Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

モデルベース並列化アルゴリズムの形式化と正当性の証明

多門 俊哉[1,a)]　磯部 祥尚[2]　枝廣 正人[1]

概要：制御システムの大規模化・複雑化が進む中でシステムの並列化手法の研究が盛んになっている．しかし並列化によって，逐次モデルでは起こらなかった実行順序逆転などの並列処理特有の問題が起こる可能性がある．そこで本論文では，筆者らが提案する並列化アルゴリズムを，仕様記述言語 CSP を用いて形式化することによって並列化フローを明確にし，特定の条件下において並列化による実行順序逆転が起こらないことを，CSP の遷移規則を用いて証明する．

1. はじめに

制御システムの大規模化・複雑化が進む中で，シングルコアでは処理の限界をむかえ，マルチ・メニーコアを利用したシステムの制御が注目を集めている．マルチ・メニーコアで処理を行うことによって，シングルコアよりも高性能な処理を行うことが可能になる．しかし，逐次実行を目的として設計されたシステムをマルチコア上で動作をさせるためには，データの依存性などを十分に理解して分割を行う必要があり，理解が不十分なまま設計を行えば予期せぬバグが混入する可能性がある．

筆者らは MATLAB/Simulink[1] モデルからマルチコア上で動作する並列制御用の実行コードを自動生成するモデルベース並列化システム MBPS を開発している [3][4]．Simulink モデルは処理ブロックと信号線を組み合わせてシステムの動作を表すブロック線図であり，ブロックは制御システムの構成要素や機能，信号線はブロック間の信号の流れを表す．このモデルベース並列化システム MBPS は，Simulink モデルからブロック間の依存関係を抽出し，その依存関係を満たすように逐次制御モデルのコア割当分割とコア間通信の挿入を行い，マルチコア用の並列制御モデルを生成する．しかし，逐次制御モデルを並列化することによって，逐次制御モデルでは起こらなかった実行順序逆転などの並列制御モデル特有の問題が生じる可能性がある．並列制御モデルに対して検証ツールを適用する研究も行われているが，検証ツールとしてモデル検査器を利用した先行研究 [2] では，1 回で検証可能な制御モデルのブロック数は数千個程度という上限があった．また，モデル検査器では制御モデルごとに検証を行う必要があるため，並列化を行うたびに検証コストがかかる問題もある．

そこで，本論文ではモデルベース並列化システム MBPS で使われている並列化アルゴリズムを形式仕様記述言語 CSP[5] で厳密に記述し，並列化アルゴリズムの正当性の証明を行う．ここで，並列化アルゴリズムの正当性とは，並列化によってブロックの実行順序が逆転しないこと，すなわち，並列化後のモデルが元の逐次制御モデルと同じ順番でブロックを実行することである．ただし，結果に影響しない実行順番の逆転は許容することによって，並列性を向上させる．この正当性を証明することによって，逐次制御モデルにおいてバグが起こらなければ並列制御モデルにおいてもバグが起こらないことを保証できる．並列化アルゴリズム自身の正当性を証明することによって，並列化ごとに生成される並列制御モデルの実行順序を検証する必要がなくなり，モデルベース並列化システム MBPS 利用時の検証コストを削減することができる．

以降，2 節でモデルベース並列化システム MBPS の並列化アルゴリズムの概要を説明し，3 節で形式仕様記述言語 CSP の基本を紹介する．4 節で仕様記述言語 CSP による並列化アルゴリズムの形式記述を与え，5 節で並列化アルゴリズムの正当性を証明する．最後に，6 節で非同期型通信の形式記述について補足し，7 節で今後の課題について述べる．

[1] 名古屋大学大学院情報学研究科
[2] 産業技術総合研究所
[a)] tamon@ertl.jp

Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

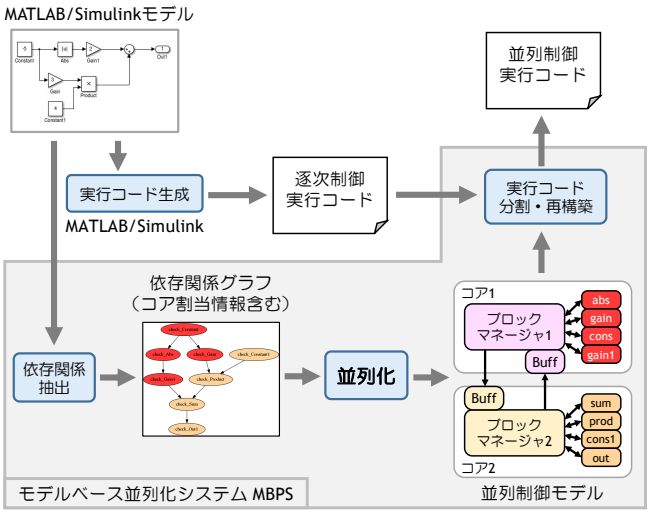

図 1　モデルベース並列化システム MBPS による並列化フロー

2. 並列化アルゴリズムの概要

筆者らは Simulink モデルからマルチコアで動作する並列制御実行コードを自動生成するモデルベース並列化システム MBPS を研究開発中である [3][4]．このモデルベース並列化システム MBPS による並列化フローを図 1 に示す．このモデルベース並列化システム MBPS では，最初に Simulink モデルからブロック間の実行順序を表す依存関係グラフ（各コアに割り当てるブロックの情報を含む）を生成する．図 1 の “**並列化**” のアルゴリズムはこの依存関係グラフを受け取り，**並列制御モデル**を生成する．並列制御モデルは，コアごとに実行する**ブロック**とその情報を伝える**コア間通信**の実行順序を表現している．最後に，この並列制御モデルにしたがって逐次実行コードを並列化し，マルチコアで実行可能な並列制御コードを生成する．

図 1 の並列化の過程で生成される並列制御モデル（コア数が 2 個の場合）の構造を図 2 に示す．並列制御モデルは，複数のブロックマネージャから構成されており，各ブロックマネージャ $\texttt{BlkMng}(c)$ は，コア c に割り当てられたブロック $\texttt{blk}.n$ の実行と，処理済みのブロック n の情報を送受信するためのコア c, c' 間の通信 $\texttt{com}.c.c'.n$ の実行を管理する．なお，モデルベース並列化システム MBPS のコア間通信は，データを一時的に保存するバッファをもつ非同期型通信である．

