Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

モデルベース並列化アルゴリズムの定理証明器による形式検証

岩田 駿[1,a)]　磯部 祥尚[2]　枝廣 正人[1]

概要：制御システムの大規模・複雑化にともない，並列化によって高速処理を可能にするメニーコアによる並列制御が注目されている．しかし，並列動作には実行順序逆転やデッドロックなど，逐次動作にはない特有の問題があり，人手による並列化は容易ではない．そこで，我々は制御システムの逐次的なモデルから，それと等価な並列制御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 MBP を開発している．本論文では，MBP の並列化アルゴリズムを形式仕様記述言語 CSP で厳密に記述し，定理証明器 Isabelle を用いて、実行順序逆転やデッドロックが起こらないことを証明したことを報告する．

1. はじめに

制御システムの大規模・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，マルチコア（メニーコア）CPU を利用したシステムの制御が注目を集めている．マルチコアで並列に処理を行うことによって，シングルコアよりも高速に制御することが可能になる．しかし，並列制御では，実行順序逆転やデッドロックなどの並列制御特有の問題が発生する可能性がある．そのため，逐次実行を目的として設計された制御システムをマルチコア上で正しく並列動作するように人手で変換するすることは容易ではない．

我々の研究室では MATLAB/Simulink[1] で作成した制御モデル（以降，**Simulink モデル**と呼ぶ）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** `MBP` を開発している [2][3]．Simulink モデルは，図 1 の左上に示すように，**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は，Simulink モデルのブロック図からブロック間の依存関係（例えば，図 1 の右上）を抽出して，その依存関係を満たすようにブロックのコア割当とコア間通信の挿入を行い，図 1 の右下に示すようなマルチコア用の並列制御モデルを生成する．並列制御モデルでは，依存関係のないブロックを順不同に並列実行することによって，全体の処理時間を短縮することが可能である．

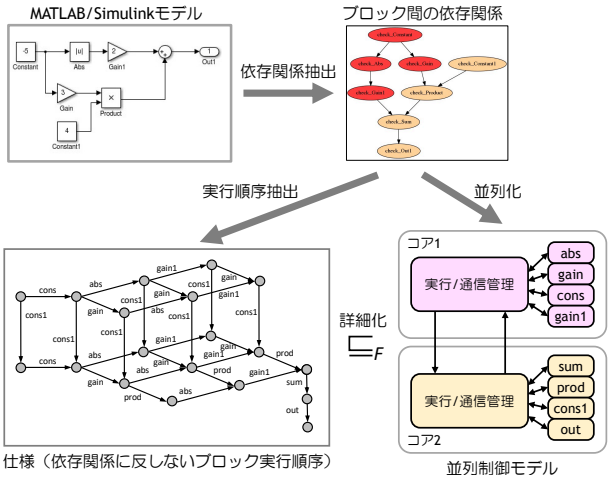

図 1　Simulink モデル，仕様，並列制御モデルの関係

モデルベース並列化システム `MBP` によって生成される並列制御モデルが正しく動作することを保証するためには，並列制御モデルは次の 2 つの性質を満たすことが重要である．

- **実行順序維持**：Simulink モデルの依存関係に反する順番でブロックを実行してはならない．
- **全ブロック実行**：Simulink モデルの実行すべき全てのブロックを実行しなければならない．

並列制御モデルでは，コア間の想定外の相互作用によって，

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)　siwata@ertl.jp

Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

実行順序逆転やデッドロックなどの不具合が発生する可能性がある．このような不具合は発生確率や再現性が低いことが多く，実装後にテストで発見することは困難である．

山本等 [4] や于等 [5] は，Simulink モデルが与えられたときに，MBP によって生成された並列制御モデルが，実行順序維持と全ブロック実行の性質をもつことをモデル検査器によって検証した．モデル検査器では，網羅的な検査によって発生確率の低い不具合も確実に自動で検出できる特長があるが，検査対象（Simulink モデル）を具体的に入力する必要がある．すなわち，任意の Simulink モデルに対する並列化アルゴリズムの正しさを証明することはできない．また，モデル検査には多くのメモリと時間を必要とするため，検査可能なブロック数は数千個程度という上限がある．

