

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

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
3.1 同期構造に起因する近視眼的な探索	13
3.1.1 同期待ち時間の無視	13
3.1.2 具体的な不適合例：金属加工システム	13
3.2 Ready アクション探索による計算コストの増大	16
3.3 本章のまとめ	17
第 4 章 Pre-Marking Direction	18
第 5 章 関連研究	19
第 6 章 結論	20
6.1 本論文のまとめ	20
6.2 将来研究	20
謝辞	21
参考文献	22

図目次

3.1	金属加工システムの環境モデルと要求モデル	14
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	15

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 [1]。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 [2]。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 [3]。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の計算を採用する。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する。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。一般のスーパー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 $t = s_0, a_0, s_1, a_1, \dots$ 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, \dots, 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

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

と定義する。

定義 5 (並列合成) . LTS 集合 $\mathcal{E} = \{E_1, \dots, E_n\}$ の並列合成 $E_{\parallel} = E_1 \parallel E_2 \parallel \dots \parallel E_n$ を $E_{\parallel} = (S_{E_{\parallel}}, s_{E_{\parallel},0}, S_{E_{\parallel}}^I, S_{E_{\parallel}}^M, A_{E_{\parallel}}, A_{E_{\parallel}}^M, \Delta_{E_{\parallel}})$ 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 $s_{E_{\parallel},0} = (s_{E_1,0}, \dots, s_{E_n,0})$ である。違反状態集合 $S_{E_{\parallel}}^I$ 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 $\{(s_1, \dots, s_n) \in S_{E_{\parallel}} \mid \exists i. s_i \in S_{E_i}^I\}$ として定義される。Marked 状態集合は $S_{E_{\parallel}}^M = \prod_{i=1}^n S_{E_i}^M$ 、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 $A_{E_{\parallel}}^M = \bigcup_{i=1}^n A_{E_i}^M$ 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 $E = (E_1, \dots, E_n)$ 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 $\mathcal{E} = \{E_1, \dots, E_n\}$ 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 $E_M = (S_{E_M}, s_{E_M,0}, S_{E_M}^I, S_{E_M}^M, A_{E_M}, A_{E_M}^M, \Delta_{E_M})$ を以下のように定義する：

- $S_{E_M} = \{0, 1\}$, $s_{E_M,0} = 0$, $S_{E_M}^I = \emptyset$, $S_{E_M}^M = \{1\}$
- $A_{E_M} = A$ (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 $E_{\parallel} = E_1 \parallel \dots \parallel E_n \parallel E_M$ 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 $E_i \in \{E_1, \dots, E_n\}$ について、その Marked 状態集合を $S_{E_i}^M = S_{E_i}$ (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_{\parallel} 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」が見つければ, 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 $E = (E_1, \dots, E_n)$, 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

```
1: function DIRECTEDCONTROLLERSYNTHESIS( $E, H$ )
2:    $ES \leftarrow$  初期状態  $s_0$  のみを持つ部分探索グラフ
3:    $Goals \leftarrow \emptyset, Errors \leftarrow \emptyset, None \leftarrow \{s_0\}$ 

4:   ▷ 探索ループ：初期状態が未分類状態でなくなるまで
5:   while  $s_0 \notin (Goals \cup Errors)$  do
6:      $(s, a, s') \leftarrow$  EXPANDNEXT( $ES, H$ )          ▷ ヒューリスティックによる次遷移の選択
7:      $ES \leftarrow ES \cup \{(s, a, s')\}$               ▷ 探索グラフの更新

8:     if  $s' \in Errors$  then
9:        $Errors \leftarrow Errors \cup \{s'\}$ 
10:      PROPAGATEERROR( $s'$ )                             ▷ 敗北状態の伝播
11:    else if  $s' \in Goals$  then
12:       $Goals \leftarrow Goals \cup \{s'\}$ 
13:      PROPAGATEGOAL( $s'$ )                               ▷ 勝利状態の伝播
14:    else if  $s'$  が新しいループを形成する then
15:       $Loop \leftarrow$  GETLOOP( $s, s'$ )
16:      if  $Loop$  が勝利条件を満たす then
17:         $newGoals \leftarrow$  FINDNEWGOALS( $Loop$ )
18:         $Goals \leftarrow Goals \cup newGoals$ 
19:        PROPAGATEGOAL( $newGoals$ )
20:      else
21:         $newErrors \leftarrow$  FINDNEWERRORS( $Loop$ )
22:         $Errors \leftarrow Errors \cup newErrors$ 
23:        PROPAGATEERROR( $newErrors$ )

24:   if  $s_0 \in Goals$  then
25:      $C \leftarrow$  EXTRACTCONTROLLER( $ES, Goals$ )
26:     return  $C$ 
27:   else
28:     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 $s = (s_{E_1}, s_{E_2}, \dots, s_{E_n})$ 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 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 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 $\mathbf{E} = (E_1, E_2, \dots, E_n)$,

探索到達済みの Marked 状態の集合 $\mathbf{S}^M \subseteq S_{E_1}^M \times S_{E_2}^M \times \dots \times S_{E_n}^M$,

各 LTS における状態の組 $\mathbf{s} = (s_{E_1}, s_{E_2}, \dots, s_{E_n})$,

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 $\mathbf{d} = (d_1, d_2, \dots, d_n)$

ただし, $d_i \in \{(0, d), (1, d), (2, \infty) \mid d \in \mathbb{N}\}$

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( $\mathbf{E}, \mathbf{S}^M, \mathbf{s}, \hat{a}$ )
2:   for  $i = 1, 2, \dots, n$  do
3:      $minSteps \leftarrow \infty$ 

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  $(s_{E_1}^*, s_{E_2}^*, \dots, s_{E_n}^*) \in \mathbf{S}^M$  do
6:        $steps \leftarrow \text{ESTIMATEDIST}(\mathbf{s}, \hat{a}, s_{E_i}^*)$ 
7:       if  $steps < minSteps$  then  $minSteps \leftarrow steps$ 
8:       if  $minSteps \neq \infty$  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:          $d_{E_i} \leftarrow (0, minSteps)$ 
10:      continue

11:     ▷ Rank 1: 任意の Marked 状態への到達可能性確認 ◁
12:     for all  $s_{E_i}^* \in S_{E_i}^M$  do
13:        $steps \leftarrow \text{ESTIMATEDIST}(\mathbf{s}, \hat{a}, s_{E_i}^*)$ 
14:       if  $steps < minSteps$  then  $minSteps \leftarrow steps$ 
15:       if  $minSteps \neq \infty$  then ▷ いずれかの Marked 状態に到達可能な場合
16:          $d_{E_i} \leftarrow (1, minSteps)$ 
17:       else
18:          $d_{E_i} \leftarrow (2, \infty)$ 

19:   return SORTDESCENDING( $(d_{E_1}, d_{E_2}, \dots, d_{E_n})$ ) ▷ 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 \mathbf{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。入力には現在の合成状態 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 $\text{CALCULATEDIST}(s'_{E_i}, s_{E_i}^*)$ 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 $s = (s_{E_1}, \dots, s_{E_n})$,

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 $d \in \mathbb{N} \cup \{\infty\}$

