修士論文

Pre-Marking アクションを用いた Directed Controller Synthesis の 探索ヒューリスティック改善

大畑 允人

24M30552

東京科学大学
情報理工学院
情報工学コース

指導教員　鄭 顕志

2026 年 1 月

概要

概要

目次

概要 **ii**

第 1 章 序論 **1**
1.1 離散事象システムと離散制御器合成 1
1.2 本論文の目的 2
1.3 本論文の構成 2

第 2 章 背景技術 **4**
2.1 離散制御器合成 4
2.1.1 ラベル付き遷移系 4
2.1.2 同期アクションと並列合成 5
2.2 Directed Controller Synthesis 5
2.2.1 制御可能性と Directed Controller 5
2.2.2 制御問題 6
2.2.3 Marking アクションから Marked 状態への変換 6
2.2.4 On-The-Fly 探索アルゴリズム 6
2.3 Ready Abstraction 8
2.3.1 候補アクションの評価 9
2.3.2 距離推定の中核 11

第 3 章 Ready Abstraction の課題 **13**

第 4 章 Pre-Marking Direction **14**

第 5 章 関連研究 **15**

第 6 章 結論 **16**
6.1 本論文のまとめ 16
6.2 将来研究 16

謝辞 **17**

参考文献 **18**

図目次

表目次

第 1 章

序論

現代の社会インフラや産業システムにおいて，ソフトウェアが担う役割は拡大の一途をたどっている．これらのシステムの多くは，離散的な状態と，その状態を遷移させる事象（アクション）の列によって振る舞いが記述される離散事象システム（Discrete Event System, DES）としてモデル化できる．システムの高機能化・複雑化に伴い，システムが安全性や活性といった要求仕様を確実に満たすことを保証するのはますます困難になっており，設計段階における支援技術の重要性が高まっている．

本章では，離散事象システムとその制御に関する背景を述べ，本研究が取り組む課題と目的，および本論文の構成について概説する．

1.1 離散事象システムと離散制御器合成

離散事象システム（DES）は，離散的な状態空間を持ち，非同期に発生する事象によって状態遷移が引き起こされる動的システムである．製造ラインの工程管理，ロボットの動作計画，通信プロトコル，組み込みシステムなど，その適用範囲は多岐にわたる．これらのシステムに対し，危険な状態に到達しないという Safety や，目的とするタスクを無限回完了できるという Non-Blocking を保証することは，システムの信頼性を確保する上で不可欠である．

従来，これらの要求を満たす制御ロジック（動作仕様）の設計は，熟練した設計者の経験則や，試行錯誤的な検証に依存していた．しかし，並行して動作する複数のコンポーネントが複雑に相互作用するシステムにおいて，人間の直感だけで全てのコーナーケースを網羅し，デッドロックや禁止状態への到達を防ぐことは極めて困難である．設計ミスは手戻りのコストを増大させるだけでなく，運用時の重大な事故につながるリスクも孕んでいる．

この課題に対する解決策として，形式手法に基づき，与えられた環境モデルと要求仕様から正しい制御器を自動生成する離散制御器合成（Discrete Controller Synthesis）の研究が進められている []．離散制御器合成は，環境の可能な振る舞いをすべて考慮した上で，制御可能なアクションを適切に許可・禁止することで，システムが常に仕様を満たすことを数学的に保証する．特に，システム自身が能動的にアクションを選択してタスクを遂行するようなシステムにおいては，各状態で高々 1 つの制御アクションのみを選択する Directed Controller の合成が有効である []．この Directed Controller を合成する技術を Directed Controller Synthesis（DCS）と呼ぶ．

しかし，DCS の実適用における最大の障壁は状態空間爆発問題である．システムを構成するコンポーネント数が増加すると，合成後の状態空間は指数関数的に増大する．これに対処するため，状態空間全体を事前に構築せず，初期状態から必要な部分のみを探索する On-The-Fly 探索手法や，探索を効率化するためのヒューリスティック技術が提案されてきた []．特に，Ciolek らが提案した Ready Abstraction（RA）は，並列合成の構造を利用して目標までの距離を効率的に推定する強力なヒューリスティックであり，DCS の適用範囲を拡大させてきた．

