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有効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 Landmark Ready Abstraction 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局所的な経路選択が行われ，探索空間が不必要に拡大する．

網羅的な距離推定による計算資源の浪費

RA はヒューリスティック値を算出する際，現在の状態遷移に関与しないコンポーネントを含めた全てに対し，一律に距離推定を実行する．この網羅的な評価プロセスは，システム規模（コンポーネント数）に比例して冗長な計算を繰り返すことを意味し，大規模なシステムにおいて 1 ステップごとの計算コストを不必要に増大させる要因となる．

そこで本研究では，従来の RA の枠組みに，目標アクションの発火に不可欠なアクション（Pre-Marking アクション）を中間目標（ランドマーク）として組み込む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局所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．

Landmark Ready Abstraction の提案

目標アクションの発火に不可欠なアクションを Pre-Marking アクション（PMA）として定義し，これを探索の指針（ランドマーク）とする新たなヒューリスティック手法 Landmark Ready Abstraction（LRA）を提案する．LRA は，RA が持つ遷移コストの精密な見積もり能力と，PMA による大域的な方向付けを統合することで，同期ボトルネックを考慮した適切な経路選択を可能にする．さらに，距離推定を行う目標を，その時点で解決すべき PMA のみに厳選することで，1 ステップあたりの計算コストの効率化も図る．これにより，局所的な評価に基づく冗長な経路探索を抑制し，探索状態数および計算時間を大幅に削減することで，システム全体の合成効率を向上させる．

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，大域的な同期制約を考慮できずに局所解への停留や探索空間の拡大を招くメカニズムについて分析する．

第 4 章「Landmark Ready Abstraction」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不可欠なアクション（Pre-Marking アクション）の定義と抽出アルゴリズム，およびそれを用いたヒューリスティック関数について詳述する．また，具体的なモデルを用いた動作例を通じ，提案手法がいかにして同期ボトルネックを考慮した適切な経路選択を行うか，その動作原理を明らかにする．

第 5 章「評価」では，提案手法の有効性を検証するための評価実験について述べる．ベンチマーク問題を用いた実験により，従来手法との探索状態数および計算時間の比較を行う．さらに，実験結果に基づき，提案手法が有効に機能する条件や計算コストに関する考察を行い，実用的な適用指針について議論する．

第 6 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 7 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

本論文における離散制御器合成の定式化として，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．

定義 4（有効トレース集合）． LTS E において，状態 $s \in S_E$ を始点とし，アクション集合 $A \subseteq A_E$ のいずれかの発火をもって終了する有限トレースのうち，途中で違反状態 S_E^I を経由しないものの集合を $T_{s \to A}$ と定義する．

$$T_{s \to A} = \left\{ (s_0, a_0, \ldots, s_n) \in T_E \;\middle|\; s_0 = s \wedge a_{n-1} \in A \wedge \bigwedge_{i=0}^{n} s_i \notin S_E^I \right\}$$

この集合が空でない（$T_{s \to A} \neq \emptyset$）とき，状態 s から A へ安全に到達可能である．

2.1.2 同期アクションと並列合成

定義 5（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 6（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 7（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 8（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 Marking アクション集合 A^M の発火として定義する．一方，RA は Marked 状態への到達可能性に基づくヒューリスティックである．そのため，本研究の定式化を RA の枠組みに適用するにあたり，アクションの発火イベントを状態として保持する Marking 判定 LTS E_M を用いて，アクションベースの目標を等価な状態ベースの目標へと帰着させる．

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，$E_M = (S_{E_M}, s_{E_M,0}, S^I_{E_M}, S^M_{E_M}, A_{E_M}, A^M_{E_M}, \Delta_{E_M})$ は以下のように定義される：

- $S_{E_M} = \{0, 1\}$, $s_{E_M,0} = 0$, $S^I_{E_M} = \emptyset$, $S^M_{E_M} = \{1\}$
- $A_{E_M} = A$（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．

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） [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 $\boldsymbol{s} = (s_{E_1}, s_{E_2}, \ldots, s_{E_n})$,
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 ← EstimateDistToState(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 ← EstimateDistToState(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

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 EstimateDistToState 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 EstimateDistToState($\boldsymbol{s}, \hat{a}, s^*_{E_i}$)
  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

  ▷ 各 LTS 上で，現在の状態から発火可能なアクション（Ready アクション）を収集 ◁
  $A^R \leftarrow \bigcup_{i=1}^{n} \{a \mid \exists s, (s_{E_i}, a, s) \in \Delta_{E_i}, s \neq s_{E_i}, s \notin S^I_{E_i}\}$

  ▷ 候補アクション $\hat{a}$ から構造的に到達可能なアクション集合 A^* を構築 ◁
  $A^* \leftarrow \emptyset$
  for $i = 1, \ldots, n$ **do**
    $s'_{E_i} \leftarrow s'$ where $(s_{E_i}, \hat{a}, s') \in \Delta_{E_i}$
    if $s'_{E_i} \neq \bot \wedge s'_{E_i} \notin S^I_{E_i}$ **then**
      $A^* \leftarrow A^* \cup \{a^* \in A^R \mid T_{s'_{E_i} \to \{a^*\}} \neq \emptyset\}$

