情報処理学会論文誌 Vol.59 No.1 1–13 (Jan. 2018)

ゲーム空間の一括構築と段階構築の統合による離散制御器合成の計算空間削減

山内 拓人[1,2,a)] 李 家隆[1] 鄭 顕志[1,2,3] 本位田 真一[3]

受付日 2016年3月4日, 採録日 2016年8月1日

概要：想定される動作環境下で安全性が保証された動作仕様を自動合成する Discrete Controller Synthesis（DCS）において，計算空間の指数爆発の課題はその実践的な適用を阻む主要因となっている．本課題に対処するため，本論文では先行の計算空間削減手法を相補的に組み合わせた Consolidated Stepwise Discrete Controller Synthesis（CSDCS）を提案する．段階的なゲーム空間構築アプローチに違反状態空間の抽象化処理を採用することで，計算空間削減効果を維持したまま計算空間の構築回数を抑えることができる．結果，従来トレードオフの関係にあった計算空間削減と計算時間削減の両方を高い水準で両立を可能にした．その性能を形式的な妥当性証明と 7 つのシナリオを通して確認した結果，計算時間を平均 43.6%，計算空間を平均 51.8%削減しつつ従来の DCS と同じ機能を果たすことが可能であることがわかった．

キーワード：離散制御器合成，二人型対戦ゲーム理論，ラベル付き遷移システム

Computational Space Reduction in Discrete Controller Synthesis via Integration of Consolidated and Stepwise Game Space Construction

Takuto Yamauchi[1,2,a)] Jialong Li[1] Kenji Tei[1,2,3] Shinichi Honiden[3]

Received: March 4, 2016, Accepted: August 1, 2016

Abstract:
In Discrete Controller Synthesis (DCS), which automatically synthesizes behavior specifications that guarantee safety under assumed environments, the issue of state space explosion remains a major obstacle to its practical application.To address this issue, this paper proposes Consolidated Stepwise Discrete Controller Synthesis (CSDCS), which complements and integrates prior state space reduction techniques. By incorporating violation-state abstraction into a stepwise game space construction approach, the number of game space constructions can be reduced while maintaining the effectiveness of state space reduction. As a result, CSDCS achieves a high level of both state space reduction and computation time reduction, which were previously in a trade-off relationship. Formal validation and evaluations on seven scenarios demonstrated that CSDCS can achieve the same functionality as conventional DCS while reducing computation time by an average of 43.6% and space by an average of 51.8%.

Keywords: Discrete Controller Synthesis, Two-players Game Theory, Labeled Transition System

1. まえがき

事象の発生によりシステムの状態が離散的に遷移する離散事象システムの開発では，開発の早期段階において安全性が保証された動作仕様を定めることが重要となる．想定される動作環境下において安全性が保証された動作仕様を自動合成する技術のひとつとして，Discrete Controller Synthesis（DCS）[1] に関する研究が進められてきた．

DCS を用いた動作仕様設計は次の手順で行われる．はじめに，開発者はシステムの動作環境を仮定し，その特性を環境モデルとしてモデル化する．次に，環境モデル下で保証すべき安全性を定義し，その安全性の充足状況を監視

[1] 早稲田大学 Waseda University
[2] 東京工業大学 Tokyo Institute of Technology
[3] 国立情報学研究所 National Institute of Informatics (NII)
[a)] takuto.yamauchi@aoni.waaseda.jp

する監視モデルを作成する．最後に，環境モデルと監視モデルを入力として DCS を実施し，環境下で安全性が保証された動作仕様を表す制御器を自動合成する．DCS は，モデル検査やソフトウェアテストにおいて繰り返し手動で行われていた動作仕様策定と検証を自動化することで，開発者の負担を軽減する．

しかしながら，DCS には計算空間が指数増加する課題があり，実践的な規模のシステム開発への適用を困難にしている．DCS は，制御器を合成する過程で環境モデルと監視モデルから安全性を満たす状態空間を分析するためのゲーム空間を構築する．このゲーム空間は，事象の同期を考慮した環境モデルと監視モデルの全状態の直積により状態を構築するため，それぞれのモデル数の増加に伴って状態数が指数的に増加する．このゲーム空間の指数増加に伴って，要求される計算空間，計算時間，必要主記憶量も指数増加するため，DCS において計算空間の状態削減は重要な課題となっている．

Consolidated Discrete Controller Synthesis（CDCS）[2] と Stepwise Discrete Controller Synthesis（SDCS）[3] は DCS のゲーム空間構築における状態空間の削減に取り組み，DCS における計算空間の指数爆発を抑制した．CDCS[2] では，ゲーム空間構築過程で安全性を違反すると判明した状態を抽象化しつつゲーム空間を構築する手法，Consolidated Game Composition を提案した．安全性ゲームを解くにあたってゲーム空間構築過程で安全性を違反すると判明した状態は探索する必要がない．よって，Consolidated Game Composition を用いてゲーム空間を構築することで，ゲーム空間構築過程で安全性を違反すると判明した状態の構築を回避し，計算空間の状態削減を実現した．SDCS[3] では，監視モデルごとに必要最小限の構成でゲーム空間の構築と分析を行う部分合成を提案した．そして，部分合成によって安全性を違反すると判明した状態を，他の監視モデルの安全性ゲームにおいて構築回避しつつ，段階的にゲーム空間を構築することで計算空間の状態削減を実現した．

これら CDCS と SDCS は，どちらも安全性を違反する状態に着目した DCS 計算空間削減手法であるが，どちらの手法が有効であるかは扱う環境モデルと監視モデルの組み合わせごとに異なる．そのため，開発者は対象システムに応じて CDCS と SDCS を適切に使い分ける必要があるが，判断するには状態爆発を引き起こす中間生成モデルの構造をすべて把握する必要があり，どちらが有効であるか判断することは現実的に困難である．

そこで本論文では，CDCS と SDCS における適用選択の課題を解消するために，SDCS と CDCS を一元化する DCS，Consolidated Stepwise Discrete Controller Synthesis（CSDCS）を提案する．一元化にあたって，CSDCS は，SDCS および CDCS のいずれと比較しても同等以上の適用可能範囲と状態空間削減効果を備える必要がある．そこで，SCDCS では，SDCS の部分合成を用いた段階的なゲーム空間構築アプローチを基盤としつつ，部分合成のゲーム空間構築処理に Consolidated Game Composition を導入することで，CDCS と SDCS の両手法を包含する適用可能範囲を実現した．また，Consolidated Game Composition の採用によって，高い計算空間削減効果を維持したまま，CSDCS における部分合成の回数を抑制することが可能となり，SDCS と CDCS の両方と比較しても短い時間で同等以上の削減効果を発揮することが可能になった．この CSDCS によって，従来必要であった SDCS と CDCS の使い分けに関する判断工程が不要となり，両者の適用選択における課題を解消する．

本論文の貢献は以下の通りである．

- CSDCS の合成アルゴリズムの定式化
- CSDCS が合成する制御器の妥当性証明
- CSDCS の計算空間削減効果の性能評価

CSDCS の性能評価では，SDCS や CDCS の評価で用いられた 7 つの離散事象システムにおいて制御器の合成実験を実施した．実験の結果，CSDCS は平均 43.6%の計算時間を削減しつつも，平均 51.8%計算空間削減効果を持つことがわかり，SDCS と CDCS の両方と同程度以上安定して高い削減効果量を持つことが確認された．結果，SDCS と CDCS の適用選択の課題を解消しうることがわかった．

本論文の構成は以下の通りである．2 章で背景技術である DCS について詳説する．続く 3 章では SDCS と CDCS がどのように DCS の計算空間を削減したかについて詳説し，SDCS と CDCS における適用選択の課題について示す．その後，適用選択の課題を解消する CSDCS の仕組みについて 4 章で詳説し，CSDCS が SDCS と CDCS における適用選択の課題を解消しうるかについて 5 章で評価実験を通して評価する．最後に，6 章で関連研究の説明の後，7 章で結論を述べる．

2. Discrete Controller Synthesis（DCS）

本章では，与えられた環境下で安全性を保証する制御器を自動合成する技術である DCS について詳説する．

2.1 DCS における事前準備

DCS を用いた動作仕様設計は次の手順で行われる．はじめに，開発者はシステムの動作環境を仮定し，その特性を環境モデル e としてモデル化する．次に，環境モデル e 下で保証すべき安全性を定義し，その安全性の充足状況を監視する監視モデル r を作成する．そして，環境モデル e と監視モデル r を入力として DCS を実施することで，環境下で安全性が保証された動作仕様を表す制御器 c を自動合成する．そのような DCS における e，r，c は次に定義される LTS を用いて表現される．

定義 1. LTS LTS は $x = (S_x, A_x, \Delta_x, s_x^0)$ で表現され

る．S_x は有限の状態集合であり，s_x^0 は初期状態である．$A_x = A^+ \cup A^-$ は事象の集合であり，A^+ はシステムが制御可能な事象，A^- はシステムが制御不可能な事象である．$(s, a, s') \in \Delta_x$ は遷移であり，事象 $a \in A_x$ によって状態 $s \in S_x$ は状態 $s' \in S_x$ へ遷移することを意味する．LTS の集合は大文字で X と表され，LTS の要素に下付き X と表記した S_X,A_X,Δ_X,s_X^0 は，X に含まれる全 LTS$x \in X$ の要素（S_x,A_x,Δ_x,s_x^0）の和集合を表す．

本論文では，LTS を区別するために，扱う全ての LTS にユニークな記号 id を付与する．よって，本論文で扱う環境モデルは決定性 LTS $e = (id_e, S_e, A_e, \Delta_e, s_e^0)$，監視モデルは決定性 LTS $r = (id_r, S_r, A_r, \Delta_r, s_r^0)$ で表され，それぞれ非決定性を持たない．また，環境モデルの集合は $e \in E$，監視モデルの集合は $r \in R$ で表される．このとき，DCS で与えられる E と R は常に次の関係式全てを満たす．

関係式 A $s_{err} \in S_r$

関係式 B $A_R \subset A_E$

関係式 C $\forall x, x' \in E \cup R : x \neq x' \Rightarrow S_x \cap S_{x'} = \emptyset$

