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

目次

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（Discrete Event System, 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 [6]．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．こうしたシステムにおいて，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境界条件や例外ケースを網羅し，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．設計ミスは再設計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 [5]．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場合には，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 [4]．この Directed Controller を合成する技術を Directed Controller Synthesis（DCS）と呼ぶ [2].

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 [1]．特に，Ciolek らが提案した Ready Abstraction（RA）[3] 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 アクションの数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）を直接的な指針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する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，これを探索の指針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 のような動的な Ready アクショングラフの探索を排除し，静的解析によって得られる各コンポーネント単体での距離情報を統合する軽量な計算手法を採用する．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 [5]．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，制御器自身がアクションを能動的に実行するアクティブなコンポーネントとして振る舞う場合には，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．このような制御器は Directed Controller と呼ばれる [4].

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 $E_\parallel = E_1 \parallel E_2 \parallel \cdots \parallel E_n$ を $E_\parallel = (S_{E_\parallel}, s_{E_\parallel,0}, S^I_{E_\parallel}, S^M_{E_\parallel}, A_{E_\parallel}, A^M_{E_\parallel}, \Delta_{E_\parallel})$ として定義する．状態集合 $S_{E_\parallel} = \prod_{i=1}^{n} S_{E_i}$ は各コンポーネントの状態の直積であり，初期状態は $s_{E_\parallel,0} = (s_{E_1,0}, \ldots, s_{E_n,0})$ である．違反状態集合 $S^I_{E_\parallel}$ 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 $\{(s_1, \ldots, s_n) \in S_{E_\parallel} \mid \exists i. s_i \in S^I_{E_i}\}$ として定義される．Marked 状態集合は $S^M_{E_\parallel} = \prod_{i=1}^{n} S^M_{E_i}$，アクション集合 $A_{E_\parallel} = \bigcup_{i=1}^{n} A_{E_i}$，Marking アクション集合 $A^M_{E_\parallel} = \bigcup_{i=1}^{n} A^M_{E_i}$ である．遷移関係 $\Delta_{E_\parallel}$ 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 $|S_{E_\parallel}|$ 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 $E_\parallel$ 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第 2.2.3 項で述べる．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，この課題に対し，ヒューリスティック関数に導かれた 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 [2]．ヒューリスティック関数は，各状態から目標までの推定距離を計算することで，有望な探索経路を優先的に選択する手法であり [1]，DCS においても探索効率の向上に重要な役割を果たす．

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．

2.3 Ready Abstraction