1.2　本論文の目的

本論文の目的は，DCS における On-The-Fly 探索の効率をさらに向上させるため，従来の Ready Abstraction（RA）が抱える構造的な課題を解決する新たな探索指針である Pre-Marking Direction を提案することである．

従来の RA は，各コンポーネントにおいて局所的に発火可能なアクション（Ready アクション）の情報に基づいて目標までの距離を推定する．しかし，この推定手法には以下の構造的な課題が存在する．

同期構造を無視した到達可能性評価の限界

RA は，Ready アクションのみをノードとするグラフ上で距離を推定する．しかし，目標達成に特定のコンポーネント間の同期が不可欠な場合，現在は Ready アクションではないが将来的に同期を解放するために必要な予備アクションが評価から漏れる．その結果，同期ボトルネックを考慮しない近視眼的な経路選択が行われ，探索空間が不必要に拡大する．

Ready アクション探索に伴う計算負荷

RA は距離推定において，Ready アクション間の遷移コストやその先の距離を考慮し，再帰的に様々なパスを探索して最小値を計算する．この探索は，特に Ready アクションの候補が多い場合に計算コストが増大し，1 ステップごとの処理時間を長引かせる要因となる．

そこで本研究では，複雑な Ready アクショングラフの探索に依存せず，目標アクションの発火に不可欠な同期イベント（Pre-Marking アクション）を直接的な道標とする新たな距離推定手法を提案する．具体的には，以下の項目に取り組む．

Ready Abstraction の探索特性と課題の分析

Ready アクションを用いた距離推定が，同期を要するシステムにおいて近視眼的な挙動を示す原因と，それによって探索空間が不必要に拡大してしまうメカニズムについて，具体的なモデルを用いて分析する．

Pre-Marking Direction の提案

目標アクションの発火に不可欠な同期アクションを Pre-Marking アクション（PMA）として定義し，これを探索の道標とする新たなヒューリスティック手法 Pre-Marking Direction（PMD）を提案する．PMD は，RA のような複雑な探索を行わず，各コンポーネント単体での距離のみを用いた軽量な計算を採用する．PMA が同期構造上の要所を捉えているため，この単純化された指標でも十分な探索性能を発揮でき，計算負荷を抑えつつ同期ボトルネックを考慮した高速な探索を実現する．

評価実験による有効性の検証

提案手法を実装し，ベンチマーク問題を用いて従来手法と比較することで，探索状態数や計算時間の削減効果を定量的に評価する．

本研究の成果は，より大規模で複雑なシステムの制御器合成を現実的な時間・メモリリソースで可能にし，高信頼なシステム開発の自動化に貢献するものである．

1.3　本論文の構成

本論文の構成は以下の通りである．

第 2 章「背景技術」では，本研究の基礎となる離散制御器合成および DCS の理論的枠組み，On-The-Fly 探索アルゴリズム，および既存のヒューリスティック手法である Ready Abstraction について詳説

する．

第 3 章「Ready Abstraction の課題」では，既存の RA が抱える構造的な限界について述べる．特に，Marking アクションの発火にコンポーネント間の同期が必要な状況において，現状態の Ready アクションのみに基づく RA の推定が機能せず，近視眼的な探索となり探索空間の爆発を招くメカニズムについて分析する．

第 4 章「Pre-Marking Direction」では，本研究の中核となる提案手法について述べる．目標アクションの発火に必要な同期イベント（Pre-Marking アクション）を定義し，複雑なグラフ探索を行わずに各コンポーネント単体での距離情報のみを用いることで，同期を考慮した高精度な探索と，計算負荷の低い高速な推論を両立するアルゴリズムを示す．また，提案手法の適用による性能向上を実験により評価する．