関係式 A は，r の状態 S_r は安全性を違反する状態を表す違反状態 s_{err} を含むことを表す．関係式 B は，R は E を踏まえて設計されるため，$a \in A_R \backslash A_E$ となる事象 a は存在しないことを表す．関係式 C は，E と R に含まれるすべての LTS の状態は共通要素を持たないことを表す．

2.2 basic DCS

DCS は，関係式 A,B,C を満たす E と R を入力として，安全性が保証されたシステムの制御器 c を自動合成する．c は，決定性 LTS $c = (id_c, S_c, A_c, \Delta_c, s_c^0)$ で表され，システムを構成する全ての $e \in E$ において，どのように制御不可能な事象が生じたとしても，いずれの $r \in R$ の s_{err} に遷移することのない，システムの制御を表した LTS である．この性質から，c は r で表現された安全性を違反する状態に到達することのないシステムの動作仕様を表す．この c の導出する基本的な DCS（以降，basic DCS と呼ぶ）[4][5][6] は Algorithm1 で表される．

Algorithm 1 basic DCS

Input: E **such that** $\forall e \in E, e = (id_e, S_e, A_e, \Delta_e, s_e^0)$
Input: R **such that** $\forall r \in R, r = (id_r, S_r, A_r, \Delta_r, s_r^0)$
Output: c
1: $m \Leftarrow$ **Parallel Composition**(E)
2: $g \Leftarrow$ **Modified Parallel Composition**(m, R)
3: $g^* \Leftarrow$ **Error State Abstraction**(g)
4: $c \Leftarrow$ **Safety Game Solving**(g^*)
5: return c

Algorithm1 では，はじめに Parallel Composition を用いて，$e \in E$ からシステムの全環境がモデル化された決定性 LTS $m = (S_m, A_m, \Delta_m, s_m^0)$ を構築する（1 行目）．m を合成する Parallel Composition は定義 2 で定義される．

定義 2. Parallel Composition 記号 $\|$ で表され，複数の LTS を同期されたひとつの LTS に統合する．入力の LTS を $S_x \cap S_y = \emptyset$ を満たす $x = (S_x, A_x, \Delta_x, s_x^0)$, $y = (S_y, A_y, \Delta_y, s_y^0)$ とする．$m = x \parallel y = (S, A, \Delta, s^0)$ の時，初期状態 s^0 は状態の集合 $\{s_x^0, s_y^0\}$ で表され，遷移 $(s, a, s') \in \Delta$ は式 6，7，8 を満たす初期状態 s^0 から到達可能な全状態間を結ぶ全遷移の集合である．

$$\Delta_x\langle s_x \xrightarrow{a} s'_x\rangle, \Delta_y\langle s_y \overset{a}{\nrightarrow} s'_y\rangle, a \in A_x \backslash A_y \Rightarrow (\{s_x, s_y\}, a, \{s'_x, s_y\}) \in \Delta \quad (1)$$

$$\Delta_x\langle s_x \overset{a}{\nrightarrow} s'_x\rangle, \Delta_y\langle s_y \xrightarrow{a} s'_y\rangle, a \in A_y \backslash A_x \Rightarrow (\{s_x, s_y\}, a, \{s_x, s'_y\}) \in \Delta \quad (2)$$

$$\Delta_x\langle s_x \xrightarrow{a} s'_x\rangle, \Delta_y\langle s_y \xrightarrow{a} s'_y\rangle, a \in A_x \cap A_y \Rightarrow (\{s_x, s_y\}, a, \{s'_x, s'_y\}) \in \Delta \quad (3)$$

$\Delta_x\langle s_x \xrightarrow{a} s'_x\rangle$ は s_x において a が生じた場合，状態が s'_x となる遷移が Δ_x に含まれることを表し，$\Delta_x\langle s_x \overset{a}{\nrightarrow} s'_x\rangle$ は s_x において a が生じた場合，状態が s'_x となる遷移が Δ_x に含まれないことを表す．S は Δ に含まれる全状態の集合，A は Δ に含まれる全事象の集合で表され，Δ が合成されたとき一意に定まる．$E = \{e_1, e_2, \ldots, e_n\}$ の時，”**Parallel Composition**(E)”は $e_1 \parallel e_2 \parallel \cdots \parallel e_n$ を表す．

次に，Modified Parallel Composition を用いて，決定性 LTS であるゲーム空間 $g = (S_g, A_g, \Delta_g, s_g^0)$ を構築する（2 行目）．g は m において保証したい安全性 r を違反する状態を重ね合わせた LTS であり，g において二人型対戦ゲーム理論に基づいた安全性ゲームを解くことで制御器 c を導出できる．この g を合成する Modified Parallel Composition は定義 3 で定義される．

定義 3. Modified Parallel Composition プロセス記号 $\|_*$ で表され，監視モデル R とその監視対象である監視対象モデル m からゲーム空間 g を合成する．入力を $S_r \cap S_m = \emptyset$ を満たす $r = (S_r, A_r, \Delta_r, s_r^0) \in R$ と，$m = (S_m, A_m, \Delta_m, s_m^0)$ とする．$g = m \parallel_* r = (S, A, \Delta, s^0)$ の時，s^0 は $\{s_m^0, s_r^0\}$ で表され，遷移 $(s, a, s') \in \Delta$ は式 4，5 を満たす初期状態 s^0 から到達可能な全状態間を結ぶ全遷移の集合である．

$$\Delta_m\langle s_m \xrightarrow{a} s'_m\rangle, \Delta_r\langle s_r \overset{a}{\nrightarrow} s'_r\rangle, a \in A_m \backslash A_r \Rightarrow (\{s_m, s_r\}, a, \{s'_m, s_r\}) \in \Delta \quad (4)$$

$$\Delta_m\langle s_m \xrightarrow{a} s'_m\rangle, \Delta_r\langle s_r \xrightarrow{a} s'_r\rangle, a \in A_m \cap A_r \Rightarrow (\{s_m, s_r\}, a, \{s'_m, s'_r\}) \in \Delta \quad (5)$$

S は Δ に含まれる全状態の集合，A は Δ に含まれる全事象の集合で表され，Δ が合成されたとき一意に定まる．また，$m = \{S_m, A_m, \Delta_m, s_m^0\}$，$R = \{r_1, r_2, \ldots, r_n\}$ の時，"**Modified Parallel Composition**(m, R)"は $m \parallel_* r_1 \parallel_* r_2 \parallel_* \cdots \parallel_* r_n$ を表す．

その後，Error State Abstraction を用いて，g から決定性 LTS, $g^* = (S_{g^*}, A_{g^*}, \Delta_{g^*}, s_{g^*}^0)$ を構築する（3 行目）．g^* は s_{err} が重ね合わされた g の全状態をひとつの s_{err} に圧縮した LTS であり，g を g^* に一度変換することで Safety Game Solving における計算量が削減される．その Error State Abstraction は定義 4 で定義される．

定義 4. Error State Abstraction ゲーム空間 g において s_{err} を含む全ての状態をひとつの s_{err} に置換し，状態空間を圧縮する．入力のゲーム空間を $g = \{S_g, A_g, \Delta_g, s_g^0\}$，出力のゲーム空間を $g^* = \{S_{g^*}, A_{g^*}, \Delta_{g^*}, s_{g^*}^0\}$ とする．$s_{g^*}^0$ は $s_{g^*}^0 = s_g^0$ となり，S_{g^*} は $S_{g^*} = \{s_{err}\} \cup \{s_g \in S_g \mid s \notin s_{err}\}$ によって導出される．Δ_{g^*} は $\Delta_{g^*} = \{(s, a, s') \in \Delta_g \mid s \in S_{g^*}\}$ で導出されるが，導出の過程で Δ_{g^*} に含まれる s_{err} を含む全ての状態は s_{err} に置換される．また，A_g は $A_g = a \mid (s, a, s') \in \Delta_g$ と表される．以上の g から g^* を導出する作業を"**State Abstruction**(g)"で表す．

最後に，g^* において Safety Game Solving することで制御機 c を合成する（4 行目）．本論文における「分析」はこの Safety Game Solving を指し，g^* において，どのように制御不可能な事象が生じたとしても s_{err} に到達することのない全状態遷移を導出し c とする．Safety Game Solving は定義 5 で定義される．

定義 5. Safety Game Solving ゲーム空間 $g^* = (S_{g^*}, A_{g^*}, \Delta_{g^*}, s_{g^*}^0)$ を入力とし，制御器 c を出力する．二人型対戦ゲーム理論を用いて $s_{err} \in S_{g^*}$ から逆伝搬的に g^* の状態の探索を行い，A^+ をどのように発生させたとしても A^- 次第で s_{err} に遷移しうる状態である全ての状態を見つけ，S_{err} に分類する．その後，S_{g^*} のうち，S_{err} に含まれない全状態を状態の集合 S_{safe} に分類し，Δ_g のうち S_{safe} に含まれる状態のみで構築された遷移 Δ_{safe} を特定する．Δ_{safe} を構成する全ての事象を A_{safe} とする．特定した S_{safe}，Δ_{safe}，A_{safe} から制御器 $c = \{S_{safe}, A_c, \Delta_{safe}, s_{g^*}^0\}$ を構築し出力する．以上の g^* から c を導出する作業を"**Safety Game Solving**(g^*)"で表す．

以上の Parallel Composition，Modified Parallel Composition，Error State Abstraction，Safety Game Solving を通して c は合成される．このとき，c の導出過程で構築される m，g，g^* の状態数は入力される環境モデルの数に応じて指数増加する．その結果，DCS に必要となる主記憶量や計算時間も指数的に増大するため，DCS を実践規模のシステムへ適用していく上で，この計算空間の指数増加を抑制することは重要な課題だとされてきた [7][8][9]．この課題に対し，SDCS[3] と CDCS[2] は，c 合成過程の冗長な計算空間を特定し，構築を回避する DCS アルゴリズムを提案することで，計算空間の指数爆発の抑制に取り組んだ．

3. 計算空間の指数爆発抑制に取り組んだ DCS

本章では，basicDCS における計算空間の指数爆発の抑制に取り組んだ DCS，CDCS と SDCS について詳説する．

3.1 CDCS

CDCS[2] は，監視モデルの s_{err} に該当する全ての状態を，ひとつの s_{err} に抽象化しつつ，ゲーム空間構築することで，DCS の計算空間を削減する．安全性ゲームを解く上で，ゲーム空間の s_{err} から伸びる遷移は探索する必要がない．そこで，ゲーム空間構築過程で複雑な s_{err} の構築を回避することで，計算空間削減を実現する．CDCS のアルゴリズムを Algorithm2 に示す．