Ready Abstraction（RA） [3] 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．この手法は，プランニング問題におけるヒューリスティック探索の一般的な枠組み [1] を，並列合成された DES の構造に適合させたものと位置づけられる．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 展開対象のアクションの評価

アルゴリズム 2 は，現在の合成状態 $\boldsymbol{s}$ において展開対象のアクション $\hat{a}$ 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

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

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 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

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 $\boldsymbol{s} = (s_{E_1}, s_{E_2}, \ldots, s_{E_n})$,
展開対象のアクション $\hat{a}$

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
```

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

Ready Abstraction における距離推定の手続きをアルゴリズム 3 に示す．この関数 EstimateDist は，現在の合成状態 $\boldsymbol{s}$，評価対象のアクション $\hat{a}$，および目標状態 $s^*_{E_i}$ を入力とし，推定距離 d を算出する．

本アルゴリズムでは，推定のために以下の 2 つの補助関数を利用する．

CalculateDist(s, a)

LTS 単体において，状態 s からアクション a へ到達するための最短ステップ数．この値は探索中頻繁に参照されるため，効率化の観点から，探索開始前にすべての状態とアクションの組について幅優先探索により算出されている．

CalculateGap$(\boldsymbol{s}, \hat{a}, a)$

現在の合成状態 $\boldsymbol{s}$ において，アクション $\hat{a}$ の実行後に別のアクション a を実行可能にするために必要な最小ステップ数（切り替えコスト）．これは動的な状態に依存するため，探索中に都度計算される．

RA は，現在の状態で発火可能な Ready アクション集合 A^R を対象に，そこへ至る遷移コスト（Gap）とその先の局所距離（Dist）の和が最小となる経路を探索する．これにより，単なる最短距離ではなく，同期のための待機やアクションの切り替えコストを含んだ，より実質的な到達距離を推定している．

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 $\boldsymbol{s} = (s_{E_1}, \ldots, s_{E_n})$,
評価対象のアクション $\hat{a}$,
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に従い，あるコンポーネント内でアクション $\hat{a}$ を経て 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

前章で述べたように，Ready Abstraction（RA）は局所的に発火可能なアクション（Ready アクション）間の関係性を利用して，目標までの距離を推定するヒューリスティック手法である．RA は，コンポーネント間の結合が疎であり，各コンポーネントが比較的自律的に目標へ遷移できるシステムにおいては強力なガイドとなる．

しかし，本研究が対象とするような，Marking アクションの発火に複数のコンポーネントによる厳密な同期が必要となるシステムにおいては，RA の推定精度と計算効率の両面で深刻な課題が生じる．本章では，RA が抱える構造的な限界について，同期構造の無視による探索効率の低下と，Ready アクション探索に伴う計算コストの増大という 2 つの観点から分析する．

3.1　同期構造に起因する近視眼的な探索

RA の最大の特徴は，現在の合成状態において直ちに発火可能なアクション（Ready アクション）のみをノードとするグラフ上で距離計算を行う点にある．この特性は，将来的に発生する同期イベントを適切に評価できないという欠点につながる．

3.1.1　同期待ち時間の無視

Marking アクション a_m が同期アクションである場合，その発火には関与する全てのコンポーネントが a_m を発火可能な状態に到達していなければならない．しかし，あるコンポーネント E_i が既に a_m の直前状態に到達していても，他のコンポーネント E_j がまだ準備できていなければ，システム全体として a_m は実行できない．

RA の距離推定アルゴリズム（アルゴリズム 3）は，現在の Ready アクション集合 A^R を起点として，目標までの最短パスを探索する．このとき，同期が必要なアクションへのパスは，パートナーの到着を待つための遷移（同期待ち）を含める必要があるため，RA のグラフ上では遠い，あるいは到達困難と判定されやすい．対照的に，同期を必要とせず単独で進行でき，かつ局所的に目標に近い場所へ遷移できるアクションが存在する場合，RA はその見かけ上の近さを優先して評価する．

RA はこの他者の同期待ちという大域的な制約を軽視し，各コンポーネントが局所的な最短経路を選択し続けるような近視眼的な評価を下すため，結果として合流不可能な経路や，効率の悪い脇道状態を優先的に探索する結果となる．

3.1.2　具体的な不適合例：金属加工システム

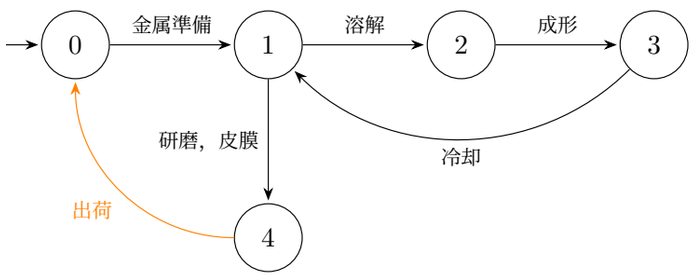
RA が不適切な誘導を行う具体的なシナリオとして，図 3.1 に示す金属加工システムのモデルを用いて説明する．

本システムは，加工プロセスを表す環境モデル（E_{env}）と，工程管理を行う要求モデル（E_{req}）の 2 つのコンポーネントから構成される．目標は「出荷」アクションの発火（Marking アクション）である．なお，説明を簡単にするため，本モデルに含まれる全てのアクションは制御可能（Controllable）であるとする．