  ▷ Gap を加味した最短経路探索 ◁
  $d \leftarrow \min_{a^* \in A^*} \{\text{CalculateGap}(\boldsymbol{s}, \hat{a}, a^*) + \text{EstimateDistToState}(\boldsymbol{s}, a^*, s^*_{E_i})\}$
  ▷ 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 ◁
  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．

2.3.3 Marking アクションまでの距離推定

RA は，Marked 状態への到達距離を推定する枠組みとして提案されている．一方，本研究では目標を Marking アクションの発火として定式化するため，RA を本研究の目的関数と同じ観点で比較するには，Marked 状態ではなく Marking アクションまでの距離をどのように見積もるかを明示しておく必要がある．そこで本節では，Marking アクション集合 A^M に対する距離推定手続きとして EstimateDistToMarking を示す．

EstimateDistToMarking は，候補アクション $\hat{a}$ が Marking アクションであれば距離 1 を返し，そうでなければ，合成状態 $\boldsymbol{s}$ 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 を介して，Gap とその先での推定距離の和の最小値を再帰的に求める．構造はアルゴリズム 3

（EstimateDistToState）と同型であり，終端条件が Marked 状態ではなく Marking アクションになっている点が相違である．

なお，EstimateDistToMarking は，探索到達済み Marked 状態集合 $\boldsymbol{S}^M$ を基準とする Rank 0 の推定が利用できない状況，特に $\boldsymbol{S}^M = \emptyset$ の場合に用いることを想定している．

アルゴリズム 4 Ready Abstraction における Marking アクションまでの距離推定

入力： 現在の状態 $\boldsymbol{s} = (s_{E_1}, \ldots, s_{E_n})$,
候補のアクション $\hat{a}$

出力： 推定距離 $d \in \mathbb{N} \cup \{\infty\}$

function EstimateDistToMarking($\boldsymbol{s}, \hat{a}$)
  if $\hat{a} \in A^M$ **then**
    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
  return $\min_{a \in A^R \cup A^M} \{\text{CalculateGap}(\hat{a}, a) + \text{EstimateDistToMarking}(\boldsymbol{s}, a)\}$

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　同期構造の看過に起因する局所的な探索

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ず，大域的な最適性を損なう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 はその局所的な遷移コストの低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考慮できず，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局所最適化に基づいた評価を下すため，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

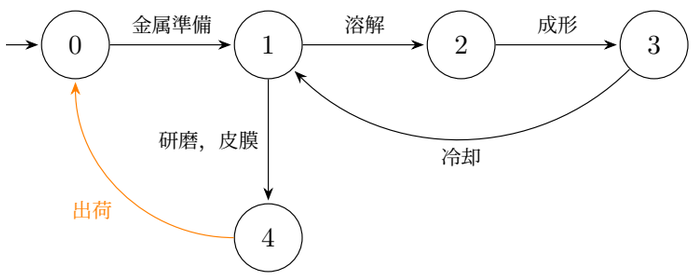
RA が不適切な誘導を行う具体的なシナリオとして，図 3.1 に示す金属加工システムのモデルを用いて説明する．

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}

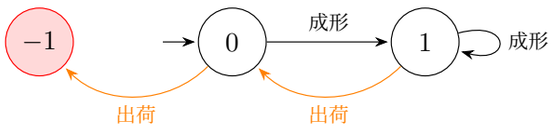


図 3.1　金属加工システムの環境モデルと要求モデル

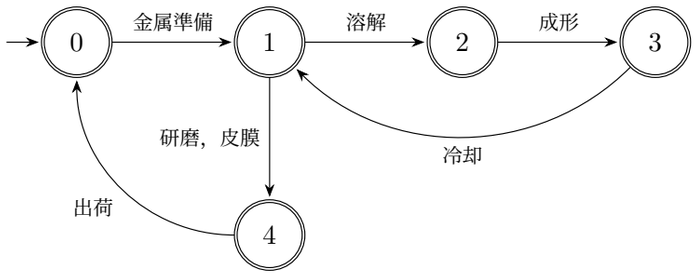
Ready Abstraction 適用のための Marked 状態への変換

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．

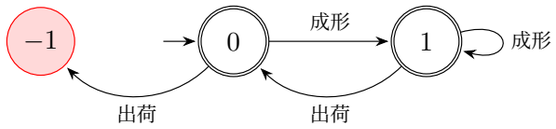
図 3.2 は，変換後のシステム構成を示している．ここでは，新たなコンポーネントとして Marking 判定モデル E_M を導入している．E_M は，「出荷」アクションが発火された直後のみ Marked 状態（状態 1）に遷移し，それ以外の場合は非 Marked 状態（状態 0）に留まる機能を持つ．いわば，アクションの発火イベントを状態遷移として表現するためのモデルである．

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

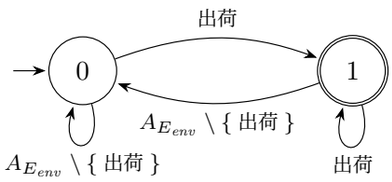


図 3.2　アクション目標を状態目標へ変換した金属加工システム

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，まずは局所的な算出コストが低い「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 のヒューリスティック値に従い 2 つの冗長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，このような冗長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 網羅的な距離推定による計算資源の浪費