Algorithm 2 CDCS

Input: E **such that** $\forall e \in E, e = (id_e, S_e, A_e, \Delta_e, s_e^0)$
Input: R **such that** $\forall r \in R, r = (id_r, S_r, A_r, \Delta_r, s_r^0)$
Output: c
1: $g^* =$ **Consolidated Game Composition**($E \cup R$)
2: $c =$ **Safety Game Solving**(g^*)
3: return c

Algorithm2 では，Algorithm1 における Parallel Composition，Modified Parallel Composition，Error State Abstraction 全てを一括で行う Consolidated Game Composition を用いて，E と R から直接 g^* を構築する（1 行目）．その後，basic DCS と同様に g^* を入力として Safety Game Solving を用いて c を導出する．CDCS を実現する Consolidated Game Composition は定義 6 で定義される．

定義 6. Consolidated Game Composition 記号 $\between$ で表され，環境モデル群 E と監視モデル群 R に含まれる全ての LTS の集合 $E \cup R$ を入力とし，g^* を直接構築し，出力する．LTS $x, y \in E \cup R$ が入力される時，$x = (S_x, A_x, \Delta_x, s_x^0)$，$y = (S_y, A_y, \Delta_y, s_y^0)$ とする．$g^* = x \between y = (S, A, \Delta, s^0)$ の時，初期状態 s^0 は状態の集合 $\{s_x^0, s_y^0\}$ で表され，遷移 $(s, a, s') \in \Delta$ は式 6，7，8 を満たす初期状態 s^0 から到達可能な全状態間を結ぶ全遷移の集合である．

$$\Delta_x\langle s_x \xrightarrow{a} s'_x\rangle, \Delta_y\langle s_y \overset{a}{\nrightarrow} s'_y\rangle, a \in A_x \backslash A_y, s_{err} \notin s'_x \Rightarrow (\{s_x, s_y\}, a, \{s'_x, s_y\}) \in \Delta \quad (6)$$

$$\Delta_x\langle s_x \overset{a}{\nrightarrow} s'_x\rangle, \Delta_y\langle s_y \xrightarrow{a} s'_y\rangle, a \in A_y \backslash A_x, s_{err} \notin s'_y \Rightarrow (\{s_x, s_y\}, a, \{s_x, s'_y\}) \in \Delta \quad (7)$$

$$\Delta_x\langle s_x \xrightarrow{a} s'_x\rangle, \Delta_y\langle s_y \xrightarrow{a} s'_y\rangle, a \in A_x \cap A_y, s_{err} \notin s'_x \cup s'_y \Rightarrow (\{s_x, s_y\}, a, \{s'_x, s'_y\}) \in \Delta \quad (8)$$

$$\Delta_x\langle s_x \xrightarrow{a} s'_x\rangle, \Delta_y\langle s_y \overset{a}{\nrightarrow} s'_y\rangle, a \in A_x \backslash A_y, s_{err} \in s'_x \Rightarrow (\{s_x, s_y\}, a, s_{err}) \in \Delta \quad (9)$$

$$\Delta_x\langle s_x \overset{a}{\nrightarrow} s'_x\rangle, \Delta_y\langle s_y \xrightarrow{a} s'_y\rangle, a \in A_y \backslash A_x, s_{err} \in s'_y \Rightarrow (\{s_x, s_y\}, a, s_{err}) \in \Delta \quad (10)$$

$$\Delta_x\langle s_x \xrightarrow{a} s'_x\rangle, \Delta_y\langle s_y \xrightarrow{a} s'_y\rangle, a \in A_x \cap A_y, s_{err} \in s'_x \cup s'_y \Rightarrow (\{s_x, s_y\}, a, s_{err}) \in \Delta \quad (11)$$

S は Δ に含まれる全状態の集合，A は Δ に含まれる全事象の集合で表され，Δ が合成されたとき一意に定まる．以上の E と R から g^* を直接導出する作業を”**Consolidated Game Composition** ($E \cup R$)”で表す．

この Consolidated Game Composition を用いることによって，s_{err} が重ね合わされる全状態をひとつの s_{err} として圧縮しつつ，g^* を構築できる．これにより，s_{err} が重ね合わされる状態の数だけ計算空間が削減される．

3.2 SDCS

SDCS[3] は，部分合成を段階的に行うことで，DCS の計算空間を削減する．部分合成とは，システムの部分的な制御器を合成する工程を意味し，システムを構成する一部の環境モデルと監視モデルに Parallel Composition, Modified Parallel Composition，Error State Abstraction, Safety Game Solving を適用することで実現される．部分制御器を用いてゲーム空間の構築と分析を段階的に行うことで，部分違反状態（先の分析で安全性を違反すると判明した状態）の構築を以降全てのゲーム空間の構築・分析で回避でき，計算空間削減を実現する．SDCS のアルゴリズムを Algorithm3 に示す．

Algorithm 3 SDCS

Input: E **such that** $\forall e \in E, e = (id_e, S_e, A_e, \Delta_e, s_e^0)$
Input: R **such that** $\forall r \in R, r = (id_r, S_r, A_r, \Delta_r, s_r^0)$
Output: c
1: $\Lambda =$ **Algorithm 4**(E, R, 1)
2: **for** $i^* = 1$ to $n(\Lambda)$ **do**
3: $\lambda = (i^*, eids_\lambda, rids_\lambda, cid_\lambda), \in \Lambda$
4: $R^* = \{\forall r \in R \mid id_r \in rids_\lambda\}$
5: $E^* = \{\forall e \in E \mid A_e \cap A_{R^*} \neq \emptyset\}$
6: $m =$ **Parallel Composition**(E^*)
7: $g =$ **Modified Parallel Composition**(m, R^*)
8: $g^* =$ **Error State Abstraction**(g)
9: $c =$ **Safety Game Solving**(g^*)
10: $c^* = (cid_\lambda, S_c, A_c, \Delta_c, s_c^0)$
11: $R = R \backslash R^*, \quad E = E \backslash E^* \cup \{c^*\}$
12: **end for**
13: **return** $c^* \in E$

Algorithm3 では，Λ に則って，繰り返し部分合成（6-9 行目）を適用することで c を導出する．このとき，Λ は定義 7 で定義され，Algorithm4 で導出される（1 行目）[10].

定義 7. 合成シーケンス 合成シーケンス Λ は合成プロセス $\lambda = (i_\lambda, eids_\lambda, rids_\lambda, cid_\lambda)$ の集合によって表される．i_λ は Λ において何番目に実行する λ であるかを表し，1 から連続する整数が与えられる．$eids_\lambda$ は λ で入力される環境モデル（LTS）の id の集合，$rids_\lambda$ は λ で入力される監視モデル（LTS）の id の集合を表す．cid_λ は λ の出力となる部分制御器（LTS）c の id を表し，λ において c を合成した際には id_c にこの cid_λ を割り当てる．

Algorithm 4 監視対象モデル数を考慮した Λ の合成

Input: E **such that** $\forall e \in E, e = (id_e, S_e, A_e, \Delta_e, s_e^0)$
Input: R **such that** $\forall r \in R, r = (id_r, S_r, A_r, \Delta_r, s_r^0)$, i^*
Output: Λ
1: $\mu^* = \infty, \quad cid^* =$**uniqueID**
2: **for all** $r \in R$ **do**
3: $\mu = n(\{\forall e \in E \mid A_e \cap A_r \neq \emptyset\})$
4: **if** $\mu < \mu^*$ **then**
5: $r^* = r, \quad \mu^* = \mu$
6: **end if**
7: **end for**
8: $E^* = \{\forall e \in E \mid A_e \cap A_{R^*} \neq \emptyset\}$
9: $\lambda^* = \{(i^*, \{\forall id_e \mid (id_e, S_e, A_e, \Delta_e, s_e^0) \in E^*\}, \{id_{r^*}\}, cid^*)\}$
10: $c^* = (cid^*, \emptyset, A_{E^*}, \emptyset, \emptyset)$
11: **if** $n(R) = 1$ **then**
12: **return** $\{\lambda^*\}$
13: **end if**
14: **return** $\{\lambda^*\} \cup$ **Algorithm 4**($E \backslash E^* \cup \{c^*\}$, $R \backslash \{r^*\}$, $i^* + 1$)

Algorithm4 では，全監視モデルのうち μ が最小となる監視モデル $r \in R$ を r^* として導出し（2-7 行目），全環境モデルのうち r^* の事象を含む全ての環境モデルを E^* として導出する（8 行目）．そして，r^* と E^* から i^* ステップ目に部分合成する合成プロセスを構築する（9 行目）．その後，r^* と E^* を部分合成した際に合成される制御器 c^* を以降の合成プロセスの導出で必要な情報（id_{c^*}，A_{c^*}）だけ埋めて仮に構築し（10 行目），$i^* + 1$ ステップ目の合成プロセスの導出する（14 行目）．これを監視対象モデル数の数だけ繰り返し実施する（11-13 行目）．

以上のようにして導出された Λ に基づいて，Algorithm4 では i_λ が 1 の $\lambda \in \Lambda$ から順に繰り返し部分合成を行う（2-12 行目）．部分合成では，はじめに，入力された R のうち $rids_\lambda$ に id_r が含まれる r の集合を R^* とし（4 行目），R^* と E から部分合成の適用可能条件式 $A_r \subseteq A_{E^*} \backslash A_{E \backslash E^*}$ を満たす $e \in E$ の集合を E^* として導出する（5 行目）．その後，導出された R^* と E^* を用いて Parallel Composition（6 行目），Modified Parallel Composition（7 行目），Error State Abstraction（8 行目），Safety Game Solving（9 行目）を適用し c を合成する．そして，合成された c の id に cid_λ を割り当て c^* を用意する（10 行目）．最後に，次の λ で使用する R と E から使用した R^* と E^* を取り除き，c^* を E に加える（11 行目）．これにより，$i^* + 1$ ステップ目