環境モデルは，初期状態から「金属準備」を経て待機状態（状態 1）へ遷移する．この状態 1 を起点として，2 つの経路が存在する．1 つ目は「溶解」,「成形」,「冷却」を行い，再び待機状態に戻る加工サイクルである．このサイクルは 0 回以上実行可能である．2 つ目は「研磨」や「皮膜」を施し，最後に「出荷」して初期状態に戻る仕上げ・出荷工程である．

要求モデルは，製品の品質担保のために「成形」が 1 度以上発火されることを強制する．具体的には，初期状態（成形未実施，状態 0）において「成形」を経ずに「出荷」アクションが発火された場合，未加工品の出荷とみなされ，違反状態 -1（図下段の赤色ノード）へ遷移する．「成形」が発火されると状態 1（成形済み）へ遷移し，以降は「出荷」を行っても初期状態に戻るだけであり，違反にはならない．

環境モデル E_{env}



要求モデル E_{req}

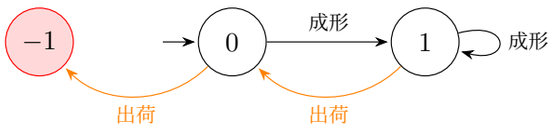


図 3.1　金属加工システムの環境モデルと要求モデル

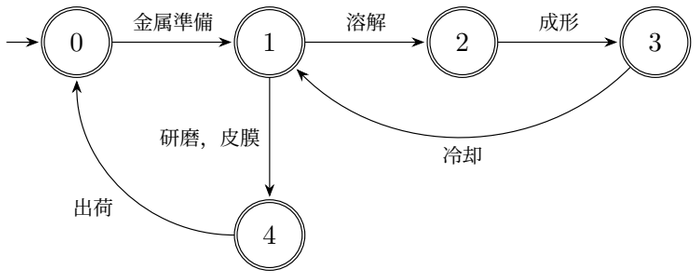
Ready Abstraction 適用のための Marked 状態への変換

RA は本来，Marked 状態への到達を目的とする探索手法である．一方，本問題の目標は「出荷」アクションの発火であるため，第 2.2.3 項で述べた変換手法を適用し，アクションベースの目標を状態ベースの目標に置き換える必要がある．

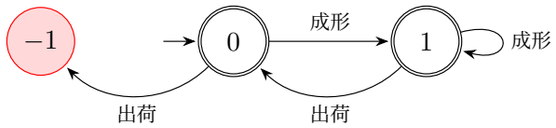
図 3.2 は，変換後のシステム構成を示している．ここでは，新たなコンポーネントとして Marking 判定モデル E_M を導入している．E_M は，「出荷」アクションが発火された直後のみ Marked 状態（状態 1）に遷移し，それ以外の場合は非 Marked 状態（状態 0）に留まる機能を持つ．いわば，アクションの発火イベントを状態遷移として表現するためのモデルである．

この変換に伴い，既存の環境モデル E_{env} および要求モデル E_{req} においては，違反状態を除くすべての状態を Marked 状態（図中の二重丸）として再定義する．これにより，システム全体の並列合成状態が Marked となる（全てのコンポーネントが同時に Marked 状態にある）ための条件は，実質的に E_M が状態 1 になること，すなわち直前に出荷アクションが発火したことと等価になる．

環境モデル E_{env}



要求モデル E_{req}



Marking 判定モデル E_M

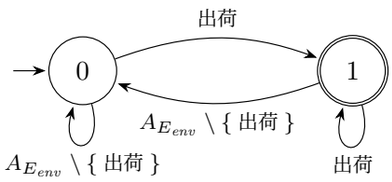


図 3.2　アクション目標を状態目標へ変換した金属加工システム

RA による探索と失敗のプロセス

前目で述べた変換後の金属加工システム（図 3.2）に対し，RA を用いた On-The-Fly 探索を適用した場合の具体的な挙動を示す．本モデルでは，E_{env} と E_{req} の全状態が Marked と設定されているため，探索の主目的は Mark 判定モデル E_M を状態 1（Marked）に遷移させること，すなわち「出荷」アクションを発火させることになる．

RA を用いた場合でも，最終的には適切な制御器が合成されるが，その過程で生じる非効率な探索挙動について以下に段階的に示す．

Step 1: 初期状態からの遷移

初期状態 $\boldsymbol{s}_0 = (0, 0, 0)$（順に E_{env}, E_{req}, E_M の状態）において，発火可能なアクションは金属準備のみである．システムはこれを選択し，状態 $\boldsymbol{s}_1 = (1, 0, 0)$ へ遷移する．このとき，E_{env} は状態 1（分岐点）に進むが，E_{req} と E_M は金属準備に関与しないため状態 0 に留まる．

Step 2: RA によるヒューリスティック評価と誤った選択

状態 $\boldsymbol{s}_1$ において，RA は次に選択すべきアクションを決定する．この時点で局所的に発火可能なアクション（Ready アクション）の集合は，$A^R = \{$ 研磨, 皮膜, 溶解, 成形, 出荷 $\}$ である．

まず，環境モデル E_{env} および要求モデル E_{req} の評価を行う．これらは設定により全状態が Marked であるため，現在の状態が既に目標であるとみなされる．したがって，これらのコンポーネントにおける距離推定値は 1 となり，アクション選択の差別化要因とならない．

したがって，評価の差が生じるのは Mark 判定モデル E_M における推定である．E_M が Marked 状態（状態 1）へ遷移するためには，同期アクションである「出荷」の発火が不可欠である．RA は，展開対象のアクション $\hat{a}$ から「出荷」への Gap（遷移コスト）と，その先の E_M における距離（Dist）の和として値を算出する．

選択肢 A：「研磨」または「皮膜」

環境モデル E_{env} において,「研磨」または「皮膜」から「出荷」への Gap は 1 である．一方，E_M において「出荷」発火後の状態は Marked 状態そのものであるため，その距離（Dist）は 1 である．よって，推定距離は $d = 1 + 1 = 2$ と算出される．

選択肢 B：「溶解」

環境モデル E_{env} において,「溶解」から「成形」「冷却」を経て「出荷」に至る最小の Gap は 3 である．E_M における距離（Dist）は同様に 1 である．よって，推定距離は $d = 3+1 = 4$ と算出される．

RA はこの推定値に基づき，距離の小さい（$2 < 4$）「研磨」および「皮膜」を優先すべき有望なアクションと判断する．その結果，アルゴリズムは正解ルートである「溶解」を後回しにし，まずは見かけ上のコストが低い「研磨」を選択して状態 $\boldsymbol{s}_2 = (4, 0, 0)$ への遷移を行う．

Step 3: 経路の行き詰まりと繰り返される不要な探索