多門等 [6] は，フィードバックループをもたない任意の Simulink モデルについて，モデルベース並列化システム MBP が生成する並列制御モデルは上記の実行順序維持の性質を満たすことを証明した．具体的には，MBP 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**[8] で厳密に記述し，Simulink モデルの依存関係を満たすブロック実行順序以外では，並列制御モデルが実行しないことを証明した．しかし，依存関係に反する順番で実行しなければ，途中でデッドロックして停止しても実行順序維持の性質を満たすことはできるため，並列制御モデルがデッドロックしない証明は今後の課題として残されていた．

本研究では，先行研究 [6] の継続として，フィードバックループをもたない任意の Simulink モデルについて，モデルベース並列化システム MBP が生成する並列制御モデルは上記の全ブロック実行の性質を満たすことを，失敗詳細化関係 $\sqsubseteq_F$（図 1 の中央）を用いて証明する．さらに，実行順序維持の証明（先行研究 [6]）と全ブロック実行の証明（本論文の 4 節）を**定理証明器 Isabelle**[7] で形式化する方法（Isabelle の理論ファイルを作成する方法）について説明し，その証明の正しさを検証したことを報告する．

Isabelle には，証明済みの大量の定理が**理論ファイル**として蓄積されており，証明済みの定理は，新しい定理の証明に適用できる．証明されていない定理は理論ファイルに登録できないため，証明ミスが入る可能性は極めて低い[*1]．Isabelle は対話型の定理証明器であり，利用者が証明コマンドを適切に入力する必要があり，その人的証明コストは高いが，帰納法や背理法等，数学の証明法を適用することができ，任意のサイズや依存関係をもつ Simulink モデルに対して，モデルベース並列化システム MBP に関する定理を証明することが可能である．証明に定理証明器 Isabelle を利用することによって次の利点を得ることができる．

- 正しい証明の作成：実行順序維持と全ブロック実行の人手による証明は 70 ページを超えており，証明ミスが入り込んでいる可能性がある．Isabelle では証明過程の正しさを全て機械的に確認するため，証明ミスがないことを保証することができる．
- 第三者による証明の検証：第三者が 70 ページを超える証明を全て詳細に確認することは難しいが，Isabelle 上に形式化された理論ファイル中の証明部分の正しさは Isabelle が保証するため，定義と定理（証明の結果）のみ確認すればよい．
- 証明の支援：本研究にて証明した補題や定理は，Isabelle の理論ファイルとして登録・蓄積されており，今後，並列化アルゴリズムに関連する定理の証明に適用できる．Isabelle の半自動証明コマンドと組み合わせることによって，新しい定理の証明支援ツールとして利用可能である．

以降，2 節で並列制御モデルを形式的に記述するための仕様記述言語 CSP を紹介し，3 節で先行研究 [6] の概要（実行順序維持の証明）を説明する．次に，4 節で全ブロック実行の証明を与え，5 節で実行順序維持と全ブロック実行の証明を定理証明器 Isabelle によって形式検証する方法について説明する．最後に，6 節で今後の課題について述べる．

2. 形式仕様記述言語 CSP

本研究では，並列化アルゴリズムの形式的な記述と証明に CSP（Communicating Sequential Processes）を用いる．CSP[8] は並列動作を厳密に記述する形式言語と解析する方法からなる形式手法（プロセス代数のひとつ）である．本節では文献 [9] にしたがって，本論文で用いる CSP の構文論と意味論，詳細化関係を紹介する．

2.1 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 Events で表し，その要素を $a, b, \ldots$ で表す．また，CSP では繰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 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 2.1 CSP のプロセスの集合 Proc（要素を $P, Q, \cdots$ で表す）は図 2 [*2]の式を含む最小の集合である．ここで，P, Q はすでに Proc の要素であるとする．

以下，図 2 の各プロセスの意味を簡単に説明しておく．停止（STOP）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（$a \to P$）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（$P \,\square\, Q$）と内部選択（$P \sqcap Q$）はプロセス P または