以降の部分合成において，i^* ステップ目の R^* を違反する状態（部分違反状態）の構築を回避することができる．

3.3 CDCS と SDCS における適用選択の課題

CDCS や SDCS を用いて，basic DCS よりも効率よく制御器を合成するにあたり，開発者は以下の制約を考慮しつつ適切に CDCS や SDCS を使い分ける必要がある．

制約 1 CDCS または SDCS のいずれか一方のみが有効となる場合がある

制約 2 SDCS は basic DCS よりも計算時間が増加（性能が悪化）する可能性がある

制約 1 は，CDCS と SDCS がそれぞれ異なる方針で計算空間の削減を行うため，削減可能な状態が異なり，適用可能範囲が一致しないことに起因する．制約 2 は、SDCS において計算空間の削減効果が限定的な場合，段階的なゲーム空間 g^* の構築・分析を繰り返すオーバヘッドが相対的に大きくなり，全体の計算時間が悪化することに起因する．

以上の制約を踏まえて，CDCS と SDCS の適用選択は慎重に判断する必要がある．しかし，判断には，各手法により生成される中間生成 LTS，m，g，g^* の構造を詳細に把握する必要があり，これを入力される環境モデルと監視モデルの組み合わせに応じて手作業で行うことは実運用上の大きな障壁となっている．本論文では，この SDCS と CDCS の適用選択の課題を解消することを目的とする．

4. CSDCS（CDCS と SDCS の一元化）

CDCS と SDCS における適用選択の課題を解消するため，本論文では SDCS と CDCS を一元化した DCS，Consolidated Stepwise Discrete Controller Synthesis（CSDCS）を提案する．CSDCS の導入により，従来必要であった SDCS と CDCS の使い分けに関する判断工程が不要となり，両者の適用選択における課題を解消できる．CSDCS では SDCS と CDCS の一元化を実現するため，3.3 節の制約 1 と制約 2 を克服するように SDCS と CDCS を組み合わせる．

4.1 技術詳細

CSDCS では, SDCS（Algorithm3）を基盤にしつつ, g^* の構築に関わる 3 つの過程（Parallel Composition，Modified Parallel Composition，Error State Abstraction）を CDCS で導入された Consolidated Game Composition（定義 6）へと置換する．CDCS は g^* の構築過程を Consolidated Game Composition に統合することで s_{err} の抽象化を通じた計算空間削減を実現する．この Consolidated Game Composition は，SDCS において繰り返し行われるゲーム空間 g^* の構築処理にも適用可能であり，SDCS の段階的な計算空間削減アプローチと整合的に機能する．SDCS の構造に CDCS の手法を統合することにより，CDCS および SDCS の両手法が対象とする適用可能範囲を包含し，CSDCS は適用範囲の不一致に起因する制約 1 の克服を実現する．その CSDCS アルゴリズムが Algorithm5 である．

Algorithm 5 CSDCS

Input: E **such that** $\forall e \in E, e = (id, S, A, \Delta, s^0)$
Input: R **such that** $\forall r \in R, r = (id, S, A, \Delta, s^0)$
Output: c
1: $\Lambda =$ **Algorithm 6**(E, R)
2: **for** $i^* = 1$ to $n(\Lambda)$ **do**
3: $\lambda = (i^*, eids_\lambda, rids_\lambda, cid_\lambda) \in \Lambda$
4: $R^* = \{\forall r \in R \mid id_r \in rids_\lambda\}$
5: $E^* = \{\forall e \in E \mid A_e \cap A_{R^*} \neq \emptyset\}$
6: $g^* =$ **Consolidated Game Composition**($R^* \cup E^*$)
7: $c =$ **Safety Game Solving**(g^*)
8: $c^* = (cid_\lambda, S_c, A_c, \Delta_c, s_c^0)$
9: $R = R \backslash R^*$, $E = E \backslash E^* \cup \{c^*\}$
10: **end for**
11: **return** c

Algorithm5 では，はじめに，任意のヒューリスティクスに基づいて合成シーケンス Λ を導出する（1-2 行目）．その後，導出された Λ において i_λ が 1 の λ から順に繰り返し Consolidated Game Composition を用いた部分合成を行う（3-11 行目）．部分合成では，入力された R のうち $rids_\lambda$ に id_r が含まれる r の集合を R^* とし（5 行目），R^* と E から部分合成の適用可能条件式 $A_r \subseteq A_{E^*} \backslash A_{E \backslash E^*}$ を満たす $e \in E$ の集合を E^* として導出する（6 行目）．その後, 導出された R^* と E^* を入力として Consolidated Game Composition を実行することでゲーム空間 g^* を合成する（7 行目）．その後，g^* を入力として Safety Game Solving を実行することで部分制御器 c を合成し（8 行目），合成された c の id に cid_λ を割り当てる（9 行目）．最後に，次の λ で使用する R と E から使用した R^* と E^* を取り除き，cid_λ が割り当てられた部分制御器 c^* を E に加える（10 行目）．この部分合成を全ての Λ において i_λ の小さい方から順に繰り返し適用し，制御器 c を合成する．

このとき，Algorithm5 は，各監視モデルに対して個別に繰り返し部分合成を実施するため，SDCS と同様に basic DCS と比較して計算時間が増加する可能性（制約 2）を内在する．この計算時間の増加に対処するため，CSDCS では複数の監視モデルを対象とした部分合成手法を導入し，部分合成の繰り返し回数を削減する．

SDCS は, 部分合成の回数を減少させるほど, 各回で構築される状態空間が大規模化するため，計算空間削減性能が低下する．対して，Consolidated Game Composition は，同時に入力される監視モデルの数が増加するにつれて，抽象化される違反状態に該当する状態が増え，計算空間削減性能が向上する傾向を持つ．CSDCS では，Consolidated Game Composition のこの傾向を活用する．複数の監視モ

デルを対象とした部分合成に Consolidated Game Composition を適用することで，SDCS 特有の計算空間削減効果を損なうことなく計算時間の増加を抑制し，制約 2 の克服を実現する．複数の監視モデルを同時に部分合成するため，CSDCS では Algorithm5 における Λ 導出過程（1 行目）を改良する．$\lambda \in \Lambda$ の $rids_\lambda$ に複数の監視モデルの id を格納することで，同一ステップ内で複数の監視モデルを同時に対象とする部分合成が実現される．その CSDCS のための Λ 導出アルゴリズムが Algorithm6 である．

Algorithm 6 CSDCS に特化した Λ の統合

Input: E **such that** $\forall e \in E, e = (id_e, S_e, A_e, \Delta_e, s_e^0)$
Input: R **such that** $\forall r \in R, r = (id_r, S_r, A_r, \Delta_r, s_r^0)$
Output: Λ
1: $\Lambda =$ **Algorithm 4**(E, R, 1)，$i^{**} = 1$
2: $cids = \{\forall cid_\lambda \mid (i_\lambda, eids_\lambda, rids_\lambda, cid_\lambda) \in \Lambda\}$
3: **for** $i^* = 1$ to $n(\Lambda)$ **do**
4: $\lambda = (i^*, eids_\lambda, rids_\lambda, cid_\lambda) \in \Lambda$
5: **if** $n(eids_\lambda \cap cids) = 1$ **then**
6: $cid^* = \exists id \in (eids_\lambda \cap cids)$
7: $\lambda^* = (i_{\lambda^*}, eids_{\lambda^*}, rids_{\lambda^*}, cid^*) \in \Lambda$
8: $eids^* = eids_\lambda \cup eids_{\lambda^*} \backslash \{cid^*\}$
9: $rids^* = rids_\lambda \cup rids_{\lambda^*}$
10: $\Lambda = \Lambda \backslash \{\lambda, \lambda^*\} \cup \{i_{\lambda^*}, eids^*, rids^*, cid_\lambda\}$
11: **else**
12: $\Lambda = \Lambda \backslash \{\lambda\} \cup \{i^{**}, eids_\lambda, rids_\lambda, cid_\lambda\}$
13: $i^{**} = i^{**} + 1$
14: **end if**
15: **end for**
16: **return** Λ

Algorithm6 では，Algorithm4 を用いて各監視モデルごとに部分合成する Λ を導出する（1 行目）．その後，Λ で合成される全ての部分制御器の cid を $cids$ として用意し（2 行目），i_λ が 1 の $\lambda \in \Lambda$ から順に $eids_\lambda$ に cid がいくつ含まれるか確認する（3-5 行目）．$eids_\lambda$ に含まれる cid が 1 つの時（5 行目），その唯一の cid を合成する λ^* と λ を統合する（6-10 行目）．統合された λ は $\{i_{\lambda^*}, eids^*, rids^*, cid_\lambda\}$ で表さる．$eids^*$ は $eids_\lambda$ に含まれる cid_{λ^*} と eid_{λ^*} を置き換えた id の集合で表される（8 行目）．$rids^*$ は $rids_\lambda$ と rid_{λ^*} の和集合で表される（9 行目）．その後，統合された λ と，統合元となった λ と λ^* を，Λ において置き換える（10 行目）．$eids_\lambda$ に含まれる cid が 2 つ以上，もしくは cid が含まれない場合は，統合で減った λ に合わせて i_λ を振り直す（11-14 行目）．以上の工程を全ての $\lambda \in \Lambda$ において繰り返し実行し，$n(eids_\lambda \cap cids) = 1$ である λ の統合を行う．

このとき，$n(eids_\lambda \cap cids)$ は，λ の部分合成で入力される環境モデル E^* に含まれる部分制御器（以前の部分合成で合成された c^*）の数を表す．部分制御器には，その部分制御器の部分合成過程で安全性を違反すると判明した部分違反状態は含まれない．部分制御器を E^* に含む部分合成では，g^* 構築時であっても，部分制御器の部分違反状態は重ね合わされないため，構築回避され，状態削減される．すると，複数の部分制御器を E^* に含む部分合成ほど，重ね合わせによって状態削減される状態数は指数的に増加するため，Algorithm6 において，λ の部分合成で削減される状態数は $n(eids_\lambda \cap cids)$ に対して指数増加する傾向がある．

そこで，CSDCS では，この $n(eids_\lambda \cap cids)$ の数に着目し，部分合成による計算空間削減効果が特に大きくなる $n(eids_\lambda \cap cids) >= 2$ の時のみ部分合成を適用する．そして，比較的効果の小さい $n(eids_\lambda \cap cids) = 1$ の時は Consolidated Game Composition が効果的に働くように複数の監視モデルを同時に部分合成することで，高い計算空間削減効果を維持したまま，Algorithm6 における部分合成回数を抑制した．