遷移後の状態 $\boldsymbol{s}_2 = (4, 0, 0)$ において，環境モデル上では出荷が可能である．しかし，要求モデル E_{req} は依然として状態 0（成形未実施）にある．この状態で出荷を発火すると，E_{req} は違反状態（-1）へ遷移してしまうため，DCS の安全性制約によりこの遷移は禁止される．他に有効なアクションも存在しないため，この経路は探索の行き詰まりとなる．

この結果，探索アルゴリズムはこの経路を破棄し，分岐点である状態 $\boldsymbol{s}_1$ までバックトラックを行う．しかし，RA のヒューリスティック評価では，残る選択肢である「皮膜」も「溶解」よりコストが低いと判定されている．そのため，アルゴリズムは「溶解」を選ぶ前に，もう一方の選択肢である「皮膜」の探索へ移行する．「皮膜」ルートも同様に同期待機ができず行き詰まりとなるため，再びバックトラックが発生する．

このように，RA の誘導により 2 つの不要な経路を探索し尽くした後に，ようやく次善の選択肢であった「溶解」ルートの探索が開始される．

以上のプロセスにより，RA を用いた探索では，同期（E_{req} の状態進行）の必要性を見抜けずに，局所的に目標アクション「出荷」へ近づきやすい経路を優先して探索する．大規模なシステムにおいては，このような不要な探索とバックトラックが頻発し，計算リソースを著しく浪費する原因となる．

3.2 Ready アクション探索による計算コストの増大

RA のもう一つの課題は，ヒューリスティック値の計算自体のオーバーヘッドである．前章のアルゴリズム 3（EstimateDist）に示した通り，RA は距離を算出するために以下の処理を行う．

$$d \leftarrow \min_{a \in A^R} \{\textsc{CalculateGap}(\boldsymbol{s}, \hat{a}, a) + \textsc{EstimateDist}(\boldsymbol{s}, a, s^*_{E_i})\}$$

この式は，現在の合成状態における全ての Ready アクション A^R に対して，遷移コスト（Gap）と再帰的な距離推定を行い，その最小値を求めることを意味している．これは実質的に，探索の 1 ステップごとに，Ready アクションをノードとするグラフ上でダイクストラ法に近い最短経路探索を行っているに等しい．

システム規模が大きくなり，並列動作するコンポーネント数が増加すると，Ready アクション集合 A^R のサイズは増大する．DCS の On-The-Fly 探索では，数十万の状態を探索することが珍しくない．その全ての遷移において，この重い計算処理が走ることは，合成時間全体に対して無視できないペナルティとなる．

特に，前節で述べたように同期待ちが発生している状況では，有効なパスが見つからずに A^R 全体を探索し尽くす最悪ケースの計算が発生しやすくなる．すなわち，RA は探索中に迷いやすいだけでなく，その計算処理自体も重いという二重の課題を抱えていると言える．

3.3　本章のまとめ

本章では，Marking アクションを目標とする離散制御器合成において，従来の Ready Abstraction が抱える課題を分析した．

第一の課題は同期の無視である．RA は局所的な最短経路を優先するため，同期のための準備動作よりも，単独で進行可能な近道を過剰評価する．これにより，パートナーの到着を待てずに脇道へ逸れる探索（不要な探索）が発生する．

第二の課題は計算コストである．RA は距離推定のために Ready アクショングラフ上の探索を都度行うため，1 ステップあたりの計算負荷が高く，大規模なシステムにおいて合成時間のボトルネックとなる．

これらの課題は，RA が動的な Ready 情報とその探索に依存しすぎていることに起因する．次章では，これらの問題を解決するために，静的な構造情報である同期アクションまでの局所距離を活用し，計算コストを抑えつつ同期を強力に指向する新たな手法「Pre-Marking Direction」を提案する．

第 4 章

Pre-Marking Direction

本章では，前章で指摘した Ready Abstraction（RA）の課題を解決するため，新たなヒューリスティック手法である Pre-Marking Direction（PMD）を提案する．PMD は，目標アクションの発火に不可欠な同期イベントに着目し，複雑な Ready アクショングラフの探索を行わずに，静的な構造情報のみを用いて効率的な探索を実現する手法である．

4.1 Pre-Marking アクション

本節では，提案手法の核となる概念である Pre-Marking アクション（PMA）を定義する．前章で述べたように，RA の主な課題は同期構造の無視にある．複数のコンポーネントが同期して Marking アクションを発火する必要がある場合，すべてのコンポーネントが準備完了状態になるまで Marking アクションは発火できない．したがって，探索の指針とすべきシステム全体のボトルネックは，目標への到達に最も時間を要するコンポーネントによって決定される．

複数の Marking アクションが存在する場合，Non-Blocking 性はそれらのうちいずれか一つへの到達が可能であれば満たされる．したがって，特定の目標への到達にのみ必須となるアクションは，別経路による回避が可能であるため，システム全体のボトルネックとはみなせない．ゆえに，本研究ではすべての Marking アクションへの到達において共通して不可避なアクションのみを Pre-Marking アクションと定義する．

4.1.1 PMA の定義

Pre-Marking アクションは，LTS の構造に基づいて静的に定義される属性である．ある LTS $E_i = (S_E, s_{E,0}, A_E, A_E^M, \Delta_E)$ において，初期状態 $s_{E,0}$ から始まり，ある特定の Marking アクション $a_m \in A_E^M$ の発火をもって終了する有限トレースの集合を $T_{s_{E,0} \to a_m}$ とする．

$$T_{s_{E,0} \to a_m} = \{t \in T_E \mid t = s_0, a_0, \ldots, s_j, a_j,\ s_0 = s_{E,0} \wedge a_j = a_m\}$$

