \subseteq \Sigma$ のすべてを避けて到達できる領域を特定できることを証明する．そしてこれらを用いて条件付き勝利領域が特定できることを証明する．

Algorithm 3 が環境の手番に到達可能な領域を伝播させられることを証明するためには **Algorithm 3** の 5 行目で求める領域に環境の手番が含まれることを証明すればよい．これに先立って以下の補題を証明する．

補題 1. $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, (\lambda, \Sigma))$ が与えられたとき，$s_g \in S_g$ が $\Sigma' \subseteq \Sigma$ に限定した勝利領域 $w_{(\lambda,\Sigma')}$ に含まれるなら s_g は $\Sigma'' \subseteq \Sigma'$ に限定した勝利領域 $w_{(\lambda,\Sigma'')}$ に含まれる．

証明. $s_g \in w_{(\lambda,\Sigma')}$ より，制御器は s_g から $G_{(\lambda,\Sigma')}$ を始めても環境の選択に非依存に勝利することができる．つまり，制御器は環境の選択に非依存に $s_\lambda \in \lambda$ に無限回到達できかつ，任意の $s_\sigma \in \sigma$，$\sigma \in \Sigma'$ を避けることができる．このとき $\Sigma'' \subseteq \Sigma'$ であることから，制御器は任意の $s_\sigma \in \sigma$，$\sigma \in \Sigma''$ も避けることが可能である．したがって制御器は s_g から $G_{(\lambda,\Sigma'')}$ を始めても環境の選択に非依存に勝利することができる． □

これを用いて以下の補題を証明する．

補題 2. $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, (\lambda, \Sigma))$ が与えられたとき，$\Gamma^-(s_g) \neq \emptyset$ であるような $s_g \in S_g$ を考える．すべての $s'_g \in \Gamma^-(s_g)$ がいずれかの $\Sigma' \subseteq \Sigma$ に限定した勝利領域 $w_{(\lambda,\Sigma')}$ に含まれるならば s_g は $\Sigma'' \in \prod^{\cap}_{s'_g \in \Gamma^-(s_g)} \{\Sigma'' \in 2^\Sigma | s'_g \in w_{(\lambda,\Sigma'')}, w_{(\lambda,\Sigma'')} \in W\}$ であるような Σ'' に対して $\Sigma''' = \Sigma'' \backslash \{\sigma \in \Sigma'' | s_g \in \sigma\}$ であるような Σ''' に限定した勝利領域 $w_{(\lambda,\Sigma''')}$ のすべてに含まれる．

証明. $\Gamma^-(s_g) \neq \emptyset$ より s_g は環境の手番である．このときすべての $s'_g \in \Gamma^-(s_g)$ が $w_{(\lambda,\Sigma'')}$ に含まれていれば環境が任意の s'_g を選んでも制御器は $G_{(\lambda,\Sigma'')}$ に環境の選択に非依存に勝利できる．補題 1 より s'_g が $\Sigma'' \subseteq \Sigma'$ であるような $w_{(\lambda,\Sigma')}$ に含まれていれば s'_g は $w_{(\lambda,\Sigma'')}$ にも含まれる．すべての $s_{gi} \in \Gamma^-(s_g)$ に対して，それぞれの s_{gi} を含む $w_{(\lambda,\Sigma'_i)}$ を 1 つずつ取り出して $\Sigma'' = \bigcap \Sigma'_i$ とすると，すべての s_{gi} は $w_{(\lambda,\Sigma'')}$ に含まれる．よって s_g は $w_{(\lambda,\Sigma'')}$ に環境の選択に非依存に到達できる．これはそれぞれの s_{gi} を含む $w_{(\lambda,\Sigma'_i)}$ の組合せごとに成り立つ．ただし $s_g \in \sigma$ であるような $\sigma \in \Sigma''$ は避けることができない．以上より s_g は $\Sigma'' \in \prod^{\cap}_{s'_g \in \Gamma^-(s_g)} \{\Sigma'' \in 2^\Sigma | s'_g \in w_{(\lambda,\Sigma'')}, w_{(\lambda,\Sigma'')} \in W\}$ に対して $\Sigma''' = \Sigma'' \backslash \{\sigma \in \Sigma'' | s_g \in \sigma\}$ であるようなすべての $w_{(\lambda,\Sigma''')}$ に含まれる． □