4.2 機能の妥当性証明

機能の妥当性証明として，CSDCS が basic DCS と常に同じ制御器を合成することを命題 1 の証明で示す．

命題 1. 1 以上の自然数 i と j において，環境モデル $E = \{e_1, e_2, \ldots, e_i\}$，監視モデル $R = \{r_1, r_2, \ldots, r_j\}$ が与えられるとき，**Algorithm 1**(E, R) の出力 c と，**Algorithm 5**(E, R) の出力 c は同じである．

証明. 同じ E と R が与えられるとき，Algorithm3（SDCS）と Algorithm1（basic DCS）は同じ c を合成する [3]．よって，式 12 が成り立つ．

$$\mathbf{Algorithm3}(\mathbf{E}, \mathbf{R}) = \mathbf{Algorithm1}(\mathbf{E}, \mathbf{R}) \quad (12)$$

次に，Algorithm3（SDCS）と Algorithm5（CSDCS）は同じ c を合成することを証明する．Algorithm5 では，Consolidated Game Composition，Algorithm3 では Parallel Composition，Modified Parallel Composition，Error State Abstraction の 3 つを用いて g^* を合成する．同じ E^* と R^* が与えられるとき，Consolidated Game Composition によって合成される g^* と，Parallel Composition，Modified Parallel Composition，Error State Abstraction の 3 つを通して合成される g^* は同じである [2]．よって，Algorithm5 と Algorithm3 で合成される g^* は同じである．すると，g^* から c を導出する過程は同じであるため，同じ E と R が与えられるとき Algorithm5 と Algorithm3 は同じ c を合成する．よって，式 13 が成り立つ．

$$\mathbf{Algorithm5}(\mathbf{E}, \mathbf{R}) = \mathbf{Algorithm3}(\mathbf{E}, \mathbf{R}) \quad (13)$$

すると，式 13 と式 12 から式 14 が成り立つ．

$$\begin{aligned} \mathbf{Algorithm5}(\mathbf{E}, \mathbf{R}) &= \mathbf{Algorithm3}(\mathbf{E}, \mathbf{R}) \\ &= \mathbf{Algorithm1}(\mathbf{E}, \mathbf{R}) \end{aligned} \quad (14)$$

式 14 より，同じ E と R が与えられるとき，Algorithm5 と Algorithm1 で合成される c は同じであるため，本命題は成り立つ． □

5. 評価

CSDCS の有効性を評価するために，本論文では以下の Reserch Question（RQ）を設定し評価実験を行う．

RQ1 CSDCS はどれほどの削減効果を持つか
RQ2 CDCS と同等以上の削減効果を持つか
RQ3 SDCS と同等以上の削減効果を持つか
RQ4 SDCS と CDCS の適用選択の課題を解消できるか

RQ1 では，basic DCS を通して CSDCS の各削減効果とその傾向を評価する．RQ2 と RQ3 では，CSDCS の各削減効果が CDCS と SDCS と比較して優位であるか相対的に評価する．RQ4 では，CSDCS が SDCS と CDCS の適用選択の課題を解消しうるか評価する．

5.1 実験設定

CDCS を評価するにあたって，本論文では basic DCS，SDCS，CDCS，CSDCS の 4 つの DCS 手法で制御器の合成実験を実施し，その際に構築された最大計算空間の状態数 $|S|$，遷移数 $|\Delta|$，合成に必要となった必要主記憶量 M，必要計算時間 T を計測した．

また，N（環境モデルの数が変化するパラメータ）と K（環境モデルの状態数が変化するパラメータ）が操作できるスケーラブルな 7 つの離散事象システムを対象に制御器の合成実験を行った．Headcount Control（HC）[11] は，複数の部屋の人数を管理することを目的としたシステムであり，入室の許可/拒否を制御する．N には制御する部屋の数，K には管理可能なひと部屋あたりの最大収容人数が該当する．Auto Warehouse（AW）[12] は倉庫の荷物運搬業務の自動化を目的としたシステムであり，同じ機能を持つ荷物運搬ロボットを在庫管理や出荷作業など複数の役割に分けつつ協調制御する．N には制御するロボットの数，K には在庫管理する棚の数が該当する．Air Traffic（AT）[13] は，空路と滑走路の混雑管理を目的としたシステムであり，複数の飛行機の離着陸，空路での待機，空路の移動を制御する．N には制御する飛行機の数，K には管理可能な空路の数が該当する．Bidding Workflow（BW）[13] は，申請書ごとの複雑な承認手続きの自動管理を目的としたシステムであり，受け取った申請書の提出先・提出順を制御する．N は承認をおこなう提出先となる管理部署数，K は申請が否認された場合に再申請できる回数 K が該当する．Cat & Mouse（CM）[13] は，制限されたフィールド上で自由に動く捕獲対象の捕獲を目的とするシステムであり，捕獲ロボットの移動を制御する．N は捕獲ロボットの数，K は捕獲対象が逃げ回るフィールドの区画数が該当する．Drone Control（DC）[14][15] は，制限されたフィールドを巡回監視する警備ドローンの飛行管理を目的としたシステムであり，複数のドローンの移動，充電を制御する．N は管理するドローンの数，K はフィールドの区画数が該当する．Access Control（AC）[16] は，複数サーバ内の情報閲覧をアクセス元に応じて管理することを目的としたシステムであり，管理サーバの情報開示処理，情報加工処理，応答処理を管理する．N も K もシステムが管理するサーバ数が該当し，本シナリオでは常に同じ値を取る．

実験には，OS が Windows 10 Pro (64bit)，CPU が Intel Xeon W-2265，RAM が 256GB(64GB × 4) の計算機で，制御器合成ツール Modal Transition Synstem Analyzer[17] を使用した．以上の実験設定で評価実験をおこなった結果が，表 1 と表 2 である．

5.2 実験結果と評価

5.2.1 CSDCS はどれほどの削減効果を持つか（RQ1）

表 1 と表 2 より，CSDCS は basic DCS に対して $|S|$ は平均-51.8%，$|\Delta|$ は平均-58.3，M は平均-45.1%，T は平均-43.6%の削減効果が確認された．$|S|$，$|\Delta|$，M，T 全てにおいて，CSDCS は常に basic DCS と同じかそれ以下となることが確認でき，basic DCS より効率的に制御器を合成できることが確認された．

5.2.2 CDCS と同等以上の削減効果を持つか（RQ2）

適用可能範囲について，CDCS で計算空間削減効果が確認された HC，AW（一部），AT，CM，DC の全ての場合において CSDCS でも計算空間削減効果が確認された．また，CSDCS は CDCS で削減効果が確認されなかった AW（一部），AC においても計算空間削減効果が確認された．以上の結果から，表 1 と表 2 において CSDCS は CDCS の適用可能範囲を包括しつつも，より広い適用可能範囲を持つことがわかる．

削減効果量について，CSDCS は CDCS と比較して $|S|$ で平均-28.4%，$|\Delta|$ で平均-21.9%，M で平均-21.7%，T で平均-13.5%の性能向上が確認された．削減効果の標準偏差は，SDCS と比較して $|S|$ が-1.6，$|S|$ が-3.1 だけ CSDCS が低い値となっており，CSDCS は CDCS より安定して高い削減効果を $|S|$ と $|\Delta|$ において発揮できたことが確認された．M と T の標準偏差は，M が+4.5，T が+2.1 と CSDCS よりも CDCS の方が削減効果が安定していることが確認されたが，CSDCS の方が削減効果の平均値が M で-22.0%，T で-13.5%だけ高い性能を上げており，性能差に比べると標準偏差の差は小さい．以上の結果から，表 1 と表 2 において CSDCS は CDCS より高い削減効果を同程度安定して発揮できていることがわかる．

以上より，表 1 と表 2 において，CSDCS は CDCS 以上の適用可能範囲を持ちつつ，より高い削減効果量を安定して発揮していることがわかった．この結果より，CSDCS は CDCS 以上の削減効果を持つと推察される．

5.2.3 SDCS と同等以上の削減効果を持つか（RQ3）

適用可能範囲について，SDCS で計算空間削減効果が確認された HC，AW，DC，AC の全ての場合において CSDCS

表 1 DCS における最大の状態数と遷移数 (削減率:%)
Table 1 Maximum number of states and transitions in DCS (Reduction rate: %)