3. 形式仕様記述言語 CSP の紹介

本論文では，並列化アルゴリズムの形式的な記述と検証に仕様記述言語 CSP（Communicating Sequential Processes）を用いる．CSP は C.A.R.Hoare によって考案されたプロセス代数であり [5]，並行システムの振舞いを形式的に記述し，解析するための理論である．本節では文献 [6] にしたがって，本論文で必要な CSP の構文論と意味論の一部を説明する．

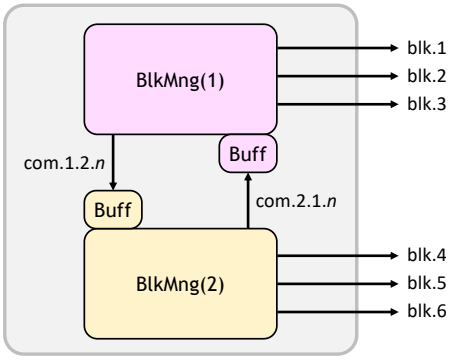

図 2　並列制御モデルの構造

3.1 構文論

CSP における動作の最小単位は**イベント**であり，本論文ではイベントの集合を $\texttt{Events}$ で表し，その要素を $a, b, \ldots$ で表す．例えば，2 節の図 2 の並列制御モデルでは，ブロック実行 $\texttt{blk}.n$ やコア間通信 $\texttt{com}.c.c'.n$ がイベントである（$\texttt{com}.c.c'.n$ はチャネル $\texttt{com}.c.c'$ でデータ n を送受信するためのイベント）．また，CSP では繰り返し動作を表現するために**プロセス名**を用いており，本論文ではプロセス名の集合を $\texttt{PName}$ で表し，その要素を $A, B, \ldots$ で表す．このとき，CSP の**プロセス**の集合を次のように定義する．

定義 3.1　CSP のプロセスの集合 $\texttt{Proc}$（要素を $P, Q, \cdots$ で表す）は次の式を含む最小の集合である[*1]．

$$
\begin{array}{rcll}
A & : & \text{プロセス名} & (A \in \texttt{PName}) \\
\texttt{STOP} & : & \text{停止} & \\
a \to P & : & \text{プレフィクス} & (a \in \texttt{Events}) \\
P \,\square\, Q & : & \text{外部選択} & \\
P \,[\![\, X \,]\!]\, Q & : & \text{並行合成} & (X \subseteq \texttt{Events}) \\
P \backslash X & : & \text{隠蔽} & (X \subseteq \texttt{Events}) \\
b \,\&\, P & : & \text{ガード} & (b \in \texttt{Bool})
\end{array}
$$

ここで，P, Q はすでに $\texttt{Proc}$ の要素であるとする．また，$\texttt{Bool}$ は論理式の集合である．

定義 3.1 の各プロセスの意味は 3.2 小節で定義するが，ここで簡単に説明しておく．停止はこれ以上何も実行しないプロセスである．プレフィクスはイベント a を実行でき，a を実行後はプロセス P のように動作するプロセスである．外部選択はプロセス P または Q のように動作することを，イベントによって外部から選択できるプロセスである．並行合成は P と Q が X に含まれるイベントで同期しながら並行に動作するプロセスである．隠蔽は X に含まれるイベントが内部で実行される以外 P のように動作するプロセスである．ガードは条件 b が真のときに P のように動作するプロセスである．

さらに，3 つ以上の外部選択を簡単に記述するために次

[*1] 内部選択や SKIP 等，本論文で使用しない構文は省略している．

Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

の略記法も用いる．

$$\Box\ x : X@P(x) = \begin{cases} \texttt{STOP} & X = \{\} \\ P(1)\ \Box\ ...\ \Box\ P(n) & X = \{1, ..., n\} \end{cases}$$

また，チャネル ch へ値 v を送信するイベント $ch!v$ と，チャネル ch から受信した値を変数 x に代入するイベント $ch?x$ も，それぞれ次の略記法により与えられる．

$$ch!v \to P = ch.v \to P$$
$$ch?x \to P = \ \Box\ x : \{v \mid ch.v \in \texttt{Events}\}@(ch.x \to P)$$

CSP の通信方式は同期型であり，この送信と受信のイベントは共に実行可能なときに限り同期できる．CSP で非同期型通信を表現するには通信路にバッファを挿入すればよい．

3.2 意味論

CSP のプロセスは観測可能なイベント ($a \in \texttt{Events}$) の他に**内部イベント** (τ で表す) も実行する．内部イベント τ は観測も制御もできないシステム内部で自動的に実行される特殊なイベントである．CSP のプロセスの意味を定義するために，イベントの集合 $\texttt{Events}$ に内部イベント τ を追加した集合を $\texttt{Events}_\tau$ で表し，その要素を $\alpha, \beta, \ldots$ で表す．

$$\texttt{Events}_\tau = \texttt{Events} \cup \{\tau\}$$

このとき，CSP のプロセスの意味はラベル付遷移システム $(\texttt{Proc}, \texttt{Events}_\tau, \{\xrightarrow{\alpha} | \ \alpha \in \texttt{Events}_\tau\})$ をもとに与えられる．ここで，$\xrightarrow{\alpha}$ は定義 3.2 で定義される遷移関係であり，$(P, P') \in \xrightarrow{\alpha}$（以降，$P \xrightarrow{\alpha} P'$ と書く）は，プロセス P はイベント α を実行可能であり，その実行後はプロセス P' のように振る舞うことを意味している．

定義 3.2 プロセス間の遷移関係 $\xrightarrow{\alpha} \subseteq \texttt{Proc} \times \texttt{Proc}$ は図 3 の推論規則（以降，**遷移規則**と呼ぶ）を満たす最小の関係である．ここで，各遷移規則は，横棒の上が 0 個以上の仮定，右横が条件，下が結果を表している．

3.3 トレース詳細化

本論文では，並列化アルゴリズムによって生成される並列制御モデルが，図 1 の依存関係グラフに反しないことを証明する．具体的には，並列制御モデルが実行するイベントの列が，許可されたイベント列であることを示す．本小節では，そのために必要なトレース詳細化という関係を定義する．