Algorithm 4 が制御器の手番に到達可能な領域を伝播させられることを証明するためには制御器の手番 s_g の遷移先が含まれるすべての領域に s_g も含まれることを証明すればよい．よって以下の補題を証明する．

補題 3. $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, (\lambda, \Sigma))$ が与えられたとき，$\Gamma^+(s_g) \neq \emptyset \wedge \Gamma^-(s_g) = \emptyset$ であるような $s_g \in S_g$ を考える．$s'_g \in \Gamma^+(s_g)$ が $\Sigma' \subseteq \Sigma$ に限定した勝利領域 $w_{(\lambda,\Sigma')}$ に含まれるなら s_g は $\Sigma'' = \Sigma' \backslash \{\sigma \in \Sigma' | s_g \in \sigma\}$ であるような $w_{(\lambda,\Sigma'')}$ に含まれる．

証明. $\Gamma^+(s_g) \neq \emptyset \wedge \Gamma^-(s_g) = \emptyset$ より s_g は制御器の手番であることより $s'_g \in w_{(\lambda,\Sigma')}$ であるような $s'_g \in \Gamma^+(s_g)$ を選ぶことで $w_{(\lambda,\Sigma'')}$ に環境の選択に非依存に到達できる．ただし $s_g \in \sigma$ であるような $\sigma \in \Sigma'$ は避けることができない．よって s_g は $\Sigma'' = \Sigma' \backslash \{\sigma \in \Sigma' | s_g \in \sigma\}$ であるような $w_{(\lambda,\Sigma'')}$ に含まれる． □

補題 2 と補題 3 を用いることで **Algorithm 2** について以下を証明できる．

補題 4. $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, (\lambda, \Sigma))$ が与えられたとき，$\Sigma' \subseteq \Sigma$ を避けて λ に到達可能な領域の集合は **Algorithm 2** によって得られる．

証明. **Algorithm 2** の W_n がつねに $\Sigma' \subseteq \Sigma$ を避けて λ に到達可能な領域の集合 W' の一部となることを証明する．$n = 0$ のとき W_1 の要素は $w_{(\lambda,\Sigma')} = \lambda \backslash \{s_\lambda \in \lambda | s_\lambda \in \sigma, \sigma \in \Sigma, \sigma \notin \Sigma'\}$ であるような $w_{(\lambda,\Sigma')}$ であるため，W' の一部である．$n \geq 1$ のとき，$n = k - 1$ として W_{k-1} が W' の一部であると仮定する．このとき補題 2 と補題 3 から **Algorithm 3** と **Algorithm 4** によって更新された $w_{(\lambda,\Sigma')} \in W_k$ はいずれも $\Sigma' \subseteq \Sigma$ を避けて λ に到達可能な領域の一部である．したがって W_k も W' の一部である．これより W_n はつねに W' の一部となる．$W_{k-1} = W_k$ となる最小の k を κ としたときに W_κ が W' であることを証明する．ある $\Sigma' \subseteq \Sigma$ を避けて λ に到達可能だが $w_{(\lambda,\Sigma')} \in W_\kappa$ に含まれない状態 $s_{(\lambda,\Sigma')}$ が存在すると仮定する．$s_{(\lambda,\Sigma')}$ が環境の手番であるとき，その遷移先すべてが $\Sigma' \subseteq \Sigma$ を避けて λ に到達可能だが，そのような $s_{(\lambda,\Sigma')}$ は **Algorithm 3** によって $w_{(\lambda,\Sigma')}$ に含められる．$s_{(\lambda,\Sigma')}$ が制御器の手番であるとき，その遷移のいずれかが $\Sigma' \subseteq \Sigma$ を避けて λ に到達可能だが，そのような状態は **Algorithm 4** によって $w_{(\lambda,\Sigma')}$ に含められる．以上より $\Sigma' \subseteq \Sigma$ を避けて λ に到達可能な $s_{(\lambda,\Sigma')}$ はすべて $w_{(\lambda,\Sigma')} \in W_\kappa$ に含まれるため，W_κ は W' である． □