			basic DCS		CDCS		SDCS		DCDCS	
	N	K	$\|S\|$	$\|\Delta\|$	$\|S\|$	$\|\Delta\|$	$\|S\|$	$\|\Delta\|$	$\|S\|$	$\|\Delta\|$
HC	3	4	14.1	61.6	***0.66(-95.3)**	***2.96(-95.2)**	3.68(-73.9)	6.90(-88.8)	***0.66(-95.3)**	***2.96(-95.2)**
	4	4	277	1514	***4.07(-98.5)**	***22.5(-98.5)**	23.1(-91.7)	45.0(-97.0)	***4.07(-98.5)**	***22.5(-98.5)**
	5	4	2016	12912	***10.5(-99.5)**	***69.0(-99.5)**	60.9(-97.0)	125(-99.0)	***10.5(-99.5)**	***69.0(-99.5)**
	6	4	23200	172000	***39.7(-99.8)**	***302(-99.8)**	233(-99.0)	492(-99.7)	***39.7(-99.8)**	***302(-99.8)**
	5	5	5212	33413	***10.5(-99.8)**	***69.0(-99.8)**	76.8(-98.5)	158(-99.5)	***10.5(-99.8)**	***69.0(-99.8)**
	5	6	11569	74220	***10.5(-99.9)**	***69.0(-99.9)**	92.7(-99.2)	191(-99.7)	***10.5(-99.9)**	***69.0(-99.9)**
	5	7	23003	147653	***10.5(-99.9)**	***69.0(-99.9)**	109(-99.5)	223(-99.8)	***10.5(-99.9)**	***69.0(-99.9)**
AW	2	2	6.40	152	6.40(±0.0)	152(±0.0)	6.40(±0.0)	***14.7(-90.3)**	6.40(±0.0)	19.1(-87.4)
	3	2	104	3267	104(±0.0)	3267(±0.0)	107(+3.7)	***359(-89.0)**	104(±0.0)	431(-86.8)
	4	2	670	25925	670(±0.0)	25925(±0.0)	670(±0.0)	***2813(-89.2)**	670(±0.0)	3108(-88.0)
	2	10	113	4734	***52.5(-53.5)**	2112(-55.4)	***52.5(-53.5)**	***139(-97.1)**	***52.5(-53.5)**	173(-96.3)
	2	20	1115	69578	***157(-86.0)**	9470(-86.4)	172(-84.6)	***523(-99.2)**	***157(-86.0)**	543(-99.2)
	2	30	4735	391001	***312(-93.4)**	25170(-93.6)	469(-90.1)	***469(-99.9)**	***312(-93.4)**	1109(-99.7)
AT	3	5	0.70	2.55	***0.51(-27.3)**	***1.91(-25.0)**	0.70(±0.0)	2.55(±0.0)	***0.51(-27.3)**	***1.91(-25.0)**
	4	5	6.14	27.4	***3.24(-47.3)**	***14.7(-46.3)**	6.14(±0.0)	27.4(±0.0)	***3.24(-47.3)**	***14.7(-46.3)**
	5	5	53.2	276	***18.3(-65.6)**	***94.6(-65.8)**	53.2(±0.0)	276(±0.0)	***18.3(-65.6)**	***94.6(-65.8)**
	2	2	0.04	0.07	***0.03(-14.3)**	***0.06(-13.7)**	0.04(±0.0)	0.07(±0.0)	***0.03(-14.3)**	***0.06(-13.7)**
	2	3	0.05	0.11	***0.04(-12.5)**	***0.10(-10.8)**	0.05(±0.0)	0.11(±0.0)	***0.04(-12.5)**	***0.10(-10.8)**
	2	4	0.06	0.16	***0.06(-11.1)**	***0.14(-8.9)**	0.06(±0.0)	0.16(±0.0)	***0.06(-11.1)**	***0.14(-8.9)**
BW	2	5	0.14	0.37	0.14(±0.0)	0.37(±0.0)	0.14(±0.0)	0.37(±0.0)	0.14(±0.0)	0.37(±0.0)
	3	5	1.66	6.40	1.66(±0.0)	6.40(±0.0)	1.66(±0.0)	6.40(±0.0)	1.66(±0.0)	6.40(±0.0)
	4	5	19.3	99.7	19.3(±0.0)	99.7(±0.0)	19.3(±0.0)	99.7(±0.0)	19.3(±0.0)	99.7(±0.0)
	5	3	25.8	158	25.8(±0.0)	158(±0.0)	25.8(±0.0)	158(±0.0)	25.8(±0.0)	158(±0.0)
	5	4	85.3	542	85.3(±0.0)	542(±0.0)	85.3(±0.0)	542(±0.0)	85.3(±0.0)	542(±0.0)
	5	5	222	1446	222(±0.0)	1446(±0.0)	222(±0.0)	1446(±0.0)	222(±0.0)	1446(±0.0)
CM	2	2	3.93	10.4	***2.70(-31.2)**	***7.09(-31.8)**	3.93(±0.0)	10.4(±0.0)	***2.70(-31.2)**	***7.09(-31.8)**
	3	2	161	592	***66.8(-58.5)**	***240(-59.5)**	161(±0.0)	592(±0.0)	***66.8(-58.5)**	***240(-59.5)**
	2	3	15.9	44.9	***12.1(-24.3)**	***33.8(-24.9)**	15.9(±0.0)	44.9(±0.0)	***12.1(-24.3)**	***33.8(-24.9)**
	2	4	45.1	132	***36.3(-19.6)**	***105(-20.0)**	45.1(±0.0)	132(±0.0)	***36.3(-19.6)**	***105(-20.0)**
	2	5	103	308	***86.3(-16.4)**	***256(-16.7)**	103(±0.0)	308(±0.0)	***86.3(-16.4)**	***256(-16.7)**
DC	2	2	4.62	23.9	0.78(-83.0)	3.94(-83.6)	0.78(-83.0)	2.86(-88.1)	***0.50(-89.3)**	***1.80(-92.5)**
	3	2	314	2441	12.4(-96.1)	88.2(-96.4)	53.3(-83.0)	297(-87.8)	***9.15(-97.1)**	***50.2(-97.9)**
	4	2	21381	221360	119(-99.4)	1093(-99.5)	3625(-83.0)	27174(-87.7)	***98.0(-99.5)**	***714(-99.7)**
	2	5	35.3	416	16.5(-53.2)	416(±0.0)	***6.67(-81.1)**	***160(-61.5)**	10.4(-70.7)	258(-38.0)
	2	6	52.0	952	32.6(-37.2)	952(±0.0)	***12.9(-75.2)**	***361(-62.1)**	20.5(-60.6)	592(-37.7)
	2	7	71.8	1884	56.8(-20.9)	1884(±0.0)	***22.0(-69.3)**	***705(-62.6)**	35.7(-50.3)	1176(-37.5)
AC	2	5	0.53	1.70	0.53(±0.0)	1.70(±0.0)	***0.25(-52.4)**	***0.72(-57.7)**	***0.25(-52.4)**	***0.72(-57.7)**
	3	5	12.2	58.7	12.2(±0.0)	58.7(±0.0)	***3.53(-71.0)**	***14.9(-74.7)**	***3.53(-71.0)**	***14.9(-74.7)**
	4	5	280	1801	280(±0.0)	1801(±0.0)	***49.4(-82.3)**	***275(-84.7)**	***49.4(-82.3)**	***275(-84.7)**
	5	2	161	1171	161(±0.0)	1171(±0.0)	***36.9(-77.1)**	***233(-80.1)**	***36.9(-77.1)**	***233(-80.1)**
	5	3	759	5822	759(±0.0)	5822(±0.0)	***120(-84.2)**	***794(-86.4)**	***120(-84.2)**	***794(-86.4)**
	5	4	2476	19548	2476(±0.0)	19548(±0.0)	***311(-87.4)**	***2115(-89.2)**	***311(-87.4)**	***2115(-89.2)**
全体	平均		-	-	-23.4	-36.4	-43.2	-51.7	-51.8	-58.3
	標準偏差		-	-	40.0	41.3	42.4	42.4	38.4	38.2

※ N：環境モデル数，K：環境モデルサイズ，$|S|$：最大状態数（K），$|\Delta|$：最大遷移数（K），「*」付数字は削減効果が最大の結果

でも計算空間削減効果が確認された．また，CSDCS は SDCS で削減効果が確認されなかった AT，CM においても計算空間削減効果が確認された．以上の結果から，表 1 と表 2 において CSDCS は SDCS の適用可能範囲を包括しつつも，より広い適用可能範囲を持つことがわかる．

削減効果量について，CSDCS は SDCS と比較して $|\Delta|$ で平均-4.0%，M で平均-57.0%，T で平均-553.6%の性能向上が確認された．$|S|$ は平均+8.6%と性能が悪化したが，削減率の標準偏差は SDCS と比較して全て CSDCS が低い値となっており，CSDCS は SDCS より安定して高い削減率を発揮できたことが確認された．特に T の削減率は，SDCS では標準偏差 953.3 とシステムによって大きくばらついていたが，CSDCS はそのばらつきを標準偏差 36.0 まで抑制できていることが確認された．以上の結果から，表 1 と表 2 において CSDCS は SDCS より高い削減効果量をより安定して発揮できていることがわかる．

表 2 DCS における主記憶量と計算時間 (削減率:%)
Table 2 Memory and computation time in DCS (Reduction rate: %)