[*1] Isabelle 自身にバグがある可能性は否定できないが，Isabelle は 1990 年頃から研究開発が継続して行われきた実績がある．

[*2] 図 2 には，5 節で定理証明器 Isabelle 上に定義する CSP の構文も併記している．

Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

プロセス	(Isabelle 版)	意味	
STOP	STOP	停止	
$a \to P$	a -> P	プレフィクス	($a \in$ Events)
$P \,\square\, Q$	P [+] Q	外部選択	
$P \sqcap Q$	P |~| Q	内部選択	
$b \,\&\, P$	b &> P	ガード	($b \in$ Bool)
$P \,[\![X]\!]\, Q$	P [|X|] Q	並行合成	($X \subseteq$ Events)
$P \backslash X$	P ~~ X	隠蔽	($X \subseteq$ Events)
A	$A	プロセス名	($A \in$ PName)

図 2 プロセスの構文 ($a \in \mathtt{Events}, \alpha \in \mathtt{Events}_\tau$)

Q のように動作するプロセスであり，その選択を外部選択は外部から選択でき，内部選択は外部から選択できない選択である．ガード ($b \,\&\, P$) 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．並行合成 ($P \,[\![X]\!]\, Q$) 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 ($P\backslash X$) 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．プロセス名 (A) は定義式 ($A = P$) によって振舞いを与えられるプロセスであり，主に繰返し動作を表現するために使われる．

例えば，2 つのイベント lock と unlock を交互に繰り返し，イベント finish で停止する鍵のプロセス Key の振舞いは，CSP で次のように記述できる．

$$\mathtt{Key} = (\mathtt{lock} \to \mathtt{unlock} \to \mathtt{Key}) \,\square\, (\mathtt{finish} \to \mathtt{STOP})$$

便宜上，3 つ以上の外部選択を記述するために，次の複製型の略記法も用いる（本論文では空集合を {} で表す）．

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

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．

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

2.2 意味論

CSP のプロセスは観測可能なイベント ($a \in$ 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 Events に内部イベント τ を追加した集合を $\mathtt{Events}_\tau$ ($= \mathtt{Events} \cup \{\tau\}$) で表し，その要素を $\alpha, \beta, \ldots$ で表す．このとき，CSP のプロセスの意味はラベル付遷移システム $(\mathtt{Proc}, \mathtt{Events}_\tau, \{\xrightarrow{\alpha} \mid \alpha \in \mathtt{Events}_\tau\})$ をもとに与えられる．ここで，$\xrightarrow{\alpha}$ は定義 2.2 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．

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

$$\mathbf{ExtCh_1}\ \frac{P \xrightarrow{a} P'}{P \,\square\, Q \xrightarrow{a} P'}$$

$$\mathbf{ExtCh_2}\ \frac{Q \xrightarrow{a} Q'}{P \,\square\, Q \xrightarrow{a} Q'}$$

$$\mathbf{IntCh_1}\ \frac{}{P \sqcap Q \xrightarrow{\tau} P}$$

$$\mathbf{IntCh_2}\ \frac{}{P \sqcap Q \xrightarrow{\tau} Q}$$

$$\mathbf{Guard}\ \frac{P \xrightarrow{\alpha} P'}{b \,\&\, P \xrightarrow{\alpha} P'} (b = \mathtt{True})$$

$$\mathbf{Para_1}\ \frac{P \xrightarrow{\alpha} P'}{P \,[\![X]\!]\, Q \xrightarrow{\alpha} P' \,[\![X]\!]\, Q} (\alpha \notin X)$$

$$\mathbf{Para_2}\ \frac{Q \xrightarrow{\alpha} Q'}{P \,[\![X]\!]\, Q \xrightarrow{\alpha} P \,[\![X]\!]\, Q'} (\alpha \notin X)$$

$$\mathbf{Para_3}\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \,[\![X]\!]\, Q \xrightarrow{a} P' \,[\![X]\!]\, Q'} (a \in X)$$

$$\mathbf{Hide_1}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash X \xrightarrow{\alpha} P'\backslash X} (\alpha \notin X)$$

$$\mathbf{Hide_2}\ \frac{P \xrightarrow{a} P'}{P\backslash X \xrightarrow{\tau} P'\backslash X} (a \in X)$$