まず，イベントの列を順番に実行するための連続した遷移関係を次のように定義する．

定義 3.3 イベント列 $t = \alpha_1 \cdots \alpha_n \in \texttt{Events}_\tau^*$ に対する遷移関係 $\xrightarrow{t}$ を次のように定義する．

$$\textbf{Prefix}\ \frac{}{(a \to P) \xrightarrow{a} P}$$

$$\textbf{ExtCh}_1\ \frac{P \xrightarrow{a} P'}{P\ \Box\ Q \xrightarrow{a} P'}$$

$$\textbf{ExtCh}_2\ \frac{Q \xrightarrow{a} Q'}{P\ \Box\ Q \xrightarrow{a} Q'}$$

$$\textbf{Para}_1\ \frac{P \xrightarrow{\alpha} P'}{P\ [|\ X\ |]\ Q \xrightarrow{\alpha} P'\ [|\ X\ |]\ Q}(\alpha \notin X)$$

$$\textbf{Para}_2\ \frac{Q \xrightarrow{\alpha} Q'}{P\ [|\ X\ |]\ Q \xrightarrow{\alpha} P\ [|\ X\ |]\ Q'}(\alpha \notin X)$$

$$\textbf{Para}_3\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P\ [|\ X\ |]\ Q \xrightarrow{a} P'\ [|\ X\ |]\ Q'}(a \in X)$$

$$\textbf{Hide}_1\ \frac{P \xrightarrow{\alpha} P'}{P \backslash X \xrightarrow{\alpha} P' \backslash X}(\alpha \notin X)$$

$$\textbf{Hide}_2\ \frac{P \xrightarrow{a} P'}{P \backslash X \xrightarrow{\tau} P' \backslash X}(a \in X)$$

$$\textbf{Guard}\ \frac{P \xrightarrow{\alpha} P'}{b\ \&\ P \xrightarrow{\alpha} P'}(b = \texttt{True})$$

$$\textbf{PName}\ \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}(A = P)$$

図 3 プロセスの遷移規則（$a \in \texttt{Events}, \alpha \in \texttt{Events}_\tau$）

$$P \xrightarrow{t} P' \iff \exists P_1 \cdots P_{n-1}.\ P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{n-1}} P_{n-1} \xrightarrow{\alpha_n} P'$$

定義 3.3 は，空列 ε（長さ 0 の列）による遷移も含むため，任意の P について，$P \xrightarrow{\varepsilon} P$ が成り立つ．次に，イベントの前後に 0 個以上の内部イベントによる遷移を許す弱い遷移関係を次のように定義する．

定義 3.4 イベント列 $t = \alpha_1 \cdots \alpha_n \in \texttt{Events}_\tau^*$ に対する弱い遷移関係 $\overset{t}{\Longrightarrow}$ を次のように定義する．

$$P \overset{t}{\Longrightarrow} P' \iff \exists P_1 \cdots P_{n-1}.\ P \overset{\alpha_1}{\Longrightarrow} P_1 \overset{\alpha_2}{\Longrightarrow} \cdots \overset{\alpha_{n-1}}{\Longrightarrow} P_{n-1} \overset{\alpha_n}{\Longrightarrow} P'$$

ここで，$\overset{\alpha}{\Longrightarrow}$ は 1 つのイベント（$\alpha \in \texttt{Events}_\tau$）の前後に 0 個以上の内部イベント τ による遷移 $(\xrightarrow{\tau})^*$ を許す，次のように定義される遷移である．

$$P \overset{\alpha}{\Longrightarrow} P' \iff \exists P_1, P_2.\ P(\xrightarrow{\tau})^* P_1 \xrightarrow{\alpha} P_2(\xrightarrow{\tau})^* P'$$

さらに，イベント列 $t \in \texttt{Events}_\tau^*$ から全ての内部イベントを削除してできるイベント列を $\widehat{t} \in \texttt{Events}^*$ と書く．

定義 3.5 イベント列 $t \in \texttt{Events}_\tau^*$ について，$\widehat{t} \in \texttt{Events}^*$ を帰納的に定義する．

$$\widehat{\varepsilon} = \varepsilon, \qquad \widehat{\alpha \cdot t} = \begin{cases} \alpha \cdot \widehat{t} & \text{if } \alpha \neq \tau \\ \widehat{t} & \text{if } \alpha = \tau \end{cases}$$

Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

プロセス P が実行可能な全ての観測可能なイベント列（**トレース**と呼ぶ）の集合 $\mathtt{traces}(P)$ を次のように定義する．

定義 3.6 プロセス $P \in \mathtt{Proc}$ について，トレース集合を次のように定義する．

$$\mathtt{traces}(P) \quad = \quad \{\, \widehat{t} \mid \exists P'.\ P \xrightarrow{t} P' \}$$

プロセス Q のトレース集合がプロセス P のトレース集合に含まれるとき，Q は P の**トレース詳細化**であるといい，$P \sqsubseteq_T Q$ と書く．

定義 3.7 プロセス Q は P のトレース詳細化（$P \sqsubseteq_T Q$）であることを次のように定義する．

$$P \ \sqsubseteq_T \ Q \quad \Longleftrightarrow \quad \mathtt{traces}(P) \ \supseteq \ \mathtt{traces}(Q)$$

本論文では，トレース詳細化を証明するために定義 3.8 の**弱模倣関係** [7] を用いる．次の命題 3.1 に示すように，弱模倣関係の存在によってトレース詳細化を証明できる．

定義 3.8 プロセス上の二項関係 $\mathcal{S} \subseteq \mathtt{Proc} \times \mathtt{Proc}$ が弱模倣関係であるとは，$(P,Q) \in \mathcal{S}$ ならば，任意の $\alpha \in \mathtt{Events}_\tau$ と $P' \in \mathtt{Proc}$ について，$P \xrightarrow{\alpha} P'$ ならば，ある Q' について，$Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q'$ かつ $(P',Q') \in \mathcal{S}$ を満たすことである．

命題 3.1 $(P,Q) \in \mathcal{S}$ となる弱模倣関係 $\mathcal{S}$ が存在するならば，$Q \sqsubseteq_T P$ である．

証明 $t \in \mathtt{traces}(P)$ とする．すなわち，t は τ を含まず，$P \xrightarrow{t} P'$ となる P' が存在する．このとき，$(P,Q) \in \mathcal{S}$ であるので，弱模倣関係の定義 3.8 より，トレース t の長さについて帰納法を適用し，ある Q' について，$Q \stackrel{t}{\Longrightarrow} Q'$ かつ $(P',Q') \in \mathcal{S}$ を得る．すなわち，$t \in \mathtt{traces}(Q)$ を得る．よって，$\mathtt{traces}(Q) \supseteq \mathtt{traces}(P)$（$Q \sqsubseteq_T P$）である． ∎

4. 並列化アルゴリズムの形式記述

本節では，図 1 の並列化アルゴリズムの正当性を証明するために，アルゴリズムの形式記述を与える．図 2 で説明したように，コア間通信はバッファをもつ非同期型通信であるが，本論文では，同期型通信に置き換えて形式化し，正当性を証明する．非同期型通信については 6 節で補足するが，非同期型の並列化アルゴリズムの正当性の証明は今後の課題である．