			basic DCS		CDCS		SDCS		DCDCS	
	N	K	M	T	M	T	M	T	M	T
HC	3	4	49.2	1.32	***32.9(-33.2)**	***1.04(-21.7)**	49.6(+0.8)	8.53(+545.9)	***32.8(-33.4)**	***1.02(-22.8)**
	4	4	244	5.98	***67.9(-72.1)**	***1.68(-72.0)**	259(+6.3)	28.5(+377.1)	***67.2(-72.4)**	***1.68(-72.0)**
	5	4	1908	22.4	***142(-92.6)**	***2.56(-88.5)**	635(-66.7)	60.8(+171.9)	***142(-92.5)**	***2.53(-88.7)**
	6	4	23984	1998	***515(-97.9)**	***5.42(-99.7)**	2326(-90.3)	223(-88.9)	***513(-97.9)**	***5.53(-99.7)**
	5	5	5408	87.5	***141(-97.4)**	***2.56(-97.1)**	759(-86.0)	71.9(-17.8)	***143(-97.4)**	***.59(-97.0)**
	5	6	10483	410	***144(-98.6)**	***2.50(-99.4)**	938(-91.1)	81.0(-80.2)	***143(-98.6)**	***2.60(-99.4)**
	5	7	20998	1623	***143(-99.3)**	***2.45(-99.8)**	1094(-94.8)	91.1(-94.4)	***143(-99.3)**	***2.58(-99.8)**
AW	2	2	257	5.74	250(-2.5)	4.34(-24.4)	127(-50.5)	44.2(+669.5)	***84.5(-67.1)**	***3.93(-31.5)**
	3	2	4710	32.4	4638(-1.5)	27.7(-14.4)	1372(-70.9)	109(+237.7)	***955(-79.7)**	***18.1(-44.2)**
	4	2	36156	362	35986(-0.5)	319(-12.0)	7850(-78.3)	599(+65.4)	***6388(-82.3)**	***198(-45.3)**
	2	10	3246	31.7	3018(-7.0)	19.2(-39.6)	1278(-60.6)	240(+657.6)	***461(-85.8)**	***9.90(-68.8)**
	2	20	16756	261	13245(-21.0)	73.8(-71.7)	7148(-57.3)	1276(+389.4)	***1505(-91.0)**	***23.5(-91.0)**
	2	30	84530	2237	33712(-60.1)	210(-90.6)	22092(-73.9)	6024(+169.3)	***3292(-96.1)**	***60.1(-97.3)**
AT	3	5	33.1	1.06	***30.4(-8.3)**	***0.91(-13.8)**	40.0(+20.7)	8.12(+669.7)	***30.4(-8.3)**	***0.90(-14.5)**
	4	5	59.9	1.73	***54.3(-9.2)**	***1.44(-16.8)**	252(+321.3)	48.1(+2682.0)	***54.0(-9.8)**	***1.47(-15.3)**
	5	5	225	4.31	***198(-11.9)**	***3.16(-26.6)**	2030(+801.7)	251(+5726.6)	***201(-10.7)**	***3.21(-25.5)**
	2	2	29.0	0.42	***27.4(-5.6)**	***0.40(-5.3)**	27.3(-5.8)	1.37(+226.3)	***27.0(-6.9)**	***0.40(-5.0)**
	2	3	29.1	0.51	***27.1(-6.8)**	***0.46(-10.2)**	27.5(-5.5)	1.83(+260.1)	***27.1(-7.0)**	***0.47(-7.9)**
	2	4	29.8	0.60	***27.6(-7.4)**	***0.53(-12.3)**	28.9(-3.1)	2.43(+302.0)	***27.6(-9.8)**	***0.54(-11.3)**
BW	2	5	29.7	0.33	***29.2(-1.7)**	***0.31(-7.6)**	***29.2(-1.8)**	0.62(+86.7)	***29.2(-1.7)**	***0.30(-7.9)**
	3	5	40.0	0.67	***39.2(-1.9)**	***0.63(-4.8)**	45.1(+12.8)	1.52(+128.6)	***39.2(-1.8)**	***0.63(-5.9)**
	4	5	221	2.69	***215(-2.5)**	***2.62(-2.8)**	265(+19.8)	9.34(+247.1)	***215(-2.6)**	***2.65(-1.7)**
	5	3	320	3.73	***313(-2.3)**	***3.57(-4.2)**	423(+32.1)	12.8(+242.7)	***312(-2.6)**	***3.57(-4.3)**
	5	4	1032	12.6	***1005(-2.7)**	***12.1(-4.3)**	1369(+32.7)	40.9(+224.1)	***1004(-2.7)**	***12.0(-4.6)**
	5	5	2697	39.6	***2617(-3.0)**	***37.7(-4.7)**	3627(+34.5)	135(+241.8)	***2617(-3.0)**	***37.6(-5.0)**
CM	2	2	51.5	0.77	***40.0(-22.4)**	***0.72(-7.0)**	49.1(-4.8)	3.67(+375.5)	***40.0(-22.4)**	***0.72(-7.0)**
	3	2	521	5.09	***466(-10.6)**	***3.87(-24.1)**	1199(+130.0)	52.5(+931.5)	***467(-10.4)**	***3.78(-25.8)**
	2	3	99.2	1.67	***93.5(-5.7)**	***1.49(-10.6)**	138(+39.4)	15.1(+804.1)	***93.1(-6.1)**	***1.44(-13.3)**
	2	4	255	3.00	***237(-6.8)**	***2.50(-16.0)**	386(+51.5)	42.7(+1324.1)	***239(-6.2)**	***2.51(-16.2)**
	2	5	586	5.84	***543(-7.4)**	***4.85(-16.9)**	910(+55.3)	104(+1683.3)	***544(-7.2)**	***4.85(-17.0)**
DC	2	2	35.5	1.04	32.3(-9.2)	0.86(-16.9)	34.2(-3.7)	4.11(+296.2)	***30.5(-14.2)**	***0.88(-15.6)**
	3	2	357	6.99	168(-52.9)	2.15(-69.3)	666(+86.5)	36.1(+416.5)	***116(-67.6)**	***2.20(-68.6)**
	4	2	25210	5347	1680(-93.3)	14.0(-99.7)	52110(+106.7)	13252(+147.9)	***1135(-95.5)**	***9.78(-99.8)**
	2	5	664	5.51	650(-2.2)	4.91(-10.9)	651(-2.0)	34.9(+532.3)	***428(-35.6)**	***3.95(-28.3)**
	2	6	1435	10.4	1406(-2.0)	9.59(-8.2)	1406(-2.0)	68.9(+559.7)	***930(-35.2)**	***7.26(-30.5)**
	2	7	2793	18.4	2762(-1.1)	18.1(-1.9)	2745(-1.7)	124(+575.1)	***1824(-34.7)**	***12.7(-31.3)**
AC	2	5	30.4	0.47	29.8(-1.8)	0.42(-9.4)	***28.5(-6.3)**	0.92(+97.4)	***28.3(-6.9)**	***0.42(-9.9)**
	3	5	150	2.09	145(-3.2)	1.79(-14.4)	***61.0(-59.3)**	1.84(-11.9)	***61.6(-58.9)**	***1.16(-44.3)**
	4	5	3639	53.9	3506(-3.7)	51.0(-5.3)	***593(-83.7)**	***7.69(-85.7)**	***593(-83.7)**	***6.7(-87.6)**
	5	2	2247	26.4	2156(-4.0)	24.9(-5.8)	***468(-79.2)**	6.79(-74.3)	***469(-79.1)**	***5.41(-79.5)**
	5	2	10344	287	9875(-4.5)	277(-3.6)	***1577(-84.8)**	***20.1(-93.0)**	***1572(-84.8)**	***18.1(-93.7)**
	5	2	35025	2528	33436(-4.5)	2555(+1.0)	***4101(-88.3)**	***69.8(-97.2)**	***4095(-88.3)**	***67.6(-97.3)**
全体	平均		-	-	-23.4	-30.1	+11.9	+510.0	-45.1	-43.6
	標準偏差		-	-	33.6	34.2	146.7	964.6	38.2	36.2

※ N：環境モデル数，K：環境モデルサイズ，M：必要主記憶量（MB），T：計算時間（s），「*」付数字は削減効果が最大の結果

以上より，表 1 と表 2 において，CSDCS は SDCS 以上の適用可能範囲を持ちつつ，より高い削減効果量を安定して発揮していることがわかった．この結果より，CSDCS は SDCS 以上の削減効果を持つと推察される．

5.2.4 適用選択の課題を解消できるか（RQ4）

本課題を解決する上で，CSDCS は SDCS と CDCS の両方の適用可能範囲を包括し，SDCS と CDCS と同等以上の削減効果量を持つ必要がある．

適用可能範囲について，表 1 と表 2 において，CSDCS は SDCS と CDCS のどちらかにおいて計算空間削減効果を持つ場合，常に CSDCS も空間削減効果を持つことが RQ2 と RQ3 の実験結果よりわかった．この結果から，CSDCS は SDCS と CDCS の適用可能範囲を包括すると考えられる．

削減効果量に関して，表 1 と表 2 において，CSDCS は SDCS と CDCS のどちらよりも削減効果（削減率）の平均値が高いことが RQ2 と RQ3 の実験結果よりわかった．ま

た，CSDCS の標準偏差は，SDCS より低い値，CDCS と同程度であることが RQ2 と RQ3 の実験結果よりわかった．この結果から，CSDCS は SDCS と CDCS より高い削減効果量を同程度以上安定して発揮できると考えられる．以上より，CSDCS は，SDCS と CDCS の適用可能範囲を包括し，SDCS と CDCS と同程度以上安定して高い削減効果量を持つと推察する．よって CSDCS は，SDCS と CDCS の適用選択の課題を解決しうると考える．

5.2.5 議論

本実験を通して，CSDCS は SDCS と CDCS の適用選択の課題を解決しうることだけでなく，CSDCS が SDCS と CDCS 以上の計算空間削減効果を持ちうることも確認された．表 1 の DC では，SDCS と CDCS のどちらよりも高い $|S|$ と $|\Delta|$ の削減効果が確認された．本結果は，DC において，SDCS と CDCS それぞれのアプローチでないと削減できない計算空間の状態・遷移が存在し，CSDCS がそれらを同時に削減できたためだと考える．将来研究として，SDCS と CDCS をよりうまく組みあわせ，削減効果をさらに向上させることができないかについて調査する．

また，本実験では SDCS と CDCS と両方において計算空間削減効果を持たない場合，CSDCS が空間削減効果を持つことはなかった．将来研究として，CSDCS が SDCS と CDCS 以上の適用可能範囲を持ちうることはないか追加実験や証明を通して調査する．

5.3 妥当性への脅威

本実験では，$|S|$, $|\Delta|$ の計測に関して，basic DCS, SDCS, CDCS，CSDCS において合成アルゴリズム以外の差分がないため，削減効果の内的妥当性を脅かす要因は存在しない．M，T の計測に関しては，OS が制御するバックグラウンドプロセスが影響を与えている可能性があるが，その影響は合成で DCS で要求される M，T と比較して非常に小さな値であるため無視できる．外的妥当性を脅かす要因としては，実験を行っていない開発対象システムの存在が挙げられる．この外的妥当性への脅威に可能な限り対処するために，性能比較対象（SDCS，CDCS）の評価実験で扱われたシステムにおいて CSDCS の評価実験を行い，恣意的な実験対象の選別を回避を目指した．

6. 関連研究

CSDCS のように，検証時の計算空間爆発を抑制するために，システム全体の検証問題をシステムの構成要素ごとのサブ問題に分解・検証するアプローチは Compositional Verification と呼ばれ，モデル検査分野で広く研究されてきた [18][19][20][21][22][23][24][25]．しかし，安全性を充足可能なシステム全体の状態空間を制御器として出力する必要がある DCS では，サブ問題の解がシステム全体の問題の状態空間のどの状態に該当するか特定する必要がある．よって，問題をサブ問題として解くだけでは DCS の計算空間削減は実現できず，DCS 分野には Compositional Verification は不適だとされてきた．

CSDCS は，Compositional Verification によって生じるサブ問題（部分合成）の直接的な解である充足可能領域ではなく，安全性ゲームで S_{err} となった非充足可能領域に着目した．サブ問題で判明した非充足可能領域を，以降の全ての DCS プロセスにおいて構築回避するよう，DCS プロセスから見直すことで，DCS 分野においても Compositional Verification が計算空間削減に有効であることを示した．よって，CSDCS は DCS 分野における Compositional Verification 適用のさきがけ的研究であると思われる．

Abstraction Synthesis 以外にも，DCS 分野では計算空間爆発に対処するアプローチとして，On-the-fly Synthesis や Abstraction Synthesis が研究されてきた．