この補題 4 を用いて以下の定理を証明する．

定理 1. $G = (S_g, \Gamma^-, \Gamma^+, s_{g0}, (\lambda, \Sigma))$ が与えられたとき，条件付き勝利領域 W は **Algorithm 1** によって得られる．

証明. **Algorithm 1** において，$Z_n = Z_{n-1}$ となるような最小の n を η とする．$0 \leq k \leq \eta$ において補題 4 より W' は Z_k に $\Sigma' \subseteq \Sigma$ を避けて到達可能な領域の集合である．任意の $w_{(\lambda,\Sigma')} \in W'$ について，$Z_k \subseteq Pre(w_{(\lambda,\Sigma')})$ であるような $w_{(\lambda,\Sigma')}$ であれば $Z_k \subseteq \lambda$ に無限に到達できることから **Algorithm 1** の 9–11 行目より $w_{(\lambda,\Sigma')} \in W$ は Σ' に限定した勝利領域の一部である．$w_{(\lambda,\Sigma')}$ は Σ' に限定した勝利領域を含むことを示す．Σ' に限定した勝利領域

に属する状態で，$w_{(\lambda,\Sigma')}$ に含まれない状態 $s_{(\lambda,\Sigma')}$ が存在すると仮定する．$s_{(\lambda,\Sigma')}$ は Σ' のいずれかに属する状態すべてを避けて λ に無限に到達できるが，$s_{(\lambda,\Sigma')}$ が到達する $\lambda' \subseteq \lambda$ 自身も同様に λ に無限に到達できなければならず，**Algorithm 1** の 9–11 行目よりそのような λ' は $w_{(\lambda,\Sigma')}$ に含まれる．したがって $s_{(\lambda,\Sigma')}$ は $w_{(\lambda,\Sigma')}$ に含まれる λ' に到達可能であるため補題 4 より $w_{(\lambda,\Sigma')}$ に含まれる．以上より W は条件付き勝利領域である． □

7.2 アルゴリズムの計算量

提案したアルゴリズムの計算量は **Algorithm 2** の計算量と **Algorithm 1** 内で **Algorithm 2** を繰り返す回数から算出できる．以下，与えられたゲームの状態数 n，遷移数 m，安全性の要素の個数を r とする．**Algorithm 1** において **Algorithm 2** を繰り返す回数の最悪値は $\lambda \subseteq S_g$ の要素数であることより n である．

次に **Algorithm 2** の計算量について考える．状態が属する領域を伝播する際には，状態にそれが属する領域を論理式でラベル付けしておき論理演算で 1 度に処理できる．したがって計算量に支配的な要素は状態に領域を伝播させる回数である．1 つの状態が複数の領域に属するため伝播回数の最悪値は $n \times 2^r$ となる．しかし以下の手順で伝播のタイミングを待ち合わせることで改善可能である．

(1) $|\Sigma'|$ の大きい $w_{(\lambda,\Sigma')} \in W'$ を優先的に伝播させる．
(2) $|\Sigma'|$ の同じ $w_{(\lambda,\Sigma')} \in W'$ の伝播では (4) まで同じ遷移からの伝播はせずに，初めての遷移のみ伝播させる．
(3) (2) の伝播は起点とした状態から深さ優先に行う．最初の起点を 1 とし，起点が変わるたびに 1 増える値を o とし，伝播させた状態に記憶させる．
(4) (2) の $o > 1$ における伝播中にすでに他の伝播が行われた状態に到達したらその先の伝播は停止し，その到達先の o を親，到達元の o を子として木を構築する．
(5) 2 回目の伝播は (4) で構築した木の葉を持つノードから伝播を開始し，子の領域をノード内に伝播する．以降，木の葉から根へと伝播を繰り返す．