RA のもう一つの課題は，ヒューリスティック関数の評価プロセスそのものに内在する計算の冗長性と，それに伴うコストの増大である．

前章のアルゴリズム 2 で示した通り，RA は 1 回の探索ステップ（ExpandNext）において，システムを構成する全てのコンポーネント E_i $(i = 1, \ldots, n)$ に対し，以下の 2 段階の評価を網羅的に実行する．

1. Rank 0 の評価: 探索中に到達済みの Marked 状態への距離を計算する．
2. Rank 1 の評価: 上記が到達不能な場合，モデル上の全ての Marked 状態への距離を計算する．

すなわち, アクションを 1 つ選択するたびに, 最大で $2n$ 回の距離推定関数（EstimateDistToState）が呼び出されることになる．ここで呼び出される距離推定関数は，第 2 章で述べた通り，Ready アクショングラフ上の探索を行う処理である．単一の計算は多項式時間で完了するが，状態や遷移に依存した動的な計算処理を伴う．

問題となるのは，この計算が現在の状態遷移に関与しないコンポーネントを含めた全てに対し，一律に実行される点である．大規模な並列システムにおいて，ある瞬間に状態遷移のボトルネックとなっているコンポーネントは全体の一部に限られ，他は待機状態や無関係な状態にあることが多い．しかし，

RA はそのような文脈を考慮せず，全てのコンポーネントに対して毎回同様に探索計算を行う．

システム規模が拡大しコンポーネント数 n が増加すると，この網羅的な探索プロセスは，探索の進行に寄与しない冗長な計算を繰り返すことを意味する．結果として，本来不要なヒューリスティック値の算出に計算機資源を費やすことになり，大規模システムにおける合成全体の効率を低下させる要因となる．

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．

第二の課題は計算の冗長性である．RA は距離推定のために，現在の状態に関与しないコンポーネントも含めて最大 $2n$ 回の計算を一律に行うため，システム規模に対して不必要な計算資源を浪費する．

これらの課題は，RA が同期構造に対する大域的な視点を欠いていることと，全コンポーネントを等しく評価対象とする網羅的な計算構造に起因する．次章では，これらの問題を解決するために，目標達成に不可欠なアクションを Pre-Marking アクション（PMA）として定義し，これを中間目標（ランドマーク）として導入する新たな手法「Landmark Ready Abstraction」を提案する．本手法は，PMA への到達を優先することで同期を適切に指向すると同時に，評価対象を解決すべき PMA のみに厳選することで計算の効率化を図る．

第 4 章

Landmark Ready Abstraction

前章で指摘した通り，Ready Abstraction (RA) は，並列合成されたシステムにおける局所的な遷移可能性を利用して距離を推定する強力な手法である．しかし，RA は現在の状態から局所的に遷移可能な方向を優先して評価するため，目標達成において将来的に不可避となる同期イベントを見落とし，非効率な探索経路を選択しやすいという構造的な課題を有する．

本章では，こ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 Landmark Ready Abstraction (LRA) を提案する．LRA は，RA が持つ動的なグラフ探索の枠組みを継承しつつ，システムの大域的な構造解析に基づいて抽出された中間目標としての Pre-Marking アクションを導入する．これにより，LRA は RA の利点である遷移コストの精密な見積もりと，静的解析による大域的な方向付けを統合し，複雑な同期構造を持つシステムに対しても効率的な探索を実現する．

4.1 Pre-Marking アクションの定義と階層構造

DCS において，探索の最終目標は Marking アクションの発火である．しかし，大規模な並列システムにおいて，初期状態から最終目標までの距離は膨大であり，単一の目標のみを指針とした探索は，広大な状態空間の中で方向を見失うリスクが高い．そこで本研究では，最終目標に至る過程で構造上回避することのできないアクションを Pre-Marking アクションとして定義し，これを最終目標へ至るための段階的な中間目標として利用する．

4.1.1 必須アクション

Pre-Marking アクションを定義するための基礎概念として，ある目標に対する不可避性を表す必須アクションを定義する．

必須アクションとは，ある状態から目標となるアクションに至るいかなるトレースを選択したとしても，その過程で必ず実行しなければならないアクションのことである．これは，システムが目標に到達するために回避することのできない，構造上のボトルネックや前提条件と解釈できる．

第 2 章で定義した有効トレース集合 $T_{s \to A^{Target}}$ を用いて，必須アクション集合 M_E^R を以下のように定義する．

定義 9（必須アクション）**．** 状態 s における A^{Target} に対する必須アクション集合 $M_E^R(s, A^{Target})$ は，目標へ至る全ての有効なトレースにおいて共通して出現するアクション（目標自身を除く）として定義される．

$$M_E^R(s, A^{Target}) = \left(\bigcap_{t \in T_{s \to A^{Target}}} A_t \right) \setminus A^{Target}$$

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．

この定義により，$M_E^R(s, A^{Target})$ に含まれるアクションが未発火である限り，s から A^{Target} への到達は不可能であることが保証される．

