「マルチメディア，分散，協調とモバイル
(DICOMO2019)シンポジウム」 令和元年7月

環境変化の差分情報を用いた 離散制御器の実行時差分合成アルゴリズム

平野 貴規[1] 相澤 和也[1] 鄭 顕志[1,2] 鷲崎 弘宜[1] 本位田 真一[1,2]

Differential Controller Synthesis at Runtime based on Difference Information of Changed Environment

TAKANORI HIRANO[1] KAZUYA AIZAWA[1] KENJI TEI[1,2] HIRONORI WASHIZAKI[1]
SHINICHI HONIDEN[1,2]

1. はじめに

システム実行時に検知した環境変化に応じて，満たすべき要求の集合を満たすように，システムが自身の振る舞いを変更する自己適応システムの研究が行われてきた．自己適応システムは実行時に観測された環境変化に対して，変化後の環境において要求の集合を満たすことができる動作仕様を高速に決定することが求められる．動作仕様を自動生成する技術として離散制御器合成技術が存在する．実行時に離散制御器合成技術を用いて動作仕様を自動生成するには計算時間オーバーヘッドが課題となる．本論文では，実行時に環境変化の差分情報を用いて動作仕様の差分更新を行うことで離散制御器合成技術を高速化するアルゴリズムを提案する．自動倉庫管理システムの事例をもとに提案したアルゴリズムによる離散制御器の合成時間の削減効果を評価し，提案手法は離散制御器合成による動作仕様の生成に比べて最大で 6.25%に，最小で 48.15%に，中央値で 11.11%にまで計算時間を削減することができた．この結果より，離散制御器の合成時間の短縮が確認できた．

2. 背景技術

本論文で提案するアルゴリズムは離散制御器合成技術 [2] を背景としている．離散制御器合成技術は与えられた環境下で与えられた要求の集合を満たす制御器を自動で合成する技術である．環境はシステムとその外部環境との相互作用を Labeled Transition System(LTS) [6] によって離散的にモデル化されたものである．要求は，環境モデル上で充足すべき性質を Fluent Linear Temporal Logic(FLTL) [4] によって記述したものである．FLTL によって離散制御器合成は安全性と活性を扱うことができる．安全性はシステムが動作する上で常に成立すべき性質を表し，活性はシステムが動作する上でいつかは必ず成立すべき性質を表す．

本論文では提案する手法の性質上，安全性に要求を限定しているため，これ以降では安全性を扱う．

離散制御器合成はこれらの環境モデルと要求の論理式を入力として 2 人対戦型ゲームを用いて行われる [2]．入力された環境モデルと要求の論理式から分析対象となるゲームを構築する．ゲームは有限の状態集合と遷移関係，勝利条件からなる．状態と遷移関係が環境モデルに，勝利条件が要求の論理式と対応している．ここで，ゲームのプレイヤーは一方が制御器であり，もう一方が環境である．

このように生成されるゲームにおいては環境側，制御器側のそれぞれが常に自身の勝利条件を満たせるような戦略と領域が存在する [5]．離散制御器合成では制御器の勝利戦略を特定する事で勝利戦略から制御器の LTS モデルを構築することができる．

システム実行時に環境変化が起こった場合，システムのアクションに対して，環境から期待と異なる応答が返される．したがって本論文では環境モデルに対してそのような遷移や状態が追加されるような環境の変化に対して焦点を当てる．

離散制御器合成で構築されるゲームの規模は入力される環境モデルの状態数と要求の要素数に対して指数関数的に増大する．システム実行時に離散制御器合成を用いる場合，ゲーム空間全体の再構築，分析，勝利領域の特定，勝利戦略の抽出が，環境が変化するたびに行われる．従って複雑なシステムの制御器を実行時に再合成する場合には膨

[1] 早稲田大学
Waseda University
[2] 国立情報学研究所
National Institute of Informatics

大な時間を要する．相澤らの研究 [8] によれば，製品加工工場におけるロボットシステムの制御器の合成において，入力として状態数 6,944 の環境モデルを与えた場合，制御器合成に 20 分かかることが述べられている．

3. 実行時離散制御器合成手法

本章では環境変化の差分情報を用いてシステム実行時の離散制御器の合成時間を短縮するアルゴリズムを説明する．

環境の変化は局所的である．そこで、環境変化時に環境の差分情報と要求からゲームの更新および分析を行い，ゲームの更新に伴い更新された勝利戦略の差分情報を用いて制御器の差分更新を行う手法を提案する．これにより，環境の変化に影響を受けない部分の再分析を省き，計算時間を短縮することを目的とする．今回提案する手法では，相澤らの研究 [1] をもとにゲームの差分更新を行なっているため，要求は安全性のみを扱うことに限定している．そして我々は，自己適応システムにおいて制御器モデルを安全に切り替えるために，実行中の制御器モデルを合成された制御器モデルがシミュレートする [3] ことを条件としている．シミュレートすることによって，制御器の切り替えを行なっても安全性は保証され続ける．