(1) は補題 1 から $\Sigma'' \subseteq \Sigma'$ ならば $w_{(\lambda,\Sigma')} \subseteq w_{(\lambda,\Sigma'')}$ であることから $w_{(\lambda,\Sigma')}$ を優先的に特定し，$w_{(\lambda,\Sigma'')}$ の特定時は $\Sigma'' \subseteq \Sigma'$ を満たすすべての $w_{(\lambda,\Sigma')}$ の領域の和集合を $w_{(\lambda,\Sigma'')}$ に事前に含めることで伝播を省略できる．

また，(2)–(5) によって $|\Sigma'|$ の同じ $w_{(\lambda,\Sigma')}$ を伝播させるときに，同じ遷移を最大でも 2 回しか通らないようにすることができる．**図 4** は $|\Sigma'| = 1$ であるような 3 つの領域の伝播時に構築した木の例である．簡単のために遷移は Γ^+ のみとしている．(2) の伝播は σ_2，σ_3，σ_1 の順に行っており右側の木のノードの値は o である．この木から (5) の伝播は σ_2 の領域（黄色）内で σ_1，σ_3 の領域を伝播させていく．個々に伝播を進めた場合，1 から 2 への遷移を最大 3 回通りうるが，伝播する領域を待ち合わせることで上限を 2 回に制限することができている．

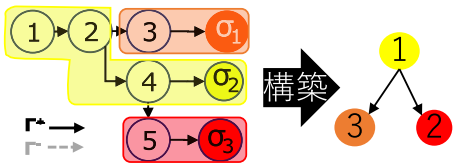


図 4 到達可能領域特定時の木の構築例
Fig. 4 An example of the tree for identifying reachable region.

表 1 環境モデル中の状態数と遷移数の例
Table 1 Examples of the number of states and transitions in environment models.

環境モデル例	状態数（n）	遷移数（m）	$n \log_2 n$（参考値）
文献 [3]	16	39	64
文献 [19]	52	188	296
文献 [7]	592	3,280	5,452
文献 [12]	2,884	7,813	33,148

以上より $|\Sigma'|$ が同じ領域は $O(2m)$ で伝播でき，$0 \leq |\Sigma'| \leq r$ と (1) から **Algorithm 2** の計算量は $O(2mr)$ となり，**Algorithm 1** の計算量は $O(2nmr)$ となる．5 章で述べたとおり，条件付き勝利領域の各要素を Büchi ゲームのアルゴリズムを用いて求めた場合の計算量は $O(n^2 \times 2^r)$ である．

Algorithm 1 の計算量を導出するにあたっては遷移数 m を用いている．これは様々なゲームの勝利戦略を導出するアルゴリズム [13] において計算量を示す際に用いられるものだが，これと $O(n^2 \times 2^r)$ を比較するために状態数 n に対して遷移数 m を見積もる．ゲームはグラフの形をとるので遷移数 m の最悪値は n^2 であるが，実際に構築されるゲームの遷移数が最悪値に近づくことは稀である．たとえば **表 1** はゲームを扱う文献 [3], [7], [12], [19] 中の環境モデルの状態数と遷移数を示している[*1]．表 1 中のいずれのモデルにおいても，その遷移数 m は状態数 n に対して $n \log_2 n$ を下回る値となっている．環境モデルから構築されるゲーム [7], [19] では環境モデル中の状態遷移関係が保存されていることより状態数と遷移数との関係も同様になる．

そこで本論文ではゲーム全体の遷移数を $m \simeq n \times \log n$ と見積もる．**Algorithm 1** の計算量は $O(2nmr) \simeq O(2n^2 \times r \log n)$ となる．これは r がある程度大きければ $O(n^2 \times 2^r)$ と比較して高速に計算できる．たとえば $O(2n^2 \times r \log n) \simeq 2n^2 \times r \log n$，$O(n^2 \times 2^r) \simeq n^2 \times 2^r$ とすると $r \geq 10$ のときに $n \leq 10^{10}$ の範囲で高速になることが確認できる．$n \geq 10^{10}$ となるゲームはきわめて稀な一方で $r \geq 10$ となるゲームの存在は十分に考えられる [12], [19]．したがって現実的な規模のゲームにおいて **Algorithm 1** は Büchi ゲームのアルゴリズムを用いる方法に比べて高速に条件付き勝利領域を求めることができると考えられる．