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

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

定義 2.2 プロセス間の遷移関係 $\xrightarrow{\alpha} \subseteq \mathtt{Proc} \times \mathtt{Proc}$ 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．

例えば，2.1 小節で例示した鍵のプロセス Key が lock を実行でき，その後はプロセス unlock → Key のように動作することを表す遷移は次のように導出できる．

$$\mathbf{PName}\ \frac{\mathbf{ExtCh_1}\ \dfrac{\mathbf{Prefix}\ \dfrac{}{\mathtt{lock} \to \mathtt{unlock} \to \mathtt{Key} \xrightarrow{\mathtt{lock}} \mathtt{unlock} \to \mathtt{Key}}}{(\mathtt{lock} \to \mathtt{unlock} \to \mathtt{Key}) \,\square\, (\mathtt{finish} \to \mathtt{STOP}) \xrightarrow{\mathtt{lock}} \mathtt{unlock} \to \mathtt{Key}}}{\mathtt{Key} \xrightarrow{\mathtt{lock}} \mathtt{unlock} \to \mathtt{Key}}$$

本論文では，プロセス P がイベント α による遷移をもつ ($\exists P'.\ P \xrightarrow{\alpha} P'$) ことを $P \xrightarrow{\alpha}$ と記述し，α による遷移をもたないことを $P \overset{\alpha}{\not\longrightarrow}$ と記述する．

2.3 トレース詳細化

トレース詳細化とは，2 つの CSP のプロセスが実行可能なイベントの列の包含関係を比較するための関係である．先行研

Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

究 [6] では，このトレース詳細化を用いて，並列制御モデルが実行順序維持の性質を満たすこと，すなわち，ブロックの実行順序逆転が発生しないことを証明した．本小節では，トレース詳細化について説明する．

まず，イベントの列 $t = \alpha_1 \cdots \alpha_n \in \mathtt{Events}_\tau^*$ 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．

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

また，後の 3.4 小節で説明する弱模倣関係で用いる弱い遷移 $\stackrel{\alpha}{\Longrightarrow}$ についてもここで定義しておく．この弱い遷移は，1 つのイベント（$\alpha \in \mathtt{Events}_\tau$）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．

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

次に，イベント列 $t \in \mathtt{Events}_\tau^*$ 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 $\widehat{t} \in \mathtt{Events}^*$ と書く．例えば，$t = \langle \mathtt{a}, \tau, \mathtt{b}, \tau, \tau, \mathtt{c}, \tau \rangle$ ならば，$\widehat{t} = \langle \mathtt{a}, \mathtt{b}, \mathtt{c} \rangle$ である．

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 2.3 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．

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

このとき，トレース詳細化は次の定義 2.4 により定義される．すなわち，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 2.4 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．

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

2.4 失敗詳細化

トレース詳細化は，許容されないイベント（例えば，実行順序が逆転したイベント）がないことを保証するために有効であるが，実行すべきイベント（例えば，全てのブロックの実行イベント）を実行することは保証できない．本小節では，実行すべきイベントを実行することを保証するために失敗詳細化を定義する．

まず，プロセス P が安定状態（内部イベント τ を実行できない状態）に至るトレース s とその安定状態で実行できないイベントの集合 X の組 (s, X) の集合を失敗集合 $\mathtt{failures}(P)$ として，次のように定義する．

定義 2.5 プロセス $P \in \mathtt{Proc}$ について，**失敗集合**を次のように定義する．

$$\mathtt{failures}(P) = \{\, (\widehat{t}, X) \mid \exists P'.\ P \xrightarrow{t} P', P' \stackrel{\tau}{\not\longrightarrow}, X \subseteq \mathtt{refs}(P') \}$$

ここで，$\mathtt{refs}(P)$ は P が実行できないイベントの集合であり，次のように定義される．

$$\mathtt{refs}(P) = \{a \in \mathtt{Events} \mid P \stackrel{a}{\not\longrightarrow},\ P \stackrel{\tau}{\not\longrightarrow}\}$$

このとき，失敗詳細化は次の定義 2.6 により定義される．すなわち，プロセス Q のトレース集合と失敗集合が，各々 P のトレース集合と失敗集合に含まれるとき，Q は P の失敗詳細化であるといい，$P \sqsubseteq_F Q$ と書く．