```

1: function ESTIMATEDIST( $s, \hat{a}, s_{E_i}^*$ )
2:   if  $\hat{a} \in A_{E_i}$  then                                     ▷  $\hat{a}$  が LTS  $E_i$  に存在している場合
3:      $s'_{E_i} \leftarrow s'$  where  $(s_{E_i}, \hat{a}, s') \in \Delta_{E_i}$       ▷  $\hat{a}$  を発火した後の状態
4:     if  $s'_{E_i} = s_{E_i}^*$  then                                   ▷ 対象の状態に到達する場合
5:       return 1
6:     else if  $s'_{E_i} \in S_{E_i}^I$  then                             ▷ 違反状態に到達する場合
7:       return  $\infty$                                              ▷ 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
8:     else if  $s'_{E_i} \neq s_{E_i}$  then                             ▷ 他の状態に遷移する場合
9:       return  $\text{CALCULATEDIST}(s'_{E_i}, s_{E_i}^*) + 1$ 
10:    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
11:      return 1

12:   ▷ 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 ◁
13:   for  $j = 1, 2, \dots, n$  do
14:      $A_{E_j}^R \leftarrow \{a \mid \exists s', (s_{E_j}, a, s') \in \Delta_{E_j}, s' \neq s_{E_j}\}$ 
15:    $A^R \leftarrow \bigcup_{j=1}^n A_{E_j}^R$                                ▷ Ready アクションの集合