4.1 並列化アルゴリズムへの入力

図 1 で説明したように，並列化アルゴリズムへの入力は依存関係グラフ（ブロックのコア割当情報含む）である．この入力を次のように，依存関係を表す有向グラフ $(\mathtt{Blks}, \mathtt{Deps})$ とコア割当を表す関数 $\mathtt{Asn}$ で形式化する．

- 依存関係を表す有向グラフ $(\mathtt{Blks}, \mathtt{Deps})$:
  Simulink モデルから抽出した依存関係グラフに含まれるブロック間の依存関係を表す有向グラフであり，$\mathtt{Blks}$ はブロックの集合，$\mathtt{Deps} \subseteq \mathtt{Blks} \times \mathtt{Blks}$ は信号線で接続されたブロックの組の集合である．例えば，$(b_1, b_2) \in \mathtt{Deps}$ は，元の Simulink モデルにおいて，ブロック b_1 から b_2 へのデータ伝搬が存在することを意味する．なお，本論文では，フィードバックのない 1 サイクル分の制御フローを対象とする．すなわち，依存関係を表す有向グラフは閉路をもたない．
- コア割当を表す関数 $\mathtt{Asn} :: \{1,2\} \to \mathtt{Pow}(\mathtt{Blks})$:
  モデルベース並列化システム MBPS は，Simulink モデルの各ブロックの処理を割り当てるコアを計算しており，$\mathtt{Asn}$ はコア ID を受け取り，そのコアに割り当てられるブロックの集合（$\mathtt{Blks}$ の部分集合）を返す関数である．なお，本論文では，コア数は 2 に限定する．すなわち，コア ID は，1 または 2 である．

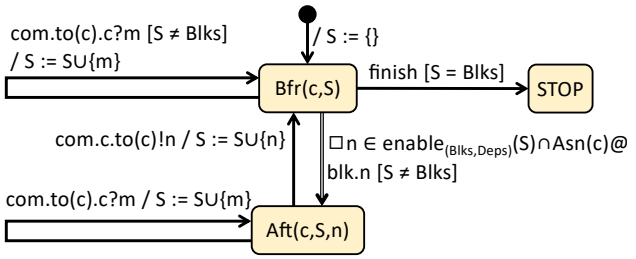

図 4 ブロックマネージャ $\mathtt{BlkMng}(c)$ の状態遷移図

4.2 並列化アルゴリズムからの出力

図 1 で説明したように，並列化アルゴリズムからの出力は図 2 の並列制御モデルである[*2]．本小節ではブロックマネージャ $\mathtt{BlkMng}(c)$ の振舞いについて説明する．

形式化する前に，ブロックマネージャ $\mathtt{BlkMng}(c)$ の振舞いを明確にするために，その振舞いの状態遷移図を図 4 に示す．図 4 の状態 $\mathtt{Bfr}(c,S)$ と $\mathtt{Aft}(c,S,n)$ の S はブロックマネージャの内部変数であり，その時点までに実行済み（遅れはあるが他のコアによる実行済みも含む）のブロックの ID の集合である．また，図 4 の $ev[b]/act$ の形のラベルをもつ状態遷移は，条件 b が真のときにイベント ev を実行でき，実行後にアクション act を実行してから，次の状態に遷移することを表している．ここで，アクションはそのプロセス内部で実行可能な振舞いであり，本論文では内部変数 S の更新に使われている．

図 4 の $\mathtt{STOP}$ は全てのブロックの実行（1 サイクルの処理）が完了した状態である．1 サイクル分の処理が完了した状態と，何らかの不具合で停止した状態を区別するために，完了時にイベント $\mathtt{finish}$ を実行するようにしている．以下，ブロック実行前状態 $\mathtt{Bfr}(c,S)$ とブロック実行後状態 $\mathtt{Aft}(c,S,n)$ について説明する．

4.2.1 ブロック実行前状態 $\mathtt{Bfr}(c,S)$

コア c で次に実行可能なブロックの実行待ちしている状態であり，次のイベントを実行可能である．

[*2] ただし，形式化では簡単のためバッファは省略する．

Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

$$
\begin{array}{l}
\texttt{ParaAlgo}((\texttt{Blks},\texttt{Deps}),\texttt{Asn}) = \textbf{let} \\
\quad \texttt{ParaCtrl} = (\texttt{BlkMng}(1)\ [|\ \texttt{ComFin}\ |]\ \texttt{BlkMng}(2)) \backslash \texttt{Com} \\
\quad \texttt{BlkMng}(c) = \textbf{let} \\
\qquad \texttt{Bfr}(c,S) = \quad (S = \texttt{Blks})\ \&\ \texttt{finish} \rightarrow \texttt{STOP} \\
\qquad\qquad \Box\ (S \neq \texttt{Blks})\ \&\ (\quad (\ \Box\ n : \texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S) \cap \texttt{Asn}(c)\ @\ \texttt{blk}.n \rightarrow \texttt{Aft}(c,S,n)) \\
\qquad\qquad\qquad\qquad \Box\ (\texttt{com.to}(c).c?m \rightarrow \texttt{Bfr}(c, S \cup \{m\}))) \\
\qquad \texttt{Aft}(c,S,n) = (\texttt{com}.c.\texttt{to}(c)!n \rightarrow \texttt{Bfr}(c, S \cup \{n\}))\ \Box\ (\texttt{com.to}(c).c?m \rightarrow \texttt{Aft}(c, S \cup \{m\}, n)) \\
\quad \textbf{within}\ \ \texttt{Bfr}(c,\{\}) \\
\textbf{within}\ \ \texttt{ParaCtrl} \\
\\
\texttt{ComFin} = \{|\texttt{com},\texttt{finish}|\} = \texttt{Com} \cup \{\texttt{finish}\} \\
\quad \texttt{Com} = \{|\texttt{com}|\} = \{e \mid \exists c, c', n.\ \ e = \texttt{com}.c.c'.n \in \texttt{Events}\}
\end{array}
$$

図 5 並列化アルゴリズム ParaAlgo の CSP による形式記述

- blk.n: ブロック n を実行するためのイベントである．実行済みのブロックの ID 集合 S と依存関係の有向グラフ (Blks, Deps) から，次に実行可能なブロックの ID 集合は次の関数 $\texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S)$ によって求められる．

$$
\begin{aligned}
&\texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S) \\
&\quad = \{n \in \texttt{Blks} \mid \texttt{src}_{(\texttt{Blks},\texttt{Deps})}(n) \subseteq S, n \notin S\} \\
&\texttt{src}_{(\texttt{Blks},\texttt{Deps})}(n) \\
&\quad = \{m \in \texttt{Blks} \mid (m,n) \in \texttt{Deps}\}
\end{aligned}
$$