4.1.2 Pre-Marking アクションの階層定義

必須アクションの概念を用い，システムの最終目標から逆算された依存関係の階層を構築する．本手法では，階層を以下の 2 段階に区分して定義する．

定義 10（Primary Pre-Marking アクション）． システム全体の最終目標である Marking アクション集合 A^M に対する必須アクションを Primary Pre-Marking アクションと呼ぶ．各状態 s における Primary Pre-Marking アクション集合 $M_E^{PP}(s)$ は以下のように定義される．

$$M_E^{PP}(s) = M_E^R(s, A^M) \cap M_E^R(s_{E,0}, A^M)$$

この定義において，$M_E^R(s_{E,0}, A^M)$ は初期状態において大域的に特定される必須アクションであり，$M_E^R(s, A^M)$ は現在の状態 s から見て必須となるアクションである．これらの積集合をとることで，$M_E^{PP}(s)$ は，構造上あらかじめ決定されたボトルネックのうち，現時点で未解決のもののみを保持する集合となる．

定義 11（Secondary Pre-Marking アクション）． ある特定の Pre-Marking アクション a^* に対する必須アクションを Secondary Pre-Marking アクションと呼ぶ．各状態 s において，特定の a^* に対して必須となるアクション集合 $M_E^{SP}(s, a^*)$ は以下のように定義される．

$$M_E^{SP}(s, a^*) = M_E^R(s, \{a^*\}) \cap M_E^R(s_{E,0}, \{a^*\})$$

ここで，Primary Pre-Marking アクションがいずれかの Marking アクションへの到達という選言的な目標に対する必須要件であるのに対し，Secondary Pre-Marking アクションは特定の Pre-Marking アクション a^* への到達という単一の目標に対する必須要件である．本研究では，これらを総称して Pre-Marking アクション（PMA）と呼び，探索の指針として用いる．

4.2 Pre-Marking アクションの抽出アルゴリズム

前節の定義に基づき，各状態における PMA を抽出するアルゴリズムについて述べる．なお，本アルゴリズムは探索開始前に一度だけ実行される静的解析である．定義に従って全ての状態とアクションの組み合わせを網羅的に検査することは，計算コストの観点から現実的ではない．そのため，本手法では以下の 2 段階からなる効率的な抽出手法を採用する．

4.2.1 必須アクションの抽出

まず，任意の目標に対する必須アクションを抽出する汎用的な手続きについて述べる．本手続きは，計算効率を考慮し，最短パス情報による候補集合の限定と，逆方向探索による必須性の判定という 2 つの工程で構成される．

Step 1: 最短パスによる候補集合の限定

必須アクションは，目標に至るいかなる経路にも含まれるという性質を持つため，初期状態から目標への最短パス上にも必ず存在する．したがって，初期状態から目標への最短パスを算出し，そこに含まれるアクションのみを必須アクションの候補とすることで，検査対象となるアクション数を大幅に削減できる．

Step 2: 逆方向探索による必須性の判定

限定された候補アクション a が真に不可避であるかを確認するため，アクション a を経由せずに目標へ到達可能か否かを判定する．これは，目標のアクションを発火可能な状態集合を起点とし，遷移を逆方向に辿る探索を行うことで効率的に実行できる．この逆探索によって初期状態 $s_{E,0}$ に到達可能である場合，アクション a を回避する経路が存在することを意味するため，a は必須ではないと判断される．逆に，$s_{E,0}$ に到達できない場合，a は初期状態において目標アクションの発火のための必須アクションであると確定する．

このアルゴリズムをアルゴリズム 5 に示す．なお，本アルゴリズムは PMA の抽出を目的としているため，初期状態において必須でないアクションは，各状態における必須性に関わらず除外する．したがって，得られる結果 M は，必須アクションの定義における $M_E^R(s, A^{Target}) \cap M_E^R(s_{E,0}, A^{Target})$ の条件を満たす集合となる．

アルゴリズム 5 必須アクションの抽出

入力： LTS $E = (S_E, s_{E,0}, S_E^I, A_E, \Delta_E)$，目標アクション集合 $A^{Target} \subseteq A_E$

出力： 各状態における必須アクション集合 $M : S_E \to 2^{A_E}$

ただし $M(s) = M_E^R(s, A^{Target}) \cap M_E^R(s_{E,0}, A^{Target})$

function ExtractRequiredActions(E, A^{Target})
  for all $s \in S_E$ **do** $M(s) \leftarrow \emptyset$

  ▷ Step 1: 最短パス情報による候補集合の限定 ◁
  $\hat{A}_E \leftarrow A_E$
  for all $a^* \in A^{Target}$ **do**
    $t \leftarrow \mathrm{argmin}_{t \in T_{s_{E,0} \to \{a^*\}}} |t|$
    $\hat{A}_E \leftarrow \hat{A}_E \cap A_t$ ▷ 全ての目標への最短パスに含まれるアクションのみを候補とする