16:   ▷ Gap を加味した最短経路探索 ◁
17:    $d \leftarrow \min_{a \in A^R} \{\text{CALCULATEGAP}(s, \hat{a}, a) + \text{ESTIMATEDIST}(s, a, s_{E_i}^*)\}$ 
18:   ▷ 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 ◁
19:   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
20:      $d \leftarrow \text{CALCULATEDIST}(s_{E_i}, s_{E_i}^*) + 1$ 
21:   return  $d$ 

```

ここで用いられる $\text{CALCULATEGAP}(\hat{a}, a)$ 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たための遷移 (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

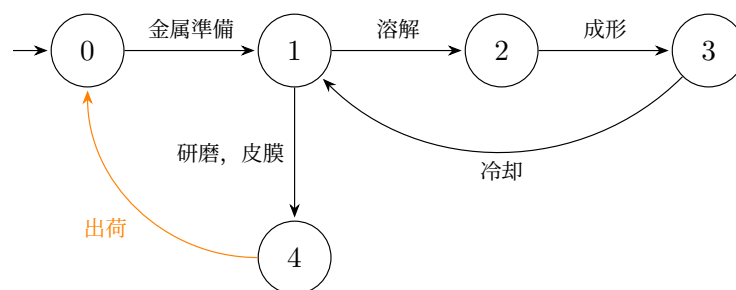
RA が不適切な誘導を行う具体的なシナリオとして、図 3.1 に示す金属加工システムのモデルを用いて説明する。

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}

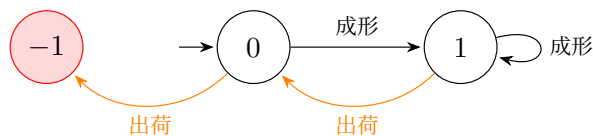


図 3.1 金属加工システムの環境モデルと要求モデル

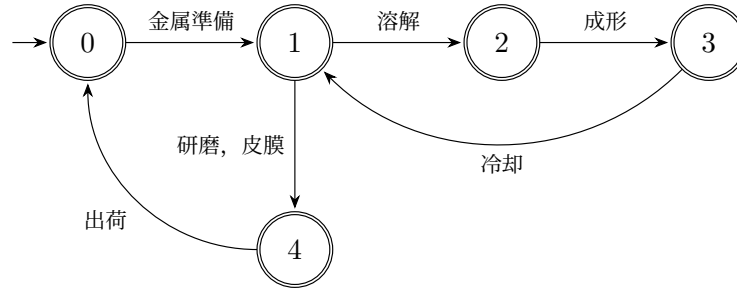
Ready Abstraction 適用のための Marked 状態への変換

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。

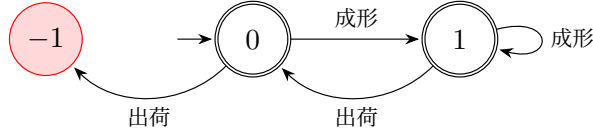
図 3.2 は、変換後のシステム構成を示している。ここでは、新たなコンポーネントとして Marking 判定モデル E_M を導入している。 E_M は、「出荷」アクションが発火された直後のみ Marked 状態 (状態 1) に遷移し、それ以外の場合は非 Marked 状態 (状態 0) に留まる機能を持つ。いわば、アクションの発火イベントを状態遷移として表現するためのモデルである。

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

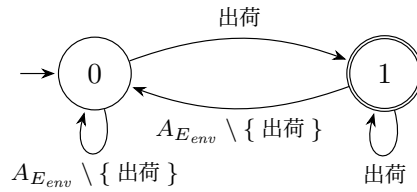


図 3.2 アクション目標を状態目標へ変換した金属加工システム

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 $s_0 = (0, 0, 0)$ （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 $s_1 = (1, 0, 0)$ 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 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， $A^R = \{ \text{溶解, 研磨, 皮膜, 成形, 出荷} \}$ である．RA はこれらの候補について，目標（Marked 状態）までの距離を推定する．

E_{env} および E_{req} の評価（差がつかない）

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状態であるとみなされる．したがって，これら 2つ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．

E_M および Ready アクションを用いた評価（差が生じる）

E_M が状態 1（Marked）へ遷移するためには，「出荷」アクションの発火が不可欠である．RA は，現在の Ready アクション候補から「出荷」を発火可能にするまでの近さ（Gap）を

評価する。

研磨・皮膜 E_{env} において、これらのアクションを実行すると状態 4 へ遷移し、直ちに「出荷」が発火可能となる。したがって、「出荷」への Gap は共に 1 と評価される。

溶解 E_{env} において、これを実行すると状態 2 へ遷移するが、その後「成形」「冷却」を経なければ「出荷」が可能な状態に戻らない。したがって、「出荷」への Gap は 3 と評価される。

この結果、RA は「出荷」への到達コストが低い「研磨」および「皮膜」の 2 つを、同等に有望なアクションであると判断する。これらは「溶解」よりも優先度が高いため、探索アルゴリズムはまずこの 2 つのアクションのいずれか（例えば「研磨」）を選択し、状態 $s_2 = (4, 0, 0)$ 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 $s_2 = (4, 0, 0)$ 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 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、残る候補である「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、もう一方の候補である「皮膜」の探索へ移行する。「皮膜」ルートも同様に同期待ち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、ようやく次善の候補であった「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。

$$d \leftarrow \min_{a \in A^R} \{ \text{CALCULATEGAP}(s, \hat{a}, a) + \text{ESTIMATEDIST}(s, a, s_{E_i}^*) \}$$

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

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

謝辭

参考文献