  このとき，コア c で実行可能なブロックの ID 集合は，コア c に割り当てられたブロックの ID 集合 $\texttt{Asn}(c)$ に制限した集合（$\texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S) \cap \texttt{Asn}(c)$）である．図 4 では，イベント blk.$n$ による状態 $\texttt{Bfr}(c,S)$ から状態 $\texttt{Aft}(c,S,n)$ への遷移は，この状態で実行可能な（複数の）ブロック $n \in \texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S) \cap \texttt{Asn}(c)$ の実行を表す遷移をひとつにたたみ込んで表現している．実行時，この複数の遷移からひとつを選択してブロック blk.n を実行する．
- com.to(c).c?m: 他のコアから実行済みのブロック ID を受信し，その ID を変数 m に代入するためのイベントである．関数 $\texttt{to}(c)$ はコア c の他のコア ID を求める関数であり，$\texttt{to}(1) = 2, \texttt{to}(2) = 1$ である．このイベントで他のコアから受信したブロック ID は，内部変数 S（実行済みのブロック ID の集合）に追加される．
- finish: 割り当てられた全てのブロックの処理が完了したことを表すイベントである．このイベントは，実行済みのブロック ID の集合 S が全てのブロックの集合 Blks と同じときのみ実行可能である．

4.2.2 ブロック実行後状態 $\texttt{Aft}(c,S,n)$

自コアでブロック n を実行したことを，他のコアに送信待ちしている状態であり，次のイベントを実行可能である．

- com.c.to(c).c!n: ブロック n を実行したことを他のコアに送信するためのイベントである．送信後に，n を内部変数 S（実行済みのブロック ID の集合）に追加する．
- com.to(c).c?m: 他のコアから実行済みのブロック ID を受信し，その ID を変数 m に代入するためのイベントである（状態 $\texttt{Bfr}(c,S)$ の com.to(c).c?m と同様）．

4.3 並列化アルゴリズムの形式記述

並列化アルゴリズム ParaAlgo の CSP による形式記述を図 5 に示す．図 5 の ParaAlgo は，依存関係を表す有向グラフ (Blks, Deps) とコア割当を表す関数 Asn を受け取り，並列制御モデルの CSP プロセス ParaCtrl を返す関数である．

図 5（2 行目）の ParaCtrl は図 2 の並列制御モデルの構造の CSP 記述であり，2 つのブロックマネージャ $\texttt{BlkMng}(c)$ ($c \in \{1,2\}$) をイベント com.c.c'.n, finish で同期するようにイベント集合 ComFin で並行合成し，イベント集合 Com でイベント com.c.c'.n を外部から隠蔽することを表している（ComFin と Com の定義も図 5 に示す）．図 5 の BlkMng は図 4 のブロックマネージャの振舞いの CSP 記述であり，ブロック実行前状態 $\texttt{Bfr}(c,S)$ とブロック実行後状態 $\texttt{Aft}(c,S,n)$ で実行可能なイベントを，CSP のプレフィクス，外部選択，ガードで形式的に記述している．ここで，enable と to は，各々 4.2.1 小節で定義した実行可能なブロック ID の集合と他のコア ID を計算する関数である．

5. 並列化アルゴリズムの形式検証

本節では，並列化アルゴリズムによって生成される並列制御モデルが依存関係を満たすこと，すなわち，図 5 の CSP プロセス ParaCtrl の任意のトレース（ブロック実行イベントの列）$t \in \texttt{traces}(\texttt{ParaCtrl})$ が有向グラフ (Blks, Deps) の順序制約を満たすことを証明する．そのために，5.1 小節で有向グラフ (Blks, Deps) を満たすトレースのみを実行可能な**逐次制御モデル**の CSP プロセス SeqCtrl を**仕様**として定義し，5.2 小節で並列制御モデルの CSP プロセス ParaCtrl が SeqCtrl のトレース詳細化（$\texttt{SeqCtrl} \sqsubseteq_T \texttt{ParaCtrl}$）であることを証明する．

Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

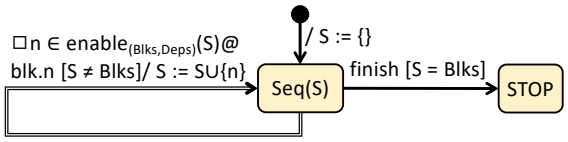

図 6　逐次制御モデル SeqCtrl の状態遷移図

```
Spec(Blks, Deps) = let
  SeqCtrl = let
    Seq(S) =    (S = Blks) & finish → STOP
              □ (S ≠ Blks) & ( □ n : enable(Blks,Deps)(S) @
                                  blk.n → Seq(S ∪ {n}))
  within Seq({})
within SeqCtrl
```

図 7　仕様 Spec の CSP による形式記述

5.1　逐次制御モデル（仕様）

Simulink モデルから生成した依存関係の有向グラフ (Blks, Deps) を満たす順番でブロックを逐次的に実行する逐次制御モデルの状態遷移図を図 6 に示す．ここで，enable は 4.2.1 小節で定義した実行可能なブロック ID の集合である．図 4 のブロックマネージャの状態遷移図と比較して，図 6 の逐次制御モデルは enable 関数にしたがって実行可能なブロックを順番に実行するプロセスであり，同期やデータの送受信もなく，その振舞いは簡潔である．

逐次制御モデルの CSP による形式記述を図 7 に示す．図 7 の形式記述が並列制御モデルの仕様 Spec である．このとき，図 7 の逐次制御モデルの CSP プロセス SeqCtrl について，次の命題 5.1 が成り立つ．この命題は逐次制御モデルがブロック n を実行するならば，その実行前に必要な全てのブロックが実行済みであることを保証している．

命題 5.1　逐次制御モデルの CSP プロセス SeqCtrl のトレースについて次の関係が成り立つ:

$$\begin{aligned}&\forall t \in \mathtt{traces}(\mathtt{SeqCtrl}).\ \forall k < \mathtt{length}(t).\\ &\quad (\exists n.\ t[k] = \mathtt{blk}.n \wedge \mathtt{src}_{(\mathtt{Blks},\mathtt{Deps})}(n) \subseteq \mathtt{done}(t,k))\\ &\qquad \vee\ t[k] = \mathtt{finish}\end{aligned}$$