第 5 章「関連研究」では，DCS の効率化やヒューリスティック探索に関する先行研究を概観し，本研究の位置づけを明確にする．

第 6 章「結論」では，本論文の成果を総括し，今後の展望について述べる．

第 2 章

背景技術

本章では，本研究の基盤となる離散制御器合成の概要，Directed Controller Synthesis（DCS）の定式化と On-The-Fly 探索アルゴリズム，および Ready Abstraction（RA）に基づくヒューリスティック関数について述べる．

2.1 離散制御器合成

離散制御器合成（Discrete Controller Synthesis）は，離散事象システム（Discrete Event System, DES）に対して，与えられた安全性などの制約を満たす制御器を自動的に構築する技術である．DES は有限状態オートマトン（あるいは LTS）の並列合成によってモデル化されるが，並列合成により得られる状態空間はコンポーネント数に対して指数的に増大しうる．この指数的爆発は離散制御問題の解決を困難にし，現在も研究上の課題となっている．

制御器は，制御可能なアクションを動的に無効化しながら，制御不能なアクションを監視することで，システムの振る舞いを制限する．一般のスーパバイザ制御では可能な限り多くの制御可能アクションを有効にする（最大許容）ことが求められることが多い．しかし，DES の制御器が「制御されたアクションを実行するアクティブなコンポーネント」として振る舞う文脈では，任意の時点で高々 1 つの制御可能アクションのみを選択する制御が適切となる場合がある．このような制御器は Directed Controller と呼ばれる．

2.1.1 ラベル付き遷移系

本研究では，システムを構成する各コンポーネントをラベル付き遷移系（Labeled Transition System, LTS）としてモデル化する．なお，本研究では Marking アクション（アクションベースの目標）を主に用いるが，既存研究では Marked 状態（状態ベースの目標）を用いることもある．この両者を扱えるよう，LTS には Marked 状態集合と Marking アクション集合の両方を持たせる（必要に応じて片方を空集合または全体集合として扱う）．

定義 1（LTS）． LTS は $E = (S_E, s_{E,0}, S_E^I, S_E^M, A_E, A_E^M, \Delta_E)$ で表現される．S_E は有限の状態集合であり，$s_{E,0} \in S_E$ は初期状態を表す．$S_E^I \subseteq S_E$ は違反状態の集合であり，システムが到達してはならない状態を表す．$S_E^M \subseteq S_E$ は Marked 状態の集合である（RA や既存の DCS 定式化で用いる）．$A_E = A_E^C \cup A_E^U$ はアクション集合であり，A_E^C は制御可能アクション，A_E^U は制御不能アクションを表す．$A_E^M \subseteq A_E$ は Marking アクション集合である（本研究の定式化で用いる）．$\Delta_E \subseteq S_E \times A_E \times S_E$ は遷移関係を表す．

定義 2（決定性）． LTS E が決定的であるとは，任意の状態 $s \in S_E$ とアクション $a \in A_E$ について，$(s, a, s_1) \in \Delta_E$ かつ $(s, a, s_2) \in \Delta_E$ ならば $s_1 = s_2$ が成り立つことをいう．

本研究では，全ての LTS は決定的であると仮定する．

定義 3（トレース）． Δ_E に従う状態とアクションの有限または無限の交互列 $t = s_0, a_0, s_1, a_1, \ldots$ をトレースと呼ぶ．ただし，各 i について $(s_i, a_i, s_{i+1}) \in \Delta_E$ が成り立つ．E 上のトレースの集合を T_E で表す．

2.1.2 同期アクションと並列合成

定義 4（同期アクション）． 複数の LTS $E_1, \ldots, E_n$ において，アクション a が 2 つ以上のコンポーネントのアクション集合に含まれるとき，a を同期アクションと呼ぶ．同期アクション集合 A^S は

$$A^S = \left\{ a \in \bigcup_{i=1}^{n} A_{E_i} \;\middle|\; |\{i \mid a \in A_{E_i}\}| \geq 2 \right\}$$