このとき，初期状態から a_m に到達するために不可避なアクション集合 $A_{s_{E,0} \to a_m}^{Required}$ は以下のように定義される．

$$A_{s_{E,0} \to a_m}^{Required} = \{a \in A_E \setminus \{a_m\} \mid \forall t \in T_{s_{E,0} \to a_m}, a \in A_t\}$$

ここで，A_t はトレース t に含まれるアクションの集合を表す．E における Pre-Marking アクションの集合 A_E^P は，すべての Marking アクションに対する $A^{Required}$ の積集合として定義される．

定義 8（Pre-Marking アクション）**．** アクション $a \in A_E$ が以下の条件を満たすとき，a は Pre-Marking

アクションである．

$$a \in \bigcap_{a_m \in A_E^M} A^{Required}_{s_{E,0} \to a_m}$$

この定義により，PMA は目標がいかなる形で達成されるとしても，回避することのできない構造上の通過点であることが保証される．

実際の探索において，各状態 $s \in S_E$ は，その時点において未発火である PMA の集合を保持する．状態 s における未発火 PMA 集合 A_s^P は，大域的に定義された A_E^P のうち，状態 s から Marking アクションへ到達するために今後必ず発火しなければならないアクションの集合である．なお，初期状態 $s_{E,0}$ においては，未発火の PMA 集合は大域的な定義と一致するため，$A^P_{s_{E,0}} = A_E^P$ が成立する．PMD はこの A_s^P を用いることで，その状態が目標達成に対して不可避な手順をあとどれだけ残しているかを定量化する．

4.1.2 PMA の抽出アルゴリズム

PMA の抽出および各状態への割り当ては，探索前の静的解析によって行われる．本手法では，後述する距離推定のために算出が必要となる，各状態から各アクションへの最短ステップ数の情報を活用し，効率的な候補の絞り込みと逆方向探索を組み合わせた 2 段階のアルゴリズムを採用している．

最短パス情報を用いた候補特定

第一段階では，Pre-Marking アクションとなり得る候補集合 A_E^{MP} を特定する．PMD ではヒューリスティック計算のために，各状態から各アクションへの最短ステップ数を事前に算出する．真に回避不能なアクション（PMA）は，初期状態から目標へ至るすべてのパスに含まれるため，最短パス上にも必ず存在する．この性質を利用し，初期状態から各 Marking アクションへの最短パス上に存在するアクションの積集合をとることで，PMA の候補を効率的に絞り込む．

具体的には，各 Marking アクション $a_m \in A_E^M$ に対し，初期状態から a_m に至る最短パスを一つ選択し，それに含まれるアクション集合を取得する．そして，すべての a_m についてこれらの積集合を計算し，これを候補集合 A_E^{MP} とする．この操作は，既存の距離計算結果を再利用するため，追加の探索コストを低く抑えることが可能である．

逆伝播による判定と状態への割り当て

第二段階では，候補アクション $a \in A_E^{MP}$ が真に PMA であるか，および具体的にどの状態において未発火であるかを判定する．これを状態ごとに順方向に探索して判定すると計算コストが高くなるため，本手法では Marking アクション側からの逆伝播による一括判定を行う．

ある候補アクション a について，以下の手順で判定を行う．まず，Marking アクションを直ちに発火可能な状態集合を起点とし，遷移を逆方向に辿りながら到達可能な状態集合 S^* を探索する．この際，アクション a による遷移は通行止めとみなして探索を行わない．この逆探索によって到達できた状態群 S^* は，アクション a を経由せずに Marking アクションの発火可能状態へ到達できる状態である．すなわち，これらの状態において a は回避可能，あるいは既に通過済みであるため，未発火の PMA には該当しない．

探索の結果，初期状態 $s_{E,0}$ が S^* に含まれない場合，初期状態から a を回避して目標へ到達する経路が存在しないことを意味する．この場合，アクション a はシステム全体の Pre-Marking アクションとして確定される．そして，逆探索で到達できなかった状態集合（$S_E \setminus S^*$）に対して，a を未発火 PMA として割り当てる．このアルゴリズムをアルゴリズム 4 に示す．

アルゴリズム 4 PMA 抽出および割り当てアルゴリズム

入力： LTS E

出力： 各状態 s における未発火 PMA 集合 $M_E^P : S_E \to 2^{A_E}$

▷ Step 1: 最短パス情報による候補特定 ◁
$A_E^{MP} \leftarrow A_E$
for all $a_m \in A_E^M$ **do**
    ▷ 最短トレースに含まれるアクション集合との積集合で更新 ◁
    $t \leftarrow \mathrm{argmin}_{t \in T_{s_{E,0} \to a_m}} |t|$
    $A_E^{MP} \leftarrow A_E^{MP} \cap A_t$

▷ Step 2: 逆伝播による状態への割り当て ◁
for all $s \in S_E$ **do** $M_E^P(s) \leftarrow \emptyset$ ▷ 全状態で空集合に初期化
for $a \in A_E^{MP}$ **do**
    ▷ Marking アクションを直ちに発火可能な状態集合 ◁
    $S^* \leftarrow \{s \in S_E \mid \exists a_m \in A_E^M, \exists s', (s, a_m, s') \in \Delta_E\}$
    $S^*_{prev} \leftarrow \emptyset$
    while $S^*_{prev} \neq S^*$ **do**
        $S^*_{prev} \leftarrow S^*$