ここで，$t[k]$ はトレース t の k 番目（先頭は 0 番目）のイベントを表し，$\mathtt{src}_{(\mathtt{Blks},\mathtt{Deps})}(n)$ は 4.2.1 小節で定義したブロック n の前に実行すべきブロック ID の集合，$\mathtt{done}(t,k)$ は次のように定義される k 番目までに実行済みのブロック ID の集合である．

$$\mathtt{done}(t,k) = \{m \mid \exists k' < k.\ t[k'] = \mathtt{blk}.m\}$$

証明 $t \in \mathtt{traces}(\mathtt{SeqCtrl})$，$k < \mathtt{length}(t)$ とする．SeqCtrl の定義より，ある n について $t[k] = \mathtt{blk}.n$ または $t[k] = \mathtt{finish}$ である．SeqCtrl が $\mathtt{blk}.n$ を実行するためには，その直前の状態 $\mathtt{Seq}(S)$ において，$n \in \mathtt{enable}_{(\mathtt{Blks},\mathtt{Deps})}(S)$ でなければならない．すなわち，enable の定義より，$\mathtt{src}_{(\mathtt{Blks},\mathtt{Deps})}(n) \subseteq S$ である．一方，S にはその状態に到達するまでに実行されたブロック ID が保存されているため，$S = \mathtt{done}(t,k)$ である．以上より，$t[k] = \mathtt{finish}$ または，ある n について，$t[k] = \mathtt{blk}.n$，$\mathtt{src}_{(\mathtt{Blks},\mathtt{Deps})}(n) \subseteq \mathtt{done}(t,k)$ である． ∎

5.2　トレース詳細化の証明

5.1 小節では逐次制御モデル SeqCtrl の全てのトレースが依存関係 (Blks, Deps) を満たすことを証明した．本小節では，並列制御モデル ParaCtrl のトレース集合が逐次制御モデル SeqCtrl のトレース集合に含まれること，すなわち，ParaCtrl が SeqCtrl のトレース詳細化であることを証明する．そのためには，命題 3.1 に示したように，$(\mathtt{ParaCtrl}, \mathtt{SeqCtrl}) \in \mathcal{S}$ となる弱模倣関係 $\mathcal{S}$ の存在を示せば十分である．その弱模倣関係の存在を示す補題 5.2 を与える．

補題 5.2　(Blks, Deps) を有向グラフ，Asn は，$\mathtt{Asn}(1) \cap \mathtt{Asn}(2) = \{\}$ を満たす関数とする．このとき，図 8 の関係 $\mathcal{S}$ は弱模倣関係である．

証明 $(P,Q) \in \mathcal{S}$，$P \xrightarrow{\alpha} P'$ とする．$\mathcal{S}$ が弱模倣関係であることを証明するために，$\mathcal{S}$ に含まれる全ての組について，$Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ かつ $(P',Q') \in \mathcal{S}$ を満たす Q' が存在することを示す．本論文では，(P,Q) が図 8 の 2 行目の集合に含まれる場合，すなわち

$$\begin{aligned}&P = (\mathtt{Bfr}(1,S)\ [|\ \mathtt{ComFin}\ |]\ \mathtt{Bfr}(2,S))\backslash \mathtt{Com},\\ &Q = \mathtt{Seq}(S),\\ &S \subseteq \mathtt{Blks}\end{aligned} \tag{1}$$

の場合のみ証明を記す．他の場合についても同様に証明できる．なお，本証明では，プロセス名の遷移規則 **PName** による推論は結論に影響しないので省略する．

(1) 式より，$P \xrightarrow{\alpha} P'$ の遷移導出に最後に適用された遷移規則は $\mathbf{Hide_i}$（$i \in \{1,2\}$）である．すなわち，P の遷移が導出されるためには，ある α' について，次の遷移が必要である．

$$\begin{aligned}&\mathtt{Bfr}(1,S)\ [|\ \mathtt{ComFin}\ |]\ \mathtt{Bfr}(2,S) \xrightarrow{\alpha'} P'',\\ &P' = P''\backslash \mathtt{Com},\ \ \alpha = \begin{cases}\alpha' & \text{if } \alpha' \notin \mathtt{Com}\\ \tau & \text{if } \alpha' \in \mathtt{Com}\end{cases}\end{aligned} \tag{2}$$

(2) 式の遷移導出に最後に適用された遷移規則は $\mathbf{Para_i}$（$i \in \{1,2,3\}$）である．各場合（$i \in \{1,2,3\}$）について証明する必要があるが，ここでは，$\mathbf{Para_1}$ による場合のみ示す（他の場合も同様である）．(2) 式の遷移が規則 $\mathbf{Para_1}$ によって導出されるためには，次の遷移が必要である．

$$\begin{aligned}&\mathtt{Bfr}(1,S) \xrightarrow{\alpha'} P_1',\\ &P'' = P_1'\ [|\ \mathtt{ComFin}\ |]\ \mathtt{Bfr}(2,S),\ \ \alpha' \notin \mathtt{ComFin}\end{aligned} \tag{3}$$

Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

$$
\begin{aligned}
\mathcal{S} = {} & \{(\texttt{ParaCtrl}, \texttt{SeqCtrl})\} \\
& \cup \{((\texttt{Bfr}(1,S)\ [|\ \texttt{ComFin}\ |]\ \texttt{Bfr}(2,S))\backslash \texttt{Com}, \texttt{Seq}(S)) \mid S \subseteq \texttt{Blks}\} \\
& \cup \{((\texttt{Aft}(1,S,n)\ [|\ \texttt{ComFin}\ |]\ \texttt{Bfr}(2,S))\backslash \texttt{Com}, \texttt{Seq}(S')) \mid S' = S \cup \{n\}, S' \subseteq \texttt{Blks}, n \notin S, n \in \texttt{Asn}(1)\} \\
& \cup \{((\texttt{Bfr}(1,S)\ [|\ \texttt{ComFin}\ |]\ \texttt{Aft}(2,S,m))\backslash \texttt{Com}, \texttt{Seq}(S')) \mid S' = S \cup \{m\}, S' \subseteq \texttt{Blks}, m \notin S, m \in \texttt{Asn}(2)\} \\
& \cup \{((\texttt{Aft}(1,S,n)\ [|\ \texttt{ComFin}\ |]\ \texttt{Aft}(2,S,m))\backslash \texttt{Com}, \texttt{Seq}(S')) \mid S' = S \cup \{n,m\}, S' \subseteq \texttt{Blks}, n \notin S, m \notin S, n \in \texttt{Asn}(1), m \in \texttt{Asn}(2)\} \\
& \cup \{((\texttt{STOP}\ [|\ \texttt{ComFin}\ |]\ \texttt{STOP})\backslash \texttt{Com}, \texttt{STOP})\}
\end{aligned}
$$