と定義する．

定義 5（並列合成）． LTS 集合 $\mathcal{E} = \{E_1, \ldots, E_n\}$ の並列合成 $E_\| = E_1 \parallel E_2 \parallel \cdots \parallel E_n$ を $E_\| = (S_{E_\|}, s_{E_\|,0}, S^I_{E_\|}, S^M_{E_\|}, A_{E_\|}, A^M_{E_\|}, \Delta_{E_\|})$ として定義する．状態集合 $S_{E_\|} = \prod_{i=1}^{n} S_{E_i}$ は各コンポーネントの状態の直積であり，初期状態は $s_{E_\|,0} = (s_{E_1,0}, \ldots, s_{E_n,0})$ である．違反状態集合 $S^I_{E_\|}$ は，いずれかのコンポーネントが違反状態にある状態の集合 $\{(s_1, \ldots, s_n) \in S_{E_\|} \mid \exists i. s_i \in S^I_{E_i}\}$ として定義される．Marked 状態集合は $S^M_{E_\|} = \prod_{i=1}^{n} S^M_{E_i}$，アクション集合 $A_{E_\|} = \bigcup_{i=1}^{n} A_{E_i}$，Marking アクション集合 $A^M_{E_\|} = \bigcup_{i=1}^{n} A^M_{E_i}$ である．遷移関係 $\Delta_{E_\|}$ については，同期アクション $a \in A^S$ は a を持つ全ての LTS で同時に発火し，非同期アクション $a \notin A^S$ は a を持つ LTS のみで発火する．

並列合成には以下の特徴がある：

- 同期アクションによりコンポーネント間の協調動作を実現できる
- 状態数 $|S_{E_\|}|$ は LTS 数に対して指数的に増大しうる

2.2 Directed Controller Synthesis

本節では，Directed Controller Synthesis（DCS）の形式的定義と，On-The-Fly 探索によるアルゴリズムについて述べる．

2.2.1 制御可能性と Directed Controller

アクションには制御可能なものと制御不能なものがある．制御可能アクションは，制御器によって有効化・無効化が可能である．一方，制御不能なアクションは環境によって発火されるため，発火し得る場合は全ての発火を考慮しなければならない．各状態の制御可能なアクションのうち高々 1 つのみを有効にする制御器を合成することで，システム全体の振る舞いを制御する．このような制御器を Directed Controller と呼ぶ．

定義 6（Directed Controller）． 並列合成 LTS $E_\|$ に対する制御器が以下の条件を満たすとき，Directed Controller と呼ぶ：

Controllable すべての制御不能アクションを常に有効化する
Directed 各状態で高々 1 つの制御可能アクションのみを有効化する
Eager Marked 状態，または Marking アクション発火後の状態において，制御可能アクションのみ存在する場合，必ず 1 つを選択する

2.2.2 制御問題

既存の DCS では，タスク完了などを表す Marked 状態への到達可能性に基づいて Non-Blocking 性を定義することがある．本研究の Marking アクションベース定式化との関係は 2.2.3 節で述べる．また，本研究では，目標を状態ではなく Marking アクション A^M の発火として与える．

定義 7（Directed Controller の制御問題）**．** LTS の組 $\boldsymbol{E} = (E_1, \ldots, E_n)$ に対し，解は以下を満たす Directed Controller である：

Safety　制御下のシステムが違反状態に到達しない
Non-Blocking　Marked 状態への到達，または Marking アクションの発火を無限回行える

2.2.3 Marking アクションから Marked 状態への変換

本研究では，目標を特定の状態への到達ではなく，Marking アクション集合 A^M のいずれかの要素が発火することとして定義する．既存の Ready Abstraction (RA) の枠組みは状態ベースの目標を想定しているため，アクションベースの目標を状態ベースに変換するための LTS E_M を導入し，他のコンポーネントの設定を調整する．