        for all $s \in S^*_{prev}$ **do**
            ▷ s へ遷移可能な親状態 s' を探索（逆伝播） ◁
            for all $(s', a', s) \in \Delta_E$ **do**
                if $a' \neq a$ **then**
                    $S^* \leftarrow S^* \cup \{s'\}$
    ▷ 初期状態から回避不能である（真の PMA である）場合のみ割り当て ◁
    if $s_{E,0} \notin S^*$ **then**
        for $s \in S_E \setminus S^*$ **do**
            $M_E^P(s) \leftarrow M_E^P(s) \cup \{a\}$

return M_E^P

4.2 Pre-Marking Direction による評価

Pre-Marking Direction（PMD）は，展開対象となるアクション $\hat{a}$ の有望さを，そのアクションを実行した結果生じる遷移先の状態 $\boldsymbol{s'}$ に基づいて評価する．この評価計算においては，各 LTS の静的構造解析により事前に算出された最短距離関数 CalculateDist が用いられる．RA が探索ステップごとに Ready アクショングラフ上の動的な最短経路探索を必要とするのに対し，PMD は事前計算された距離関数の参照と単純な比較演算によって評価値を算出するため，探索時における計算負荷を大幅に低減している．

具体的な評価手順をアルゴリズム 5 に示す．本手法では，まずアクション $\hat{a}$ を実行した後の状態 $\boldsymbol{s'}$ を導出し，その時点で未発火である PMA の集合 $A^P_{\boldsymbol{s'}}$ を特定する．これは，その遷移を選んだ場合に依然として回避できない必須タスクの残存量を表す．

続いて，各コンポーネント E_i について，自身の目標である Marking アクション，およびシステム全体で必要とされる未発火 PMA 群への到達距離を評価する．ここで，各コンポーネントの推定距離 d_{E_i}

は，自身の Marking アクションへの最短距離と，自身が関与すべき未発火 PMA への最短距離に 1 を加えた値のうち，より大きな値として算出される．この加算処理は，PMA が目標達成のための通過点として定義されており，その発火後も Marking アクションに至るまでに少なくとも 1 ステップの遷移を要するという構造的制約を反映したものである．この最大値をとる操作により，たとえ自身の局所的な目標が近くても，遠方にある同期アクションの発火に参加しなければならない場合，その移動と待機がシステム全体の進行を律速する要因となることを数理的に表現している．

算出された各コンポーネントの距離 d_{E_i} は降順にソートされ，辞書式順序による比較に用いられる．これにより，システム内で最も遅延しているボトルネックを優先的に解消する遷移が選択されることになる．

4.3 探索挙動の改善例：金属加工システム

本節では，第 3.1.2 項で述べた金属加工システムの例題に対し，提案手法である PMD を適用した場合の探索挙動を示す．RA が陥った局所解への迷走を，PMD がいかにして回避し，同期に不可欠なアクションを見据えた適切な探索を行うかについて詳述する．

4.3.1 Pre-Marking アクションの抽出結果

探索に先立ち，PMD は各 LTS の構造解析を行い，Pre-Marking アクション（PMA）の抽出を行う．本システムにおいて抽出される PMA は以下の通りである．

まず，環境モデル E_{env} においては，初期状態から目標アクション「出荷」へ至るすべての経路において，最初のアクションである「金属準備」が必ず実行される．したがって，「金属準備」は E_{env} における PMA として抽出される．なお，「成形」などの加工アクションは，E_{env} 単体で見ると「研磨」ルートによる回避が可能であるため，環境モデルの PMA とはならない．

次に，要求モデル E_{req} においては，初期状態から「出荷」へ至る適法なトレースにおいて，アクション「成形」は必ず一度は発火されなければならない．「成形」を回避して「出荷」を行う経路は，違反状態へ遷移する経路のみであるため，「成形」は E_{req} における PMA として正しく抽出される．

この結果，探索開始時の初期状態において，システム全体の未発火 PMA 集合は $\{$ 金属準備, 成形 $\}$ となる．

4.3.2 分岐点におけるヒューリスティック評価の比較

RA が誤った選択を行った分岐点である状態 $\boldsymbol{s}_1 = (1, 0, 0)$ における PMD の挙動を追跡する．この状態に至る遷移で「金属準備」は既に実行されているため，ここでの課題は残る PMA である「成形」をいかに効率よく消化するかにある．

この状態において展開可能なアクションは「研磨」,「皮膜」,「溶解」の 3 つである．なお，要求モデル E_{req} は，これらいずれのアクションによっても状態遷移せず初期状態（状態 0）に留まる．そのため，E_{req} 側での評価値はすべての候補で一定となり，選択の差異要因とはならない．したがって，以下では環境モデル E_{env} 側の遷移と評価値に焦点を当てて比較を行う．

PMD は展開可能な各アクション $\hat{a}$ を選択した後の次状態 $\boldsymbol{s}'$ を導出し，その時点での未発火 PMA への距離を考慮して以下の式で距離 $d_{E_{env}}$ を算出する．

$$d_{E_{env}} = \max(\textsc{CalculateDist}(s'_{E_{env}}, \text{出荷}),\quad \textsc{CalculateDist}(s'_{E_{env}}, \text{成形}) + 1)$$

選択肢 A：「研磨」または「皮膜」

これらのアクションを選択した場合，環境モデル E_{env} は状態 4 へ遷移する．この遷移後の状態