  ▷ Step 2: 逆方向探索による必須性の判定 ◁
  for $\hat{a} \in \hat{A}_E$ **do**
    ▷ アクション $\hat{a}$ を通らずに目標に到達可能な状態（不要状態）を探索 ◁
    $S^* \leftarrow \{s \mid \exists a^* \in A^{Target}, \exists s', (s, a^*, s') \in \Delta_E \wedge s' \notin S_E^I\}$
    $S_{new}^* \leftarrow S^*$
    while $S_{new}^* \neq \emptyset$ **do**
      $S_{next}^* \leftarrow S^*$
      for all $s^* \in S_{new}^*$ **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 $a \neq \hat{a}$ **then**
            $S_{next}^* \leftarrow S_{next}^* \cup \{s\}$
      $S_{new}^* \leftarrow S_{next}^* \setminus S^*$
      $S^* \leftarrow S_{next}^*$
    ▷ 初期状態から不可避であれば，必須状態に $\hat{a}$ を割り当て ◁
    if $s_{E,0} \notin S^*$ **then**
      for $s \in S_E \setminus S^*$ **do**
        $M(s) \leftarrow M(s) \cup \{\hat{a}\}$
  return M

4.2.2 階層構造の構築

次に, アルゴリズム 5 で定義した必須アクション抽出関数を用い, システム全体における Pre-Marking アクションの階層構造を構築する．本アルゴリズムは，Primary Pre-Marking アクションの抽出と，それに続く Secondary Pre-Marking アクションの連鎖的な抽出の 2 段階からなる．

第一段階では，Marking アクション集合 A^M を目標として Primary PMA M_E^{PP} を抽出する．ここで，Marking アクションが特定のコンポーネントのみに関連する場合や同期アクションである場合を考慮し，Marking アクション集合全体を含む LTS のみを抽出対象とする．これにより，目標との関連性が薄いコンポーネントにおける誤検出を回避する．

第二段階では，抽出された Primary PMA を起点として，Secondary PMA M_E^{SP} の抽出を行う．こ

こでは，あるコンポーネントで発見された必須アクションが，同期を介して他のコンポーネントにおける必須アクションとなる依存関係を網羅する必要がある．そのために，不動点反復に基づくアプローチを採用する．具体的には，初期状態で有効な PMA を探索すべき目標集合として管理し，未処理の目標に対して各 LTS で必須アクション抽出を実行する．そこで新たに初期状態で有効な必須アクションが発見された場合，それを新たな目標として集合に追加する．この操作を新たな目標が発見されなくなるまで繰り返すことで，システム全体に跨る多段階の依存関係を漏らさず抽出する．

このシステム全体に対する階層構築アルゴリズムをアルゴリズム 6 に示す．

アルゴリズム 6 Pre-Marking アクション階層の構築

入力： 全ての LTS の組 $\boldsymbol{E} = (E_1, E_2, \ldots, E_n)$

出力： 各 LTS の各状態における Primary Pre-Marking アクション集合 $M_E^{PP} : S_E \to 2^{A_E}$，
各 LTS の各状態における Secondary Pre-Marking アクション集合 $M_E^{SP} : (S_E \times A_E) \to 2^{A_E}$

function ExtractPreMarkingActions($\boldsymbol{E}$)
  ▷ Step 1: Primary Pre-Marking アクションの抽出 ◁
  $A^M \leftarrow \bigcup_{i=1}^{n} A_{E_i}^M$ ▷ 全 LTS の Marking アクション集合の和集合
  for $i = 1, \ldots, n$ **do**
    if $A^M \subseteq A_{E_i}$ **then**
      ▷ Marking アクションを全て含む LTS においてのみ抽出 ◁
      $M_{E_i}^{PP} \leftarrow$ ExtractRequiredActions(E_i, A^M)
    else
      for all $s \in S_{E_i}$ **do** $M_{E_i}^{PP}(s) \leftarrow \emptyset$

  ▷ Step 2: Secondary Pre-Marking アクションの連鎖的抽出 ◁
  $A^* \leftarrow \bigcup_{i=1}^{n} M_{E_i}^{PP}(s_{E_i,0})$ ▷ 初期状態で有効な Primary PMA が初期目標
  $A_{new}^* \leftarrow A^*$ ▷ 新たに追加された目標集合
  while $A_{new}^* \neq \emptyset$ **do** ▷ 新たな目標が追加されなくなるまで反復
    for all $a^* \in A_{new}^*$ **do** ▷ 未処理の目標アクションについて
      for $i = 1, \ldots, n$ **do**
        if $a^* \in A_{E_i}$ **then**
          ▷ 目標アクション a^* を含む LTS においてのみ抽出 ◁
          $M \leftarrow$ ExtractRequiredActions$(E_i, \{a^*\})$
          for all $s \in S_{E_i}$ **do** $M_{E_i}^{SP}(s, a^*) \leftarrow M(s)$
        else
          for all $s \in S_{E_i}$ **do** $M_{E_i}^{SP}(s, a^*) \leftarrow \emptyset$
    ▷ 新たに発見された PMA を次回の目標とする ◁
    $A_{new}^* \leftarrow \left(\bigcup_{a^* \in A_{new}^*} \bigcup_{i=1}^{n} M_{E_i}^{SP}(s_{E_i,0}, a^*) \right) \setminus A^*$
    $A^* \leftarrow A^* \cup A_{new}^*$
  return $M_{E_i}^{PP},\ M_{E_i}^{SP} \quad (i = 1, \ldots, n)$