図 8 弱模倣関係 $\mathcal{S}$

(3) 式の遷移導出に最後に適用された遷移規則は $\mathbf{ExtCh_i}$ ($i \in \{1,2\}$) である．ここでは，より重要な $\mathbf{ExtCh_2}$ による場合のみ示す．すなわち，次の遷移が必要である（$\mathbf{ExtCh_i}$ の遷移のイベントは α ではなく a であることに注意）．

$$(S \neq \texttt{Blks})\ \&\ (P_{\texttt{blk}} \ \Box\ P_{\texttt{com}}) \xrightarrow{\alpha'} P_1',\quad \alpha' \neq \tau \tag{4}$$

ここで，$P_{\texttt{blk}}$ と $P_{\texttt{com}}$ を次のようにおいている．

$$
\begin{aligned}
P_{\texttt{blk}} = {} & \Box\ n : \texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S) \cap \texttt{Asn}(1)\ @ \\
& \texttt{blk}.n \rightarrow \texttt{Aft}(1,S,n) \\
P_{\texttt{com}} = {} & \texttt{com}.2.1?m \rightarrow \texttt{Bfr}(1, S \cup \{m\})
\end{aligned} \tag{5}
$$

(4) 式の遷移導出に最後に適用された遷移規則は **Guard** であるので，次の遷移が必要である．

$$P_{\texttt{blk}} \ \Box\ P_{\texttt{com}} \xrightarrow{\alpha'} P_1',\quad S \neq \texttt{Blks} \tag{6}$$

(6) 式の遷移導出に最後に適用された遷移規則は $\mathbf{ExtCh_1}$ または $\mathbf{ExtCh_2}$ である．以下，各場合に分けて証明する．

- $\mathbf{ExtCh_1}$ によって (6) 式の遷移が導出されたと仮定すると，次の遷移が必要である．

$$P_{\texttt{blk}} \xrightarrow{\alpha'} P_1' \tag{7}$$

(5) 式より，(7) 式の遷移導出に最後に適用された遷移規則は $\mathbf{ExtCh_i}$（$i \in \{1,2\}$）であり，次の条件を満たす n と遷移が必要である．

$$
\begin{aligned}
& (\texttt{blk}.n \rightarrow \texttt{Aft}(1,S,n)) \xrightarrow{\alpha'} P_1', \\
& n \in \texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S) \cap \texttt{Asn}(1)
\end{aligned} \tag{8}
$$

(8) 式の遷移導出に最後に適用された遷移規則は **Prefix** であるので，次の条件が得られる．

$$\alpha' = \texttt{blk}.n,\quad P_1' = \texttt{Aft}(1,S,n) \tag{9}$$

イベント $\texttt{blk}.n$ は導出中に得られた条件（$\alpha' \neq \tau, \alpha' \notin \texttt{ComFin}$）を満たしており，$\alpha' \notin \texttt{Com}$ であるので，(2) 式より，P の遷移（$P \xrightarrow{\alpha} P'$）のイベント α を得る．

$$\alpha = \alpha' = \texttt{blk}.n \tag{10}$$

また，(2) 式，(3) 式，(9) 式より，P の遷移先 P' を得る．

$$
\begin{aligned}
P' &= P''\backslash \texttt{Com} \\
&= (P_1'\ [|\ \texttt{ComFin}\ |]\ \texttt{Bfr}(2,S))\backslash \texttt{Com} \\
&= (\texttt{Aft}(1,S,n)\ [|\ \texttt{ComFin}\ |]\ \texttt{Bfr}(2,S))\backslash \texttt{Com}
\end{aligned} \tag{11}
$$

以下，この P の遷移に対応する Q の遷移を導出する．まず，遷移規則 **Prefix** よって次の遷移を導出できる．

$$(\texttt{blk}.n \rightarrow \texttt{Seq}(S \cup \{n\})) \xrightarrow{\texttt{blk}.n} \texttt{Seq}(S \cup \{n\}) \tag{12}$$

(8) 式より，$n \in \texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S)$ であるので，

$$
\begin{aligned}
Q_{\texttt{blk}} = {} & \Box\ n : \texttt{enable}_{(\texttt{Blks},\texttt{Deps})}(S)\ @ \\
& (\texttt{blk}.n \rightarrow \texttt{Seq}(S \cup \{n\})
\end{aligned} \tag{13}
$$

とおくと，(12) 式の遷移から，遷移規則 $\mathbf{ExtCh_i}$ によって次の遷移を導出できる．

$$Q_{\texttt{blk}} \xrightarrow{\texttt{blk}.n} \texttt{Seq}(S \cup \{n\}) \tag{14}$$

さらに，(6) 式より $S \neq \texttt{Blks}$ であるので，(14) 式の遷移から，遷移規則 **Guard** によって次の遷移を導出できる．

$$(S \neq \texttt{Blks})\ \&\ Q_{\texttt{blk}} \xrightarrow{\texttt{blk}.n} \texttt{Seq}(S \cup \{n\}) \tag{15}$$

最後に，(15) 式の遷移から，遷移規則 $\mathbf{ExtCh_2}$ によって次の遷移を導出できる．

$$\texttt{Seq}(S) \xrightarrow{\texttt{blk}.n} \texttt{Seq}(S \cup \{n\}) \tag{16}$$

ここで，

$$Q' = \texttt{Seq}(S'),\quad S' = S \cup \{n\} \tag{17}$$

とおくと，(1) 式，(10) 式，(16) 式より，次の遷移が得られる．

$$Q \xrightarrow{\alpha} Q' \tag{18}$$

また，(1) 式，(8) 式，(17) 式と関数 enable の定義より，次の条件が成り立つ．

$$S' \subseteq \texttt{Blks},\quad n \notin S,\quad n \in \texttt{Asn}(1) \tag{19}$$

結果として，(11) 式，(17) 式，(19) 式より，次の条件が成り立つ．

Vol.2019-ARC-235 No.9
Vol.2019-SLDM-187 No.9
Vol.2019-EMB-50 No.9
2019/3/17

$$(P', Q') \in \mathcal{S} \quad \text{（図 8 の 3 行目の集合に含まれる）} \tag{20}$$

遷移関係 $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ は遷移関係 $\stackrel{\alpha}{\longrightarrow}$ よりも弱いので，(18) 式と (20) 式より，弱模倣関係のための次の条件が成り立つ．

$$Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q', \ \ (P', Q') \in \mathcal{S} \tag{21}$$