合成対象のコンポーネント集合を $\mathcal{E} = \{E_1, \ldots, E_n\}$ とし，全コンポーネントのアクション集合の和を $A = \bigcup_{i=1}^{n} A_{E_i}$ とする．このとき，Marking アクションの発生を状態として保持する Marking 判定 LTS $E_M = (S_{E_M}, s_{E_M,0}, S^I_{E_M}, S^M_{E_M}, A_{E_M}, A^M_{E_M}, \Delta_{E_M})$ を以下のように定義する：

- $S_{E_M} = \{0, 1\}$, $s_{E_M,0} = 0$, $S^I_{E_M} = \emptyset$, $S^M_{E_M} = \{1\}$
- $A_{E_M} = A$ （同期のために全アクションをアルファベットに含める）
- 遷移関係 Δ_{E_M}：
  - 任意の $a \in A^M$ に対し，$(0, a, 1) \in \Delta_{E_M}$ および $(1, a, 1) \in \Delta_{E_M}$
  - 任意の $a \notin A^M$ に対し，$(0, a, 0) \in \Delta_{E_M}$ および $(1, a, 0) \in \Delta_{E_M}$

さらに，並列合成 $E_{\|} = E_1 \parallel \cdots \parallel E_n \parallel E_M$ において目標達成の判定を E_M に一任するため，他のすべてのコンポーネント $E_i \in \{E_1, \ldots, E_n\}$ について，その Marked 状態集合を $S^M_{E_i} = S_{E_i}$ （全状態を Marked とする）へ再定義する．

この変換により，合成システム $E_{\|}$ が Marked 状態にあることは，E_M が状態 1 にあることと等価になる．E_M は直前に実行されたアクションが A^M に属する場合のみ状態 1 に遷移するため，この Marked 状態を無限回訪問することは，元のシステムにおいて Marking アクションを無限回発火し続ける Non-blocking な振る舞いと完全に一致する．

2.2.4 On-The-Fly 探索アルゴリズム

従来のアプローチでは，まずオートマトンを完全に合成し，その後で制御問題を解く．しかし，この方法は指数的な状態空間爆発を引き起こす可能性がある．Ciolek ら [] は，ヒューリスティックに導かれた On-The-Fly 探索により，状態空間の一部のみを探索して Directed Controller を発見するアルゴリズムを提案した．

On-The-Fly 探索は，状態空間全体を事前に構築することなく，初期状態から到達可能な必要な部分のみを段階的に展開しながら制御器を合成する手法である．この手法は，主に以下の 3 つのプロセスから構成される．

1. **展開（Expansion）**：現在の探索フロンティアから，ヒューリスティック関数（例えば RA）を用いて最も有望な未探索の遷移を選択し，探索グラフに追加する（ExpandNext）．これにより，Marked 状態への到達可能性が高い経路を優先的に探索する．
2. **伝播（Propagation）**：ある状態が *Goals*（勝利状態）または *Errors*（敗北状態）であることが確定した場合，その情報を探索グラフの逆方向（親状態）へ伝播させる（PropagateGoal, PropagateError）．状態の分類は以下の論理に基づき決定される：
   - **Goals への伝播**：ある状態から制御可能な遷移によって少なくとも 1 つの *Goals* 状態へ到達可能であるか，あるいはその状態から発生しうる全ての制御不能な遷移の先が *Goals* である場合，その状態は *Goals* となる．
   - **Errors への伝播**：ある状態から発生しうる全ての制御可能な遷移の先が *Errors* かつ制御不能な遷移によって少なくとも 1 つの *Errors* 状態へ到達可能である場合，その状態は *Errors* となる．
3. **閉路検出と判定（Loop Detection）**：探索中に新たな閉路が形成された場合，その閉路が Non-Blocking を満たすか否かを判定する．Marking アクションを含む閉路などの「勝利閉路」が見つかれば，その閉路上の状態は *Goals* となり，逆に Marking アクションを含まず脱出不可能な閉路は *Errors* となる．

探索は初期状態が *Goals* または *Errors* に分類されるまで継続される．初期状態が *Goals* に分類された場合，探索グラフから有効な部分グラフを抽出することで制御器が得られる．逆に *Errors* に分類された場合は，制御器の合成は不可能であると結論付けられる．