定義 2.6 プロセス Q は P の**失敗詳細化**（$P \sqsubseteq_F Q$）であることを次のように定義する．

$$P \ \sqsubseteq_F \ Q \iff ((\mathtt{traces}(P) \supseteq \mathtt{traces}(Q)) \wedge (\mathtt{failures}(P) \supseteq \mathtt{failures}(Q)))$$

本来の CSP[8] では成功終了を表す特殊なイベント $\surd$ が定義されているが，本研究では意味論を簡単にするため $\surd$ は省略している．そのため，本研究ではイベント `finish` を成功終了とみなして，デッドロックフリー性を次のように定義する．

定義 2.7 プロセス P が次の性質を満たすとき，P はイベント集合 A について**デッドロックフリー**であるという．

$$\forall (s, X) \in \mathtt{failures}(P).\ (X \neq A \vee \mathtt{finish} \in \mathtt{set}(s))$$

ここで，$\mathtt{set}(s)$ はトレース s に含まれるイベントの集合である．

すなわち，プロセス P がイベント集合 A についてデッドロックフリーとは，終了イベント `finish` を実行するまでは，常に A の中のイベントを少なくともひとつは実行可能である（全てのイベントの実行が拒否されることはない）ことを意味している．例えば，2.1 小節で例示した鍵のプロセス `Key` はイベント集合 $\{\mathtt{lock}, \mathtt{unlock}, \mathtt{finish}\}$ についてデッドロックフリーである．

次の命題 2.1 に示すように，デッドロックフリー性は失敗詳細化によって保存される．

命題 2.1 プロセス P はイベント集合 A についてデッドロックフリーであり，かつ $P \sqsubseteq_F Q$ ならば，Q もイベント集合 A についてデッドロックフリーである．

証明 デッドロックフリーと失敗詳細化の定義より，$(s, X) \in \mathtt{failures}(Q)$ とすると，$\mathtt{failures}(Q) \subseteq \mathtt{failures}(P)$ であるので，$X \neq A$ または $\mathtt{finish} \in \mathtt{set}(s)$ を得る． ∎

3. 並列制御モデルの実行順序維持性 [6]

本節では，先行研究 [6] をもとに，モデルベース並列化システム `MBP` の並列化アルゴリズム（簡略版）によって生成される並列制御モデル `ParaCtrl` と仕様 `SeqCtrl` の形式記述と，並列制御モデルが実行順序維持の性質を満たす定理について説明する．本節の詳細については，先行研究 [6] を参照してほしい．

3.1 並列化アルゴリズムへの入力

並列化アルゴリズムへの入力はブロック間の依存関係とコア割当合関数である．Simulink モデルの制御ブロック間の依存関係は有向グラフ $(\mathtt{Blks}, \mathtt{Deps})$ によって形式化される．ここで，`Blks` はブロック ID の集合，$\mathtt{Deps} \subseteq \mathtt{Blks} \times \mathtt{Blks}$ は信号線で接

Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

$$
\begin{aligned}
&\texttt{ParaCtrl} = (\texttt{Bfr}(1,\{\})\ [|\ \texttt{ComFin}\ |]\ \texttt{Bfr}(2,\{\}))\backslash\texttt{Com}\\
&\texttt{Bfr}(c,S) = \quad (S = \texttt{Blks})\ \&\ \texttt{finish} \rightarrow \texttt{STOP}\\
&\qquad\qquad \Box\ (S \neq \texttt{Blks})\ \&\\
&\qquad\qquad\quad (\quad (\ \Box\ n : \texttt{enable}(S) \cap \texttt{Asn}(c)\ @\\
&\qquad\qquad\qquad\qquad \texttt{blk}.n \rightarrow \texttt{Aft}(c,S,n))\\
&\qquad\qquad\qquad \Box\ (\texttt{com.co}(c).c?m \rightarrow \texttt{Bfr}(c, S \cup \{m\})))\\
&\texttt{Aft}(c,S,n) = \quad (\texttt{com}.c.\texttt{co}(c)!n \rightarrow \texttt{Bfr}(c, S \cup \{n\}))\\
&\qquad\qquad \Box\ (\texttt{com.co}(c).c?m \rightarrow \texttt{Aft}(c, S \cup \{m\}, n))\\
&\\
&\quad \texttt{ComFin} = \texttt{Com} \cup \{\texttt{finish}\}\\
&\qquad \texttt{Com} = \{e \mid \exists c, c', n.\ \ e = \texttt{com}.c.c'.n \in \texttt{Events}\}\\
&\texttt{enable}(S) = \{n \in \texttt{Blks} \mid \texttt{src}(n) \subseteq S, n \notin S\}\\
&\quad \texttt{src}(n) = \{m \in \texttt{Blks} \mid (m,n) \in \texttt{Deps}\}\\
&\qquad \texttt{co}(c) = 2 - c
\end{aligned}
$$