4.3 Landmark Ready Abstraction による評価

本節では，抽出された PMA を用い，各状態の有望さを定量的に評価するヒューリスティック関数について述べる．従来の Ready Abstraction (RA) は，最終目標への幾何的な距離のみを評価指標として

おり，その到達過程で構造上不可避となる中間のボトルネックを考慮していない．そのため，局所的には目標に近づいているように見えても，実際には必須となる手順を回避してしまい，後の探索で行き詰まるような経路を高く評価してしまう可能性がある．

これに対し Landmark Ready Abstraction (LRA) では，抽出された PMA を，最終目標に至るために経由しなければならない中間目標（ランドマーク）として扱う．未解決の PMA を距離計算の対象に含めることで，ボトルネックの解消にかかるコストを評価値に反映し，より精度の高い距離推定を実現する．LRA は距離推定の実行対象を，その時点で解決すべき PMA および最終目標のみに限定するため，探索 1 ステップあたりの計算コストを抑制できるという利点も持つ．

4.3.1 ヒューリスティック関数の構成

LRA におけるヒューリスティック関数 LandmarkReadyAbstractionHeuristic の全体構成について述べる．本関数は，最終目標である Marking アクションへの到達距離を基準としつつ，抽出された PMA への距離を評価に組み込むことで，構造的なボトルネックを反映した値を算出する．

具体的な計算手順は以下の通りである．まず，基本となる評価値として，システムの最終目標である Marking アクション集合 A^M への到達距離を算出する．これにより，障害物がない理想的な状況下での最短ステップ数が得られる．

次に，現在の状態において解決すべき PMA 集合 A^P を構築する．この集合は，現在の状態で必須となっている Primary PMA を起点とし，そこから依存関係にある Secondary PMA を再帰的に探索することで生成される．具体的には，Primary PMA 集合に対し，各アクションの前提となる Secondary PMA を追加する操作を，新たなアクションが追加されなくなるまで繰り返す．これにより，Primary 層だけでなく，より深い階層にある潜在的なボトルネックも網羅的に収集される．

最後に，収集された各 PMA $a \in A^P$ について到達距離を算出し，これらを現在の推定距離と比較して最大値を更新する．ここで，PMA への距離に 1 を加算しているのは，PMA が最終目標に至るための通過点であり，その達成後も少なくとも 1 ステップ以上の遷移が必要であることを評価値に反映させるためである．並列システムの完了時間は，最も解決に時間を要するボトルネック工程によって律速されるため，これらの距離の最大値を採用することで，システム全体としての実質的な残り距離を見積もる．

このアルゴリズムをアルゴリズム 7 に示す．なお，関数 EstimateDistToActions は次項で定義する距離推定関数であり，指定された状態から目標アクション集合への Gap を考慮した距離を返すものとする．

アルゴリズム 7 Landmark Ready Abstraction によるヒューリスティック関数

入力： 全ての LTS の組 $\boldsymbol{E} = (E_1, E_2, \ldots, E_n)$,
Primary Pre-Marking アクション写像 $\boldsymbol{M}^{PP} = (M^{PP}_{E_1}, \ldots, M^{PP}_{E_n})$,
Secondary Pre-Marking アクション写像 $\boldsymbol{M}^{SP} = (M^{SP}_{E_1}, \ldots, M^{SP}_{E_n})$,
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： 推定距離 $d \in \mathbb{N} \cup \{\infty\}$

function LandmarkReadyAbstractionHeuristic($\boldsymbol{E}, \boldsymbol{M}^{PP}, \boldsymbol{M}^{SP}, \boldsymbol{s}, \hat{a}$)
  ▷ Step 1: 最終目標（Marking アクション）への距離計算 ◁
  $A^M \leftarrow \bigcup_{i=1}^{n} A^M_{E_i}$
  $d \leftarrow$ EstimateDistToActions($\boldsymbol{s}, \hat{a}, A^M$)
  ▷ Step 2: Pre-Marking アクション集合の構築と距離更新 ◁
  $A^P \leftarrow \bigcup_{i=1}^{n} M^{PP}_{E_i}(s_{E_i})$
  $A^P_{new} \leftarrow A^P$
  while $A^P_{new} \neq \emptyset$ **do**
    $A^P_{new} \leftarrow \left(\bigcup_{a \in A^P_{new}} \bigcup_{i=1}^{n} M^{SP}_{E_i}(s_{E_i}, a)\right) \setminus A^P$
    $A^P \leftarrow A^P \cup A^P_{new}$
  for all $a \in A^P$ **do**
    $d' \leftarrow$ EstimateDistToActions($\boldsymbol{s}, \hat{a}, \{a\}$)
    $d \leftarrow \max(d, d' + 1)$
  return d

4.3.2 アクション集合への距離推定

前節のヒューリスティック関数内で用いられる EstimateDistToActions は，現在の合成状態 $\boldsymbol{s}$ において，ある候補アクション $\hat{a}$ を実行した後，指定された目標アクション集合 A^{Target} のいずれかを発火するまでに必要な最小ステップ数を推定する関数である．