アルゴリズム 1 On-The-Fly 探索による Directed Controller Synthesis

入力： 全ての LTS の組 $\boldsymbol{E} = (E_1, \ldots, E_n)$，ヒューリスティック関数 H

出力： Directed Controller C または 合成不可能

function DirectedControllerSynthesis($\boldsymbol{E}, H$)
  $ES \leftarrow$ 初期状態 $\boldsymbol{s}_0$ のみを持つ部分探索グラフ
  $Goals \leftarrow \emptyset, Errors \leftarrow \emptyset, None \leftarrow \{\boldsymbol{s}_0\}$

  ▷ 探索ループ：初期状態が未分類状態でなくなるまで ◁
  while $\boldsymbol{s}_0 \notin (Goals \cup Errors)$ **do**
    $(\boldsymbol{s}, a, \boldsymbol{s}') \leftarrow$ ExpandNext(ES, H)　▷ ヒューリスティックによる次遷移の選択
    $ES \leftarrow ES \cup \{(\boldsymbol{s}, a, \boldsymbol{s}')\}$　▷ 探索グラフの更新

    if $\boldsymbol{s}' \in Errors$ **then**
      $Errors \leftarrow Errors \cup \{\boldsymbol{s}'\}$
      PropagateError($\boldsymbol{s}'$)　▷ 敗北状態の伝播
    else if $\boldsymbol{s}' \in Goals$ **then**
      $Goals \leftarrow Goals \cup \{\boldsymbol{s}'\}$
      PropagateGoal($\boldsymbol{s}'$)　▷ 勝利状態の伝播
    else if $\boldsymbol{s}'$ が新しいループを形成する **then**
      $Loop \leftarrow$ GetLoop($\boldsymbol{s}, \boldsymbol{s}'$)
      if $Loop$ が勝利条件を満たす **then**
        $newGoals \leftarrow$ FindNewGoals($Loop$)
        $Goals \leftarrow Goals \cup newGoals$
        PropagateGoal($newGoals$)
      else
        $newErrors \leftarrow$ FindNewErrors($Loop$)
        $Errors \leftarrow Errors \cup newErrors$
        PropagateError($newErrors$)

  if $s_0 \in Goals$ **then**
    $C \leftarrow$ ExtractController($ES, Goals$)
    return C
  else
    return 合成不可能

2.3 Ready Abstraction

Ready Abstraction（RA）は，並列合成構造を活用して，Marked 状態への到達に必要なステップ数を多項式時間で推定するヒューリスティック手法である．各コンポーネントの局所情報のみを用いて距離を推定することで状態空間爆発を回避しつつ，On-The-Fly 探索における次遷移選択（ExpandNext）を導く．

RA の基本的な着想は，合成状態 $\boldsymbol{s} = (s_{E_1}, s_{E_2}, \ldots, s_{E_n})$ の周辺で局所的に発火可能なアクション（Ready アクション）を集め，それらの間に，あるアクションの実行が別のアクションの実行可能性を高

めるという関係を仮想的に張ったグラフ上で最短路を解くことで，目標（Marked 状態）までの距離を見積もる点にある．同期制約のため，局所的に発火可能（Ready）であっても合成状態全体では直ちに発火できない場合があるが，RA はこの差を局所到達可能性とグラフ探索に基づいて保守的に見積もる．

2.3.1 候補アクションの評価

アルゴリズム 2 は，現在の合成状態 $\boldsymbol{s}$ において候補アクション $\hat{a}$ を選ぶことがどれだけ目標に近いかを評価するためのヒューリスティック関数である．評価は各コンポーネント E_i ごとに (rank, d) の形で返され，rank は距離推定に用いる目標集合の優先度を表す．

本章で示す RA の記述では, 探索中にすでに到達した Marked 状態集合 $\boldsymbol{S}^M$ を基準にした距離（Rank 0）を優先する．これは，探索が進むほど既知の到達済み領域へ近づく選択を上位に扱うことで，任意の Marked 状態を一様に目指す場合に比べて探索が過度に広がることを避け，探索空間を抑制できるという見込みに基づく．Rank 0 での推定が不可能な場合に限り，任意の Marked 状態への距離（Rank 1）を用いる．どちらも不可能と推定された場合は $(2, \infty)$ を返す．