- **ExtCh$_2$** によって (6) 式の遷移が導出されたと仮定すると，次の遷移が必要である．

$$P_{\texttt{com}} \stackrel{\alpha'}{\longrightarrow} P_1' \tag{22}$$

(22) 式の遷移導出に適用された遷移規則は **Prefix** と **ExtCh**（受信は略記法であることに注意）であるので，ある $m \in \texttt{Blks}$ について，次の関係が得られる．

$$\alpha' = \texttt{com}.2.1.m, \ \ P_1' = \texttt{Bfr}(1, S \cup \{m\}) \tag{23}$$

ここで，$\alpha' = \texttt{com}.2.1.m$ は，(3) 式の $\alpha' \notin \texttt{ComFin}$ に矛盾する．すなわち，(6) 式の遷移導出に最後に適用された遷移規則は **ExtCh$_2$** ではない．

他の場合についても同様に弱模倣関係の条件を満たすことを示せる． ∎

最後に，並列制御モデルがブロック n を実行するならば，その前に必要な全てのブロックが実行済みであることを保証する命題 5.3 を与える．

命題 5.3 $(\texttt{Blks}, \texttt{Deps})$ を有向グラフ，$\texttt{Asn}$ は，$\texttt{Asn}(1) \cap \texttt{Asn}(2) = \{\}$ を満たす関数とする．並列制御モデルの CSP プロセス $\texttt{ParaCtrl}$ のトレースについて次の関係が成り立つ:

$$\begin{aligned} &\forall t \in \texttt{traces}(\texttt{ParaCtrl}).\ \forall k < \texttt{length}(t).\\ &\quad (\exists n.\ t[k] = \texttt{blk}.n \wedge \texttt{src}_{(\texttt{Blks},\texttt{Deps})}(n) \subseteq \texttt{done}(t, k))\\ &\qquad \vee\ t[k] = \texttt{finish} \end{aligned}$$

証明 $t \in \texttt{traces}(\texttt{ParaCtrl})$，$k < \texttt{length}(t)$ とする．命題 3.1 と補題 5.2 より，$\texttt{traces}(\texttt{ParaCtrl}) \subseteq \texttt{traces}(\texttt{SeqCtrl})$ である．すなわち，命題 5.1 より，ある n について，$t[k] = \texttt{blk}.n \wedge \texttt{src}_{(\texttt{Blks},\texttt{Deps})}(n) \subseteq \texttt{done}(t, k)$ または $t[k] = \texttt{finish}$ である． ∎

6. 非同期通信への対応

CSP の通信方式は同期型通信であるが，バッファをモデル化して通信路に挿入することによって，非同期型通信を表現することもできる．モデルベース並列化システム MBPS で生成される並列制御コードのコア間通信は，容量 1 のバッファをもつ非同期型通信である．本論文では，モデルベース並列化システム MBPS の並列化アルゴリズムの正当性を証明する最初の試みとして，同期型通信に簡略化して証明した．

図 5 の並列化アルゴリズムで生成する並列制御モデルを非同期型通信に変更するには，次のようにバッファ $\texttt{Buff}$ を挿入すればよい．

$$\begin{aligned} \texttt{ParaCtrl} &= (\texttt{BBlkMng}(1)\ [|\ \texttt{ComFin}\ |]\ \texttt{BBlkMng}(2))\backslash\texttt{Com}\\ \texttt{BBlkMng}(c) &= (\texttt{BlkMng}(c)[[\texttt{com.to}(c).c \leftarrow \texttt{out}]]\\ &\qquad [|\ \{|\texttt{out}|\}\ |]\ \texttt{Buff})\backslash\{|\texttt{out}|\}\\ &\qquad [[\texttt{in} \leftarrow \texttt{com.to}(c).c]]\\ \texttt{Buff} &= \texttt{in}?x \rightarrow \texttt{out}!x \rightarrow \texttt{Buff} \end{aligned}$$

ここで，$P[[a \leftarrow b]]$ は P のイベント a を b に名前変更してできるプロセスである．非同期型通信では，ブロック ID を送信してから受信するまでに遅れがあるため，並列制御モデルの状態数は増加し，弱模倣関係は図 8 よりも複雑になる．非同期型通信の場合の並列化アルゴリズムの証明は今後の課題である．

7. おわりに

本論文では，モデルベース並列化システム MBPS の並列化アルゴリズム（簡略版）を仕様記述言語 CSP によって形式的に記述し，CSP の遷移規則によってその正当性「ブロックの依存関係に反しないこと」を証明した．ただし，今回の並列化アルゴリズムでは次の制約がある．

- 入力の Simulink モデルはフィードバックなしに限る（ブロック間依存関係の有向グラフは閉路をもたない）
- 出力の並列制御モデルのコア間通信は同期型に限る（送信データを一時的に保存するバッファをもたない）

6 節で説明したように，同期型通信の CSP でも，バッファをもつ非同期型通信を形式的に記述可能である．上記の制約を弱めた正当性の証明は今後の課題である．また，今回はトレース詳細化によって依存関係に反するトレースが存在しないことを証明したが，今後は失敗詳細化 [5] によってデッドロックフリー性の証明も予定している．

参考文献

[1] MathWorks Makers of MATLAB and Simulink. 入手先 ⟨http://www.mathworks.co.jp.⟩
[2] 山本 尚平, 鈴木 悠太, 峰田 憲一, 森 裕司, 枝廣 正人: モデルベース並列化における CSP モデルを利用した形式検証の適用, ETNET2017, 電子情報通信学会技術研究報告, Vol. 2017-EMB-44, No. 6, pp.33-38, 2017.
[3] 鍾兆前, 枝廣 正人: モデルベース開発におけるマルチ・メニーコア向け自動並列化, ETNET2017, 電子情報通信学会技術研究報告, Vol. 2017-EMB-44, No. 47, pp.273-278, 2017.
[4] 山口滉平, 竹松慎弥, 池田良裕, 李瑞徳, 鍾兆前, 近藤真己, 枝廣正人: Simulink モデルからのブロックレベル並列化, 情報処理学会 組込みシステムシンポジウム（ESS）, pp.123–124, 2015.
[5] C.A.R.Hoare: Communicating Sequential Processes, Prentice Hall (1985).
[6] 磯部祥尚：並行システムの検証と実装 – 形式手法 CSP に基づく高信頼並行システム開発入門，近代科学社, 2012
[7] R. Milner: Communication and Concurrency, Prentice-Hall (1989).