図 4 並列制御モデルの CSP 記述 ParaCtrl（先行研究 [6] の図 5）

続されたブロックの組の集合である．例えば，$(b_1, b_2) \in \texttt{Deps}$ 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，ここ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しており，依存関係を表す有向グラフは閉路をもたない有向非巡回グラフである．

コア割当関数 $\texttt{Asn}(c) \subseteq \texttt{Blks}$ はコア c に割り当てられるブロックの集合を返す関数である．先行研究 [6] ではコア数を 2 個に制限しており，$\texttt{Asn}(1) \cup \texttt{Asn}(2) = \texttt{Blks}$ かつ $\texttt{Asn}(1) \cap \texttt{Asn}(2) = \{\}$ である．本研究でも同様にコア数は 2 個に制限する．

3.2 並列制御モデルの形式化

並列化アルゴリズムに，有向グラフ $(\texttt{Blks}, \texttt{Deps})$ と割当関数 Asn を入力して生成される並列制御モデルのプロセス（CSP 記述）$\texttt{ParaCtrl}_{(\texttt{Blks},\texttt{Deps},\texttt{Asn})}$ を図 4 に示す．以降，文脈から明確な場合はパラメータ $(\texttt{Blks}, \texttt{Deps}, \texttt{Asn})$ を省略し，$\texttt{ParaCtrl}_{(\texttt{Blks},\texttt{Deps},\texttt{Asn})}$ を ParaCtrl と略記する．

並列制御モデル ParaCtrl は，コア $c \in \{1, 2\}$ に割り当てられたプロセス $\texttt{Bfr}(c, S)$ を並列に実行する．コア c のプロセス $\texttt{Bfr}(c, S)$ は実行済みのブロック ID の集合 S から，次に実行可能なブロック ID $n \in \texttt{enable}(S) \cap \texttt{Asn}(c)$ を求め，ブロック n（イベント $\texttt{blk}.n$）を実行し，その実行後はプロセス $\texttt{Aft}(c, S, n)$ のように振舞う．プロセス $\texttt{Aft}(c, S, n)$ は他のコア $\texttt{co}(c)$ に，チャネル $\texttt{com}.c.\texttt{co}(c).n$ をとおして実行したブロック IDn を送信して，その後は $\texttt{Bfr}(c, S \cup \{n\})$ のように振舞う．一方，各コアのプロセスは常に他のコアで実行されたブロック ID をチャネル $\texttt{com.co}(c).c.n$ をとおして受信できる．各コアのプロセス $\texttt{Bfr}(c, S)$ は全ブロックを実行後（$S = \texttt{Blks}$）に成功終了（finish）する．

3.3 振舞い仕様の形式化

並列制御モデル ParaCtrl の仕様として，Simulink モデル $(\texttt{Blks}, \texttt{Deps})$ の依存関係を満たす全てのブロック実行順序を含むプロセス SeqCtrl の CSP による形式記述を図 5 に示す．もし並列制御モデル ParaCtrl が仕様 SeqCtrl に含まれないトレース（ブロック実行列）を実行するならば，それは依存関係に反する実行順序逆転が起きていることを意味する．

$$
\begin{aligned}
&\texttt{SeqCtrl} = \texttt{Seq}(\{\})\\
&\texttt{Seq}(S) = \quad (S = \texttt{Blks})\