[*1] 確認時の環境モデルは以下に保管，https://github.com/k-aizawa/softwareEngineering2019

8. おわりに

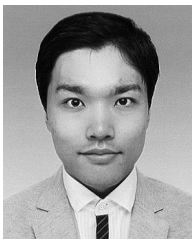
本論文では与えられた環境下で活性を満たしながら保証可能な安全性を特定するためのゲームおよび条件付き勝利領域の定義と条件付き勝利領域特定のためのアルゴリズムを提案した．また，提案したアルゴリズムによって特定される領域が条件付き勝利領域であることの証明とアルゴリズムの計算量の検討を行い，提案したアルゴリズムがゲーム内の状態数 n，遷移数 m，扱う安全性の要素数 r に対して $O(2nmr)$ で動作することを導いた．

本論文の将来研究としては次の 3 つがあげられる．1 つ目は提案したアルゴリズムを実装し，3 章にあげた例を始めとする問題に対する実計算時間の評価を行うことである．2 つ目は扱える要求の範囲を広げるために活性の保証可否も扱うことである．3 つ目は本アルゴリズムを実行時に適用するための拡張を行うことである．実行時に変化した環境下でも，保証可能な安全性を高速に特定するために変化の差分のみを分析し直すことなどが考えられる．

謝辞 本研究の一部は JSPS 科研費 18H03225，17H00732，および独立行政法人情報通信研究機構（NICT）の委託研究「欧州との連携によるハイパーコネクテッド社会のためのセキュリティ技術の研究開発」の成果です．