アルゴリズム 5 Pre-Marking Direction によるヒューリスティック関数

入力： 全ての LTS の組 $\boldsymbol{E} = (E_1, E_2, \ldots, E_n)$,
現在の状態 $\boldsymbol{s} = (s_{E_1}, s_{E_2}, \ldots, s_{E_n})$,
展開対象のアクション $\hat{a}$

出力： 各 LTS における距離の降順の組 $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)$
ただし，$d_i \in \mathbb{N} \cup \{\infty\}$

function PreMarkingDirectionHeuristic($\boldsymbol{E}, \boldsymbol{s}, \hat{a}$)
  if $\hat{a} \in A^M$ **then**
    return $(0, 0, \ldots, 0)$ ▷ Marking アクションの場合，距離 0（最優先）を返す

  ▷ Step 1: アクション $\hat{a}$ 実行後の次状態 $\boldsymbol{s}'$ を導出 ◁
  for $i = 1, \ldots, n$ **do**
    if $\hat{a} \in A_{E_i}$ **then**
      $s'_{E_i} \leftarrow s'$ where $(s_{E_i}, \hat{a}, s') \in \Delta_{E_i}$
    else
      $s'_{E_i} \leftarrow s_{E_i}$
  $\boldsymbol{s}' \leftarrow (s'_{E_1}, s'_{E_2}, \ldots, s'_{E_n})$

  ▷ Step 2: 次状態 $\boldsymbol{s}'$ における未発火 PMA 集合の特定 ◁
  $A^P_{\boldsymbol{s}'} \leftarrow \emptyset$
  for $i = 1, \ldots, n$ **do**
    $A^P_{\boldsymbol{s}'} \leftarrow A^P_{\boldsymbol{s}'} \cup M^P_{E_i}(s'_{E_i})$

  ▷ Step 3: 各コンポーネントのボトルネック距離推定 ◁
  for $i = 1, \ldots, n$ **do**
    $d_{E_i} \leftarrow 1$
    ▷ 自身の Marking アクションへの最短距離で更新 ◁
    if $A^M_{E_i} \neq \emptyset$ **then**
      $\mathit{minSteps} \leftarrow \min_{a_m \in A^M_{E_i}}$ CalculateDist(s'_{E_i}, a_m)
      $d_{E_i} \leftarrow \mathit{minSteps}$
    ▷ 未発火の PMA への距離によって更新 ◁
    if $A^P_{\boldsymbol{s}'} \cap A_{E_i} \neq \emptyset$ **then**
      $\mathit{maxSteps} \leftarrow \max_{a_p \in A^P_{\boldsymbol{s}'} \cap A_{E_i}}$ CalculateDist$(s'_{E_i}, a_p) + 1$
      $d_{E_i} \leftarrow \max(d_{E_i}, \mathit{maxSteps})$

  return SortDescending$((d_{E_1}, d_{E_2}, \ldots, d_{E_n}))$ ▷ 距離を降順でソートして返す

においても PMA「成形」は未発火のままであるため，PMD はそこへの距離を計算に含める．

目標（出荷）への距離　直後の遷移で「出荷」が可能であるため，距離は 1 である．

未発火 PMA（成形）への距離　状態 4 から「成形」へ到達するためには，一度「出荷」を行って初期状態へ戻り，再度「金属準備」「溶解」を経る必要がある．この最短パス（出荷 → 金属準備 → 溶解 → 成形）の長さは 4 である．

したがって，評価値は $d_{E_{env}} = \max(1, 4+1) = 5$ と算出される．

選択肢 B：「溶解」

このアクションを選択した場合，環境モデル E_{env} は状態 2 へ遷移する．同様に PMA「成形」は未発火であるが，状態が変化したことでそこへの距離が短縮される．

目標（出荷）への距離　状態 2 から「出荷」へ至るには，「成形」「冷却」を経て待機状態に戻り，そこから「研磨」等を経由する必要がある．この最短パス（成形 → 冷却 → 研磨 → 出荷）の長さは 4 である．

未発火 PMA（成形）への距離　直後の遷移で「成形」が可能であるため，距離は 1 である．

したがって，評価値は $d_{E_{env}} = \max(4, 1+1) = 4$ と算出される．

4.3.3　探索の結果

算出された推定距離を比較すると，「研磨・皮膜」の評価値 5 に対し，「溶解」の評価値は 4 となり，より小さい値を示す．PMD は，見かけ上は目標（出荷）に近い「研磨」ルートに対し，未発火 PMA（成形）への到達が困難になる（初期状態への手戻りが必要になる）という構造的理由からペナルティを与え，逆に目標へは遠回りでも PMA を確実に消化できる「溶解」ルートを正しく推奨する．

結果として，探索アルゴリズムはバックトラックを発生させることなく，初手から正解ルートである「溶解」→「成形」→「冷却」→「出荷」の経路を選択する．これにより，RA で発生していた不要な探索空間の展開が完全に抑制され，最短ステップでの解の発見が実現される．

4.4　評価実験

複数のシナリオにおいて，PMD・RA・BFS の探索状態数と計算時間を比較し，あわせて各問題での PMA 抽出数も記載する．結果として，PMA が抽出された問題では PMD が RA を大幅に上回り，逆に PMA がない場合は RA が有利あるいは同等となる傾向を示す記述を行う．（注：現在数値を再計測中だが，傾向は変わらないため構成確認用として一旦旧データを用いて記述する）