図 1 では本論文で提案するアルゴリズムにおける環境変化の検知から制御器の差分更新までの過程を示している．提案するアルゴリズムはシステム設計時と実行時の 2 つに分かれている．

設計時には環境モデル (E) と安全性の集合の候補 $(G_n, G_{n-1}, \ldots, G_1)$ を入力とする．環境モデルは開発者の想定通りにシステムが動作するという仮定の下に用意したものである．安全性の集合の候補は，システムが充足すべき安全性の要素の集合で定義され，優先度付けがなされている．安全性の集合の候補は優先度の高いものから順に $\mathrm{n}, \mathrm{n}-1, \ldots, 1$ と異なる要求レベルを割り当てる．環境モデルと安全性の集合の候補からゲームの集合 $(Game_n, Game_{n-1}, \ldots, Game_1)$ を構築する．ゲームの集合は入力となった安全性の集合の候補のレベルに対応したレベルが割り当てられる．それぞれのゲームから制御器側の勝利戦略 $(WS_n, WS_{n-1}, \ldots, WS_1)$ が抽出される．それぞれのゲームにおいて抽出されている勝利戦略から制御器モデルを構築し，制御器モデルの集合 $(C_n, C_{n-1}, \ldots, C_1)$ を作成する．制御器モデルも同様にレベルが割り当てられている．また，構築する制御器モデルの集合は集合の中で最上位のレベルの制御器モデル (図 1 の設計時においては C_n) をそれより下のレベルの制御器モデルのいずれかがシミュレートするような関係を持たせるとする．システムは環境モデル E のもとで安全性の集合 G_n を充足するようなコントローラ C_n で動作を開始する．

実行時には環境変化を反映したモデル $(E + \Delta E)$ とアルゴリズムが持っているゲームの集合 $(Game_n, Game_{n-1}, \ldots, Game_1)$，制御器側の勝利戦略の集合 $(WS_n, WS_{n-1}, \ldots, WS_1)$ を入力とする．ここで，ΔE は環境変化による差分情報を表す．環境変化を反映したモデルは環境の学習手法 [7] によって実行時に取得する．環境変化を反映したモデルから各々のゲームに対して差分更新 $(Game + \Delta Game)$ 及び分析を行う [1]．ここで，$\Delta Game$ は環境変化によるゲームの差分を表す．ゲームの更新により，それぞれの制御器の勝利領域と勝利戦略 $(WS + \Delta WS)$ が更新される．ここで，ΔWS は更新された勝利戦略の差分を表す．ゲームを分析している際に合成される制御器モデルが実行中の制御器モデルをシミュレートしているかを確認する．シミュレートしているなら制御器モデルの生成に進み，していないならば生成は行わない．それぞれのゲームの勝利戦略の差分情報 ΔWS から制御器モデルの差分情報 ΔC を構築し，環境が変化する前の制御器モデルと合成し，制御器モデルの差分更新 $(C + \Delta C)$ を行う．更新したゲームの集合とコントローラの集合はシステムが保持する．システム実行時には環境の変化が起こるたびにこれを行う．このようにして，環境変化によって影響を受ける部分にのみ着目することで制御器の合成時間を省く．

4. 評価

本章では提案手法の有用性を，制御器の再合成にかかる計算時間を測ることによって評価する．本論文の研究課題を以下に示す．

研究課題　提案手法はどのくらい合成時間を短縮できたか．

評価時に利用した PC の計算機の CPU は 2.8GHz クアッドコア Intel Core i7，メモリは 16GB，OS は Mac OS High Sierra である．評価に用いたアルゴリズムについては Java で実装している．

4.1 評価実験の設定と結果

研究課題を評価するにあたって，自動倉庫管理システムをケーススタディとして用いる．自動倉庫管理システムの環境モデルの状態数は 19，全安全性の論理式の数は 8，またそれらから生成されるゲームの状態数は 88，最終的に出力される制御器モデルの状態数は 15 である．この数値はシステムが動作する上で理想的な環境モデルのものである．

また，自動倉庫管理システムにおいて提案手法でシステム設計時に用意する，開発者が充足してほしい安全性を 5 段階に分けて用意し，制御器を 5 つ階層状に用意するものとする．

環境の変化が起きた際の切り替え可能な新しい制御器が合成できるまでの時間として，制御器の合成時間を計測した．提案手法における制御器の合成時間は階層制御を行い実行中の制御器から切り替え可能な制御器を合成するまでの時間である．離散制御器合成における制御器の合成時間は一から制御器の合成を行い実行中の制御器から切り替え