アルゴリズム 2 Ready Abstraction によるヒューリスティック関数

入力： 全ての LTS の組 $\boldsymbol{E} = (E_1, E_2, \ldots, E_n)$,
探索到達済みの Marked 状態の集合 $\boldsymbol{S}^M \subseteq S^M_{E_1} \times S^M_{E_2} \times \cdots \times S^M_{E_n}$,
各 LTS における状態の組 $\boldsymbol{s} = (s_{E_1}, s_{E_2}, \ldots, s_{E_n})$,
候補のアクション $\hat{a}$

出力： 各 LTS における距離の降順の組 $\boldsymbol{d} = (\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_n)$
ただし，$\boldsymbol{d}_i \in \{(0, d), (1, d), (2, \infty) \mid d \in \mathbb{N}\}$
各 LTS における距離の意味は以下の通り：
$(0, d)$：次にアクション $\hat{a}$ を発火して d ステップで探索到達済みの Marked 状態に到達可能
$(1, d)$：次にアクション $\hat{a}$ を発火して d ステップで Marked 状態に到達可能
$(2, \infty)$：Marked 状態に到達不可能

```
1:  function ReadyAbstractionHeuristic(E, S^M, s, â)
2:      for i = 1, 2, ..., n do
3:          minSteps ← ∞

4:          ▷ Rank 0: 探索到達済みの Marked 状態への到達可能性確認 ◁
5:          for all (s*_{E_1}, s*_{E_2}, ..., s*_{E_n}) ∈ S^M do
6:              steps ← EstimateDist(s, â, s*_{E_i})
7:              if steps < minSteps then minSteps ← steps
8:          if minSteps ≠ ∞ then        ▷ いずれかの探索到達済みの Marked 状態に到達可能な場合
9:              d_{E_i} ← (0, minSteps)
10:             continue

11:         ▷ Rank 1: 任意の Marked 状態への到達可能性確認 ◁
12:         for all s*_{E_i} ∈ S^M_{E_i} do
13:             steps ← EstimateDist(s, â, s*_{E_i})
14:             if steps < minSteps then minSteps ← steps
15:         if minSteps ≠ ∞ then                ▷ いずれかの Marked 状態に到達可能な場合
16:             d_{E_i} ← (1, minSteps)
17:         else
18:             d_{E_i} ← (2, ∞)

19:     return SortDescending((d_{E_1}, d_{E_2}, ..., d_{E_n}))        ▷ 距離を辞書式降順でソートして返す
```

アルゴリズム 2 の戻り値は各コンポーネントの距離推定の組であるが，実際の探索（ExpandNext）において候補アクション $\hat{a}$ を選択する際は，以下の優先順位に従って順位付けを行う：

1. **制御不能アクションの優先**：$\hat{a} \in A^U$ であるアクションを，全ての制御可能アクション $\hat{a} \in A^C$ よりも優先する．
2. **辞書式順序による比較**：アクションの属性が同じ（共に制御可能，あるいは共に制御不能）である場合，アルゴリズム 2 が返す距離の組 $\boldsymbol{d}$ を辞書式に比較し，値が小さい（より目標に近いと推定される）ものを優先する．

制御不能アクションを最優先するのは，環境側で発生しうる全ての振る舞いを早期に探索し，反例（違反状態やデッドロック）を迅速に検出するためである．

2.3.2 距離推定の中核

アルゴリズム 3 は，アルゴリズム 2 が内部で呼び出す距離推定関数である．入力は現在の合成状態 $\boldsymbol{s}$，次に実行する候補アクション $\hat{a}$，距離の対象となるコンポーネント状態 $s^*_{E_i}$ であり，出力は推定距離 d である．