On-the-fly Synthesis を用いた DCS[26][27][28][29] では，環境モデルと監視モデルに加えて目標状態が与えられ，安全性を違反することなく初期状態から目標状態に到達可能な状態遷移列を制御器として合成する．安全性を違反することのない状態空間を初期状態から逐次探索かつ構築し，目標状態に到達次第探索を終了する．これによりシステムの全状態空間の構築を回避し，計算空間削減を実現した．しかし，初期状態から目標状態までの制御器しか導出できないため，On-the-fly Synthesis では目標状態のない継続的な稼動を目的とするシステムの制御器の合成はできない限界が存在する．対して CSDCS は，安全性を充足するシステムの全状態空間を制御器として導出できることから，このような限界は存在しない．

Abstraction Synthesis を用いた DCS[30][31][8] では，安全性充足を分析する上で詳細である必要のない状態空間を双模倣性を維持しつつ抽象化することで，構築されるゲーム空間の状態数を削減した．しかし，CSDCS と異なり安全性と関わらない一部の状態遷移が抽象化された制御器が合成されるため，システム全体の詳細な制御を定める必要のある開発初期段階における動作仕様設計には用いることができない限界が存在する．

以上，CSDCS 以外の DCS 計算空間削減手法の多くは，制御器の用途を制限することで，その用途で不要となる計算空間の構築を回避してきた．対して CSDCS は，従来と同じ制御器を合成しつつ状態削減可能であるため，制御器の用途が制限されない．そのため，制御器の用途を制限することで，CSDCS は他の DCS 計算空間削減手法との両立も可能と考えられるため，用途に応じた更なる計算空間削減手法の拡張が期待される．

7. あとがき

本論文では，CDCS と SDCS における適用選択の課題を解消するために，両手法と比較して同等以上の適用可能範囲

と状態空間削減効果を備える CSDCS を提案した．SDCS の段階的なゲーム空間構築アプローチを基盤としつつ，ゲーム空間構築処理に Consolidated Game Composition を採用することで，高い計算空間削減効果と少ない計算時間の両立を実現した．この CSDCS によって，SDCS と CDCS の使い分けに関する判断工程を不要とし，両者の適用選択における課題を解消した．今後の研究課題としては，CSDCS アルゴリズムを制御器の用途に応じて最適化し，さらなる計算空間の削減を図ることが挙げられる．

参考文献

[1] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25, 1987.

[2] 山内 拓人, 李 家隆, 鄭 顕志, and 本位田 真一. ゲーム空間の一括構築による離散制御器合成の計算空間削減. **情報処理学会論文誌**, 65(8):1–12, 8 2024.

[3] 山内 拓人 and 鄭 顕志. 段階的な部分合成による離散制御器合成の分析空間削減. **電子情報通信学会論文誌** *D*, J106-D(4):218–230, 4 2023.

[4] Nicolás Roque D'Ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesis of live behaviour models. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 77–86. ACM, 2010.

[5] Yehia Abd Alrahman, Víctor A. Braberman, Nicolás D'Ippolito, Nir Piterman, and Sebastián Uchitel. Synthesis of run-to-completion controllers for discrete event systems. In *2021 American Control Conference*, pages 4892–4899, 2021.

[6] Jeff Magee and Jeff Kramer. *Concurrency: State Models and Java Programs*. Wiley Publishing, 2nd edition, 2006.

[7] D. Ciolek, V. Braberman, N. D'Ippolito, and S. Uchitel. Directed controller synthesis of discrete event systems: Taming composition with heuristics. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4764–4769, Dec 2016.

[8] 相澤和也, 鄭 顕志, and 本位田 真一. 違反状態抽象化による保証可能な安全性特定のための分析空間削減. **電子情報通信学会論文誌** *D*, J103-D(4):238–246, 4 2020.

[9] Kazuya Aizawa, Kenji Tei, and Shinichi Honiden. Analysis space reduction with state merging for ensuring safety properties of self-adaptive systems. In *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation*, pages 1363–1370, 2019.

[10] 山内 拓人, 鄭 顕志, and 本位田 真一. 段階的離散制御器合成における stepwise policy 設計の自動化. In *Technical Committee on Knowledge-Based Software Engineering (SIG-KBSE)*, volume 123, pages 73–75, 3 2024.

[11] A. Borda, L. Pasquale, V. Koutavas, and B. Nuseibeh. Compositional verification of self-adaptive cyber-physical systems. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 1–11, 2018.

[12] John Enright and Peter R. Wurman. Optimization and coordinated autonomy in mobile fulfillment systems. In *Automated Action Planning for Autonomous Mobile Robots, 2011 AAAI Workshop, San Francisco, California, USA, August 7, 2011*, 2011.

[13] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, Jan 1989.

[14] Yuki Arioka, Takuto Yamauchi, and Kenji Tei. Pre-controller synthesis for runtime controller synthesis. In *13th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2023, Penang, Malaysia, August 25-26, 2023*, pages 161–166. IEEE, 2023.

[15] Jialong Li, Wallace Manzano, Takuto Yamauchi, Nobuhiro Matsuyama, Elisa Yumi Nakagawa, and Kenji Tei. Employing discrete controller synthesis for developing systems-of-systems controllers. In Pablo Oliveira Antonino, Doo-Hwan Bae, Antonia Bertolino, Francesca Lonetti, and Ítalo Santos, editors, *Proceedings of the 12th ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems, SESoS 2024, Lisbon, Portugal, 14 April 2024*, pages 1–8. ACM, 2024.

[16] Takuto Yamauchi, Kenji Tei, and Shinichi Honiden. Method for low-cost environment partitioning modeling in dynamic update. In *3rd IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2020, Laguna Hills, CA, USA, December 9-13, 2020*, pages 183–187. IEEE, 2020.

[17] Nicolás D'Ippolito, Dario Fischbein, Marsha Chechik, and Sebastián Uchitel. MTSA: the modal transition system analyser. In *23rd IEEE/ACM International Conference on Automated Software Engineering, 15-19 September 2008, L'Aquila, Italy*, pages 475–476. IEEE Computer Society, 2008.

[18] Rajeev Alur, P. Madhusudan, and Wonhong Nam. Symbolic compositional verification by learning assumptions. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, pages 548–562, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[19] Yu-Fang Chen, Azadeh Farzan, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Learning minimal separating dfa's for compositional verification. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 31–45, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[20] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. PĂsĂreanu. Learning assumptions for compositional verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[21] Anubhav Gupta, Kenneth L. McMillan, and Zhaohui Fu. Automated assumption generation for compositional verification. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, pages 420–432, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[22] Wonhong Nam, P. Madhusudan, and Rajeev Alur. Automatic symbolic compositional verification by learning assumptions. *Formal Methods Syst. Des.*, 32(3):207–234, 2008.

[23] Corina S. Pasareanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M. Cobleigh, and Howard Barringer. Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. *Formal Methods Syst. Des.*, 32(3):175–205, 2008.

[24] Corina S. Pasareanu and Dimitra Giannakopoulou. Towards a compositional SPIN. In Antti Valmari, editor, *Model Checking Software, 13th International SPIN Workshop, Vienna, Austria, March 30 - April 1, 2006, Proceedings*, volume 3925 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2006.
[25] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2619 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003.
[26] Vadim Alimguzhin, Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci. On-the-fly control software synthesis. In Ezio Bartocci and C. R. Ramakrishnan, editors, *Model Checking Software - 20th International Symposium, SPIN 2013, Stony Brook, NY, USA, July 8-9, 2013. Proceedings*, volume 7976 of *Lecture Notes in Computer Science*, pages 61–80. Springer, 2013.
[27] Daniel Ciolek, Matias Duran, Florencia Zanollo, Nicolas Pazos, Julián Braier, Víctor A. Braberman, Nicolás D'Ippolito, and Sebastián Uchitel. On-the-fly informed search of non-blocking directed controllers. *Autom.*, 147:110731, 2023.
[28] D. Ciolek, V. Braberman, N. D'Ippolito, and S. Uchitel. Directed controller synthesis of discrete event systems: Taming composition with heuristics. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4764–4769, Dec 2016.
[29] Daniel Ciolek, Matias Duran, Florencia Zanollo, Nicolas Pazos, Julián Braier, Víctor A. Braberman, Nicolás D'Ippolito, and Sebastián Uchitel. On-the-fly informed search of non-blocking directed controllers. *Autom.*, 147:110731, 2023.
[30] Sahar Mohajerani, Robi Malik, Simon Ware, and Martin Fabian. Compositional synthesis of discrete event systems using synthesis abstraction. In *2011 Chinese Control and Decision Conference (CCDC)*, pages 1549–1554, 2011.
[31] Robi Malik, Sahar Mohajerani, and Martin Fabian. A survey on compositional algorithms for verification and synthesis in supervisory control. *Discrete Event Dynamic Systems*, 33(3):279–340, August 2023.

山内 拓人 （正会員）

2024 年に早稲田大学理工学術院基幹理工学研究科博士課程修了．同大学助手を経て，2024 年より早稲田大学講師．東京工業大学特別研究員を兼任．博士 (工学) (早稲田大学)．自己適応システム，要求工学の研究に従事．

李 家隆 （正会員）

2025 年早稲田大学基幹理工学研究科情報理工・情報通信専攻博士課程修了．同大学助手経て 2025 年より早稲田大学講師，東京科学大学特別研究員 (兼任)，現在に至る．自己適応システム，要求工学の研究に従事

鄭 顕志 （正会員）

2008 年早稲田大学大学院基幹理工学研究科博士課程修了．同大学助手，助教，国立情報学研究所助教，准教授，早稲田大学 研究院准教授/主任研究員，准教授を経て 2023 年より東京工業大学准教授．現在に至る．現在，早稲田大学基幹理工学研究科 客員准教授，国立情報学研究所 GRACE センター特任研究員を兼任．博士 (工学) (早稲田大学)．自己適応システム，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．本会シニア会員．

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，国立情報学研究所教授，同研究所副所長，東京大学大学院情報理工学系研究科教授，英国 UCL 客員教授，パリ第 6 大学招聘教授，リヨン第 1 大学招聘教授，早稲田大学理工学術院教授などを歴任し，2024 年より国立情報学研究所特任教授．現在に至る．日本ソフトウェア科学会名誉会員，本会フェロー，本会終身会員．