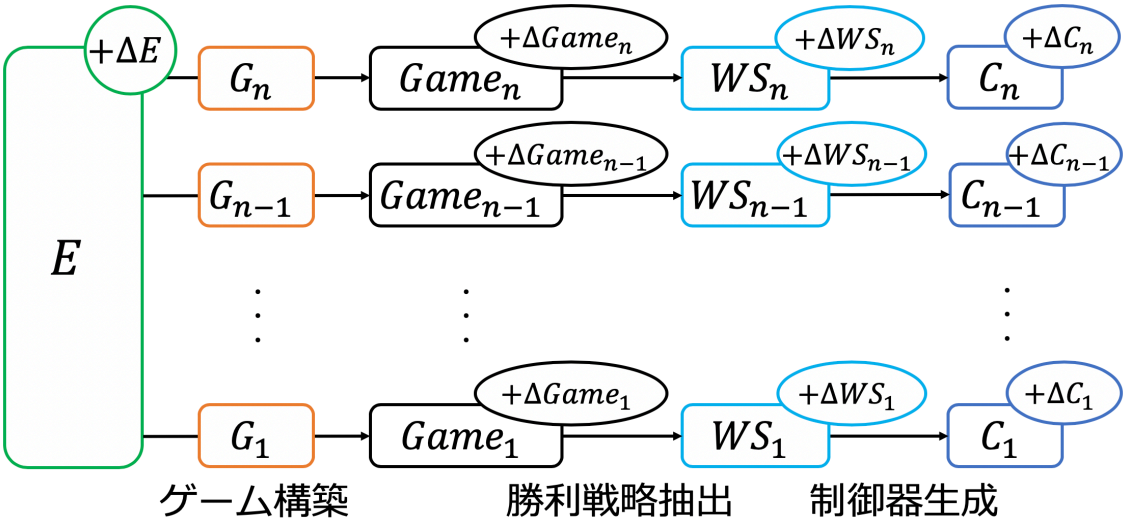

図 1 提案アルゴリズムにおける制御器の差分更新の過程

可能な新しい制御器が合成できるまでの時間である．

評価実験はシステムのアクションに対して，環境から期待と異なる応答が返されるような 15 ケースの環境変化を想定した．それぞれの環境の変化が起きた際の切り替え可能な新しい制御器が合成できるまでの時間を計測した．15 ケースの環境変化のシナリオのうち 15 ケースで合成時間が短縮された．また、提案手法は離散制御器合成による制御器の合成に比べて最大で 6.25% に，最小で 48.15% に，中央値で 11.11% にまで計算時間を削減することができた．

5. 議論

評価実験の結果として合成時間の短縮が見られた．離散制御器合成による一つの制御器の合成時間の時間計算量は分析するゲームの状態数を N，遷移数を M とすると $O(N+M)$ である [5]．提案手法による一つの制御器の合成時間も同様に $O(N+M)$ である．切り替え可能な制御器を合成するのに入力として与えた階層の数を K とすると離散制御器合成と提案手法のどちらも最悪時間の計算量は $O(K(N+M))$ となる．離散制御器合成と提案手法の最悪時間の計算量は同じであるが，提案手法では環境変化の差分情報のみを分析することからゲームの状態数と遷移数が小さくなっているので，提案手法の方が制御器の合成時間は早くなる．

6. おわりに

本論文では環境変化時に環境変化の差分情報を用いてシステム実行時の離散制御器の合成時間を短縮するアルゴリズムを提案した．今回の評価実験に用いたケーススタディでは，従来の離散制御器合成による制御器の合成よりも提案手法による制御器の合成の方が合成時間が短くなったことが確認できた．

将来研究としては次の 2 つをあげる．1 つは本論文における評価実験に用いたケーススタディでは分析する対象となるゲームの規模が小さかったため，分析対象となるゲームの規模が大きなケーススタディで評価実験を行うことである．2 つ目は扱う要求を活性にまで広げることである．

参考文献

[1] Kazuya Aizawa, Kenji Tei, and Shinichi Honiden. Identifying safety properties guaranteed in changed environment at runtime. In *2018 IEEE International Conference on Agents (ICA)*, pp. 75–80. IEEE, 2018.

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

[3] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering*, pp. 688–699. ACM, 2014.

[4] Dimitra Giannakopoulou and Jeff Magee. Fluent model checking for event-based systems. In *ACM SIGSOFT Software Engineering Notes*, Vol. 28, pp. 257–266. ACM, 2003.

[5] Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, Vol. 2500. Springer Science & Business Media, 2002.

[6] Jeff Magee and Jeff Kramer. *State models and java programs*. wiley New York, 1999.

[7] Moeka Tanabe, Kenji Tei, Yoshiaki Fukazawa, and Shinichi Honiden. Learning environment model at runtime for self-adaptive systems. In *Proceedings of the Symposium on Applied Computing*, pp. 1198–1204. ACM, 2017.

[8] 相澤和也, 鄭顕志, 本位田真一ほか. 環境変化時に保証可能な安全性を特定するためのゲーム分析アルゴリズム. 情報処理学会論文誌, Vol. 60, No. 4, pp. 1025–1039, 2019.