参考文献

[1] Nuseibeh, B.: Weaving together requirements and architectures, *Computer*, Vol.34, No.3, pp.115–119 (2001).
[2] Cleland-Huang, J., Hanmer, R., Supakkul, S. and Mirakhorli, M.: The Twin Peaks of Requirements and Architecture, *IEEE Software*, Vol.30, No.2, pp.24–29 (2013).
[3] D'Ippolito, N., Braberman, V., Kramer, J., Magee, J., Sykes, D. and Uchitel, S.: Hope for the Best, Prepare for the Worst: Multi-tier Control for Adaptive Systems, *Proc. 36th International Conference on Software Engineering*, *ICSE 2014*, pp.688–699, ACM (2014).
[4] Lamport, L.: Proving the Correctness of Multiprocess Programs, *IEEE Trans. Softw. Eng.*, Vol.3, No.2, pp.125–143 (1977).
[5] Alpern, B. and Schneider, F.B.: Recognizing Safety and Liveness, *Distributed Computing*, Vol.2, No.3, pp.117–126 (1987).
[6] Piterman, N., Pnueli, A. and Sa'ar, Y.: Synthesis of Reactive (1) Designs, *Verification, Model Checking, and Abstract Interpretation*, Emerson, E.A. and Namjoshi, K.S. (Eds.), pp.364–380, Springer Berlin Heidelberg (2006).
[7] D'Ippolito, N.R., Braberman, V., Piterman, N. and Uchitel, S.: Synthesis of Live Behaviour Models, *Proc. 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, *FSE '10*, pp.77–86, ACM (2010).
[8] Debois, S., Hildebrandt, T. and Slaats, T.: Safety, Liveness and Run-Time Refinement for Modular Process-Aware Information Systems with Dynamic Sub Processes, *FM 2015: Formal Methods*, Bjørner, N. and de Boer, F. (Eds.), pp.143–160, Springer International Publishing (2015).
[9] Cámara, J., Schmerl, B., Moreno, G.A. and Garlan, D.: MOSAICO: Offline synthesis of adaptation strategy repertoires with flexible trade-offs, *Automated Software Engineering* (2018).
[10] Calinescu, R., Ghezzi, C., Kwiatkowska, M. and Mirandola, R.: Self-adaptive Software Needs Quantitative Verification at Runtime, *Comm. ACM*, Vol.55, No.9, pp.69–77 (2012).
[11] Tsigkanos, C., Pasquale, L., Menghi, C., Ghezzi, C. and Nuseibeh, B.: Engineering topology aware adaptive security: Preventing requirements violations at runtime, *2014 IEEE 22nd International Requirements Engineering Conference* (*RE*), pp.203–212 (2014).
[12] 相澤和也，鄭 顕志，本位田真一：環境変化時に保証可能な安全性を特定するためのゲーム分析アルゴリズム，情報処理学会論文誌，Vol.60, No.4, pp.1025–1039 (2019).
[13] Grädel, E., Thomas, W. and Wilke, T. (Eds.): *Automata Logics, and Infinite Games: A Guide to Current Research*, Springer-Verlag New York, Inc. (2002).
[14] Abushark, Y., Thangarajah, J., Miller, T., Harland, J. and Winikoff, M.: Early Detection of Design Faults Relative to Requirement Specifications in Agent-Based Models, *Proc. 2015 International Conference on Autonomous Agents and Multiagent Systems*, *AAMAS '15*, pp.1071–1079, International Foundation for Autonomous Agents and Multiagent Systems (2015).
[15] DeVries, B. and Cheng, B.H.C.: Using Models at Run Time to Detect Incomplete and Inconsistent Requirements, *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems* (*MODELS 2017*), pp.201–209 (2017).
[16] Ciolek, D., Braberman, V., D'Ippolito, N. and Uchitel, S.: Directed Controller Synthesis of discrete event systems: Taming composition with heuristics, *2016 IEEE 55th Conference on Decision and Control* (*CDC*), pp.4764–4769 (2016).
[17] Lewerentz, C. and Lindner, T. (Eds.): *Formal Development of Reactive Systems – Case Study Production Cell*, Springer-Verlag (1995).
[18] Giannakopoulou, D. and Magee, J.: Fluent Model Checking for Event-based Systems, *SIGSOFT Softw. Eng. Notes*, Vol.28, No.5, pp.257–266 (2003).
[19] Aizawa, K., Honiden, S. and Tei, K.: Analysis space reduction with state merging for ensuring safety properties of self-adaptive systems, *The 16th IEEE International Conference on Advanced and Trusted Computing 2019* (*ATC 2019*), Leicester, United Kingdom (Great Britain) (2019) (online), available from ⟨http://www.honiden.jp/wp-content/uploads/2019/05/ATC2019_aizawa_camera.pdf⟩.
[20] Chatterjee, K. and Henzinger, M.: An O(N2) Time Algorithm for Alternating Büchi Games, *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, *SODA '12*, pp.1386–1399, Society for Industrial and Applied Mathematics (2012).



相澤 和也 （学生会員）

2014 年早稲田大学基幹理工学部情報理工学科卒業．2016 年同大学大学院基幹理工学研究科情報理工・情報通信専攻修士課程修了．みずほ情報総研（株）を経て 2018 年より早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻博士課程．2019 年同大学情報理工学科助手．現在に至る．

鄭 顕志 （正会員）

2008 年早稲田大学大学院理工学研究科情報・ネットワーク専攻博士課程修了．2006 年同大学理工学部助手．2007 年同大学基幹理工学部助手．2008 年同大学メディアネットワークセンター助教．2010 年国立情報学研究所アーキテクチャ科学研究系助教．2015 年同准教授．2018 年早稲田大学理工学術院総合研究所研究院准教授/主任研究員．2019 年同大学理工学術院国際理工学センター准教授，現在に至る．現在，国立情報学研究所 GRACE センター特任准教授を兼任．博士（工学）（早稲田大学）．自己適応ソフトウェア，ソフトウェアアーキテクチャ，モデル駆動工学の研究に従事．

本位田真一 （正会員）

1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年国立情報学研究所教授，2012 年同研究所副所長を兼務．2001 年東京大学大学院情報理工学系研究科教授を兼任，2018 年より早稲田大学理工学術院教授，現在に至る．現在，英国 UCL 客員教授ならびに国立情報学研究所 GRACE センター長を兼任．2005 年度パリ第 6 大学招聘教授．2015 年度リヨン第 1 大学招聘教授．日本学術会議連携会員．本会フェロー．