RA における距離推定（アルゴリズム 3）が，目標となる状態集合への到達距離を算出するのに対し，本関数は目標となるアクション集合への到達距離を算出する点が異なる．計算の基本構造は RA と共通しており，現在の状態で発火可能なアクション（Ready アクション）を介して目標へ至る経路を探索し，その過程で生じる遷移コスト（Gap）と再帰的な距離の和を最小化することで値を求める．具体的な手続きをアルゴリズム 8 に示す．

アルゴリズムの手順は以下の通りである．まず，候補アクション $\hat{a}$ 自体が目標集合 A^{Target} に含まれる場合，距離は 1 と判定される．そうでない場合，現在の状態で発火可能な Ready アクション集合 A^R を起点として探索を行う．具体的には，$\hat{a}$ から各 Ready アクション $a \in A^R$ への切り替えコスト（Gap）と，その a から目標までの再帰的な距離の和を計算し，その最小値を推定距離とする．

ここで用いられる CalculateGap($\boldsymbol{s}, \hat{a}, a$) は，RA と同様に，各コンポーネントにおいて $\hat{a}$ から a へ至るための局所的な最大コストに基づいて算出される．この再帰的な定義により，同期による待機やアクションの切り替えを考慮した，目標までの実質的な距離が見積もられる．

アルゴリズム 8 目標アクション集合への距離推定

入力： 現在の状態 $\boldsymbol{s} = (s_{E_1}, \ldots, s_{E_n})$,
候補のアクション $\hat{a}$,
目標アクション集合 $A^{Target} \subseteq \bigcup_{i=1}^{n} A_{E_i}$

出力： 推定距離 $d \in \mathbb{N} \cup \{\infty\}$

function EstimateDistToActions($\boldsymbol{s}, \hat{a}, A^{Target}$)
  if $\hat{a} \in A^{Target}$ **then**
    return 1
  ▷ 各 LTS 上で，現在の状態から発火可能なアクション（Ready アクション）を収集 ◁
  $A^R \leftarrow \bigcup_{i=1}^{n} \{a \mid \exists s, (s_{E_i}, a, s) \in \Delta_{E_i}, s \neq s_{E_i}, s \notin S^I_{E_i}\}$

  ▷ 候補アクション $\hat{a}$ から構造的に到達可能なアクション集合 A^* を構築 ◁
  $A^* \leftarrow \emptyset$
  for $i = 1, \ldots, n$ **do**
    $s'_{E_i} \leftarrow s'$ where $(s_{E_i}, \hat{a}, s') \in \Delta_{E_i}$
    if $s'_{E_i} \neq \bot \wedge s'_{E_i} \notin S^I_{E_i}$ **then**
      $A^* \leftarrow A^* \cup \{a^* \in A^R \cup A^{Target} \mid T_{s'_{E_i} \to \{a^*\}} \neq \emptyset\}$

  ▷ 依存関係にあるアクションの中から最短経路を探索 ◁
  $d \leftarrow \min_{a^* \in A^*} \{\text{CalculateGap}(\boldsymbol{s}, \hat{a}, a^*) + \text{EstimateDistToActions}(\boldsymbol{s}, a^*, A^{Target})\}$
  return d

4.4 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 LRA を適用した場合の探索挙動を示す．RA において発生した冗長な探索を，LRA 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．

4.4.1 Pre-Marking アクションの抽出結果

探索に先立ち，LRA は各 LTS の構造解析を行い，Pre-Marking アクション（PMA）の抽出と階層化を行う．

まず，最終目標に対する必須アクションである Primary PMA の抽出を行う．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 Primary PMA として抽出される．一方，要求モデル E_{req} については，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」が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ため，「成形」は E_{req} における Primary PMA として抽出される．

続いて，PMA に対する必須アクションである Secondary PMA の抽出を行う．抽出された Primary PMA のうち，未解決の依存関係を持つ「成形」に着目すると，環境モデル E_{env} において「成形」を実行可能な状態へ到達するためには，その前段階として「溶解」が必ず実行されなければならない．したがって，「溶解」は「成形」に対する Secondary PMA として抽出される．

以上の解析により，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, 溶解 } となり，これらが LRA の評価対象集合 A^P に含まれることになる．

4.4.2　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 LRA の挙動を追跡する．この状態に至る遷移で「金属準備」は既に実行済みであるため，ここでの課題は残る PMA である「成形」および「溶解」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．LRA は，これらの候補アクション $\hat{a}$ それぞれについて，実行後の状態からの最終目標（「出荷」）への距離と未消化 PMA（「成形」・「溶解」）への距離を，全コンポーネントの同期構造を考慮した Gap 計算（EstimateDistToActions）により算出し，その最大値を評価値 d とする．

すなわち，評価値は以下の式に基づき決定される．

$$d = \max \begin{pmatrix} \textsc{EstimateDistToActions}(\boldsymbol{s}_1, \hat{a}, \{\text{ 出荷 }\}), \\ \textsc{EstimateDistToActions}(\boldsymbol{s}_1, \hat{a}, \{\text{ 成形 }\}) + 1, \\ \textsc{EstimateDistToActions}(\boldsymbol{s}_1, \hat{a}, \{\text{ 溶解 }\}) + 1 \end{pmatrix}$$

各選択肢における評価の詳細を以下に示す．

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 へ遷移し，要求モデル E_{req} は状態 0 に留まる．