推定は大きく二段階で行われる．まず，$\hat{a}$ がコンポーネント E_i に存在するなら，$\hat{a}$ を局所的に一回発火した後の状態 s'_{E_i} を確認し，そこからの局所最短距離 CalculateDist$(s'_{E_i}, s^*_{E_i})$ を用いて直接効果を見積もる．これで判断できない場合には，現在の合成状態から（同期を無視すれば）状態を変化させ得るアクション集合 A^R を列挙し，切り替えコスト（Gap）とその先の推定距離の和の最小値として距離を与える．この再帰的最小化は，実装上は RA グラフ上の最短路として扱うことで効率化できる．

アルゴリズム 3 Ready Abstraction における距離推定関数

入力： 現在の状態 $\boldsymbol{s} = (s_{E_1}, \ldots, s_{E_n})$,
候補のアクション $\hat{a}$,
距離を推定する対象の状態 $s^*_{E_i}$
出力： 推定距離 $d \in \mathbb{N} \cup \{\infty\}$

function EstimateDist$(\boldsymbol{s}, \hat{a}, s^*_{E_i})$
  if $\hat{a} \in A_{E_i}$ **then** ▷ $\hat{a}$ が LTS E_i に存在している場合
    $s'_{E_i} \leftarrow s'$ where $(s_{E_i}, \hat{a}, s') \in \Delta_{E_i}$ ▷ $\hat{a}$ を発火した後の状態
    if $s'_{E_i} = s^*_{E_i}$ **then** ▷ 対象の状態に到達する場合
      return 1
    else if $s'_{E_i} \in S^I_{E_i}$ **then** ▷ 違反状態に到達する場合
      return ∞ ▷ 距離を無限と推定し，Marked 状態に到達不可能であることを表す
    else if $s'_{E_i} \neq s_{E_i}$ **then** ▷ 他の状態に遷移する場合
      return CalculateDist$(s'_{E_i}, s^*_{E_i}) + 1$
  else if $s_{E_i} = s^*_{E_i}$ **then** ▷ $\hat{a}$ で遷移が発生せず，すでに対象の状態の場合
    return 1

  ▷ 各 LTS 上で，現在の状態から発火可能なアクションのうち，自己ループでないものの集合 ◁
  for $j = 1, 2, \ldots, n$ **do**
    $A^R_{E_j} \leftarrow \{a \mid \exists s', (s_{E_j}, a, s') \in \Delta_{E_j}, s' \neq s_{E_j}\}$
  $A^R \leftarrow \bigcup_{j=1}^{n} A^R_{E_j}$ ▷ Ready アクションの集合

  ▷ Gap を加味した最短経路探索 ◁
  $d \leftarrow \min_{a \in A^R}\{\text{CalculateGap}(\boldsymbol{s}, \hat{a}, a) + \text{EstimateDist}(\boldsymbol{s}, a, s^*_{E_i})\}$
  ▷ この再帰的な最小値問題は，実装上はダイクストラ法により効率的に解かれる ◁
  if $d = \infty$ **then** ▷ Ready アクションを用いて距離を推定できない場合
    $d \leftarrow \text{CalculateDist}(s_{E_i}, s^*_{E_i}) + 1$
  return d

ここで用いられる CalculateGap$(\hat{a}, a)$ は，アクション間の切り替えのしやすさを表すコストであり，既存研究における定義 $G^e_E(\hat{a}, a)$ に従い，あるコンポーネント内でアクション $\hat{a}$ から a へ至る最短パスの長さ（から 1 を引いた値）として計算される．

以上により，RA は局所情報から，次にどのアクションを展開するのが有望かを推定する枠組みを提

供する．On-The-Fly 探索側では，これらの推定値を用いて展開順序を制御することで，合成状態空間の全探索を避けつつ，目的を満たす制御器の発見を狙う．

第 3 章

Ready Abstraction の課題

第 4 章

Pre-Marking Direction

第 5 章

関連研究

第 6 章

結論

6.1 本論文のまとめ

6.2 将来研究

謝辞

参考文献