4.4.1　実験設定

4.4.2　実験結果

4.5　考察

本実験の結果は，提案手法である PMD の有効性が Pre-Marking アクション（PMA）の抽出可否に強く依存することを示している．

PMA が抽出された問題において，PMD は RA と比較して探索効率を大幅に向上させた．この要因として，PMA がシステム全体のボトルネックとなる同期イベントを捉え，必須アクションを含む経路を優先的に探索できたことが挙げられる．また，探索空間の削減率が限定的であった場合でも，計算時間の短縮が見られたケースが存在する．これは，RA が各探索ステップにおいて Ready アクショングラフ上の動的な最短経路探索を必要とするのに対し，PMD は事前計算済みの静的な距離表を参照するの

みであり，1 状態あたりの計算負荷が低く抑えられたことに起因する．すなわち，PMD は探索方向の的確な示唆とヒューリスティック計算の軽量性という二つの側面から高速化に寄与していると言える．

一方，PMA が抽出されなかった問題においては，PMD の探索性能は RA を下回る結果となった．PMA が存在しない場合，PMD の距離推定は各コンポーネントにおける Marking アクションまでの単純な静的最短距離に依存することになる．この指標は，コンポーネント間の同期による待ち時間やデッドロックの可能性を考慮できないため，Ready アクション間の遷移コストを評価できる RA と比較して推定精度が劣る．その結果，同期制約によって本来は進行不可能な方向や，目標から遠ざかる方向を有望と誤認して探索を行い，RA であれば回避できた不要な探索枝を展開してしまう傾向が見られた．このように，PMA という強力な指針を欠いた状態での PMD は，RA よりも不正確な探索を行うだけでなく，時間およびメモリ消費量の増大を招くリスクがあることが明らかとなった．

以上のことから，PMD は常に RA より優れているわけではなく，PMA という構造的特徴が存在する場合に特化した手法であると結論付けられる．この特性は，対象問題の構造に応じて適切なアルゴリズムを選択する必要性を示唆している．

4.6　RA との動的切り替え戦略

PMA 抽出にかかる計算コストのテーブルを提示し，それが探索全体の時間に比べて無視できるほど軽量であることを示す．このデータに基づき，合成開始時にまず静的解析（PMA 抽出）を実行し，有効な PMA が得られた場合のみ PMD を適用し，そうでない場合は RA を選択するという「動的切り替え戦略」について論じる．

第 5 章

関連研究

以下の 3 点について触れ，本研究の立ち位置を明確にする．(1) 離散制御器合成（DCS）における状態空間爆発への対処手法の変遷，(2) 本研究の直接的な比較対象である Ready Abstraction（RA）の理論的背景，(3) 近年の機械学習を用いたヒューリスティック手法との対比.

第 6 章

結論

6.1　本論文のまとめ

本論文で提案した PMD の総括を行う．特に，同期構造に着目した静的解析が，特定の構造を持つ問題に対して劇的な探索効率化を実現した点をまとめる．

6.2　将来研究

本研究では，PMD の導入によりヒューリスティック値の計算コストを大幅に削減し，同期ボトルネックを持つ問題における探索効率を改善した．しかし，実験を通じて，探索済み状態数が増大するにつれて，ヒューリスティック計算以外の処理に起因する探索速度の鈍化が確認された．これは，現在の On-The-Fly 探索アルゴリズムにおいて，既訪問状態の照合や新規状態の保存といった管理コストが，探索空間の規模に対してスケーラブルでない可能性を示唆している．

したがって，今後の課題として，On-The-Fly 探索アルゴリズムそのものの改良が挙げられる．具体的には，状態保存に用いるデータ構造をより効率的なものへ刷新することや，探索の進行に伴う管理オーバーヘッドを削減するようアルゴリズムを見直すことが求められる．これにより，PMD の軽量性を活かしつつ，さらに大規模なシステムに対しても高速な制御器合成が可能になると考えられる．

謝辞

参考文献

[1] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, Vol. 129, No. 1, pp. 5–33, 2001.

[2] Daniel Ciolek, Victor Braberman, Nicolás D'Ippolito, and Sebastián Uchitel. Directed controller synthesis of discrete event systems: Taming composition with heuristics. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 4764–4769, 2016.

[3] Daniel Ciolek, Matias Duran, Florencia Zanollo, Nicolas Pazos, Julián Braier, Victor Braberman, Nicolas D' Ippolito, and Sebastian Uchitel. On-the-fly informed search of non-blocking directed controllers. *Automatica*, Vol. 147, No. C, January 2023.

[4] Jing Huang and Ratnesh Kumar. Directed control of discrete event systems for safety and nonblocking. *IEEE Transactions on Automation Science and Engineering*, Vol. 5, No. 4, pp. 620–629, 2008.

[5] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, Vol. 25, No. 1, pp. 206–230, 1987.

[6] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, Vol. 77, No. 1, pp. 81–98, 1989.