まず，目標（「出荷」）への距離について確認する．この遷移後の状態において，「研磨」または「皮膜」と「出荷」の間の Gap（遷移ステップ数）は 1 である．これに目標アクション自身のコスト 1 を加算し，推定距離は $1 + 1 = 2$ となる．

次に，PMA（「成形」・「溶解」）への距離について確認する．状態 4 から「溶解」や「成形」へ到達するためには，構造的な迂回（「出荷」→「金属準備」→ ...）が必要となる．LRA の距離推定はこのコストを検出し，それぞれの距離を以下のように算出する．「成形」までの Gap は 4 であり，目標コスト 1 を加えて推定距離は 5 となる．「溶解」までの Gap は 3 であり，目標コスト 1 を加えて推定距離は 4 となる．

結果として，最も遠い PMA（「成形」）への項が支配的となり，LRA 評価値は以下のように算出される．

$$d = \max(2, 5, 4) = 5$$

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．

まず，目標（「出荷」）への距離について確認する．状態 2 から「出荷」へ至る最短パス（「成形」→「冷却」→「研磨」→「出荷」）に基づき，Gap は 3 となる．これに目標コスト 1 を加算し，推定距離は 4 となる．

次に，PMA（「成形」・「溶解」）への距離について確認する．「溶解」自体が PMA であるため，推定距離は 1 と算出される．また，「成形」については，直後の状態からの Gap は 1 と計算される．これに目標コスト 1 を加え，推定距離は $1 + 1 = 2$ となる．

結果として，目標への距離の項が支配的となり，LRA 評価値は以下のように算出される．

$$d = \max(4, 1, 2) = 4$$

4.4.3　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．RA は目標（「出荷」）への局所的な近さのみを見て「研磨」（距離 2）を推奨したが，LRA はシステム全体として PMA（「成形」）への到達が困難になるコストを評価値に反映している．これにより，LRA は局所的な近道である研磨ルートを却下し，必須アクションを確実に消化できる「溶解」ルートを正しく選択する．

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」の経路を探索する．このように，LRA は抽出された PMA の階層構造と，全モデルを考慮した動的距離推定を組み合わせることで，複雑な同期構造を持つシステムにおいても効率的な探索を実現する．

第 5 章

評価

5.1 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）

5.1.1 実験設定

5.1.2 実験結果

5.2 考察

5.3 RA との動的切り替え戦略

第 6 章

関連研究

以下の 4 点について触れ，本研究の立ち位置を明確にする．(1) 離散制御器合成における状態空間爆発への対処手法の変遷，(2) On-The-Fly を用いた他の分野の話（状態爆発系），(3) RA や機械学習を用いたヒューリスティック手法との対比，(4) 他の分野における Landmark Heuristic.

第 7 章

結論

7.1　本論文のまとめ

本論文で提案した LRA の総括を行う．

7.2　将来研究

本研究では，LRA の導入により，評価すべきターゲットをシステムにとって真に不可欠な PMA に絞り込むことで，RA が抱える多重な距離計算の冗長性を解消し，探索効率を大幅に改善した．今後の展望として，以下の三つの方向性が挙げられる．

第一に，On-The-Fly 探索アルゴリズム自体のデータ構造とメモリ管理の最適化である．実験を通じて，LRA によって探索状態数が削減された場合でも，システム規模が極めて大きい場合には，既訪問状態の保存や照合にかかる管理コストが探索速度の律速要因となる傾向が確認された．現在の実装ではハッシュセットを用いた明示的な状態管理を行っているが，数百万状態を超える規模の探索においては，メモリ使用量とキャッシュ効率の観点で改善の余地がある．したがって，今後は BDD（二分決定グラフ）を用いたシンボリックな状態管理手法の導入や，探索済み状態の圧縮表現技術を適用することで，より大規模なシステムに対しても高速かつ省メモリな制御器合成を実現する必要がある．

第二に，PMA 抽出アルゴリズムの拡張と汎用化である．現行の手法では，目標達成に必ず発火しなければならないアクションを PMA として定義している．この定義は強力な剪定効果を持つ一方で，複数の加工ルートが存在する場合（OR 条件）や，確率的な分岐を含むシステムにおいては，必須アクションが存在せず PMA が抽出されないケースがある．今後は，必須性の条件を緩和し，複数の候補集合を選言的な中間目標として扱う枠組みや，静的解析と動的探索を組み合わせたより柔軟なボトルネック抽出手法を開発することで，LRA の適用範囲をさらに広げることが期待される．

第三に，探索履歴に基づいたヒューリスティック評価の高度化である．RA は，探索中に到達済みの Marked 状態への距離（Rank 0）を，未到達の Marked 状態への距離（Rank 1）よりも優先して評価する仕組みを持つ．この戦略は，探索を既知の安全なループへ誘導し，Non-Blocking 性を早期に確定させる上で有効に機能する場合がある．現在の LRA は，Marking アクションが複数存在する場合でも，それらを一律の目標集合として扱っている．そこで，LRA においても RA の知見を取り入れ，既に発火履歴のある Marking アクションへの指向性を強化する動的な重み付けを導入することで，探索の安定性と収束速度をさらに向上させることが可能であると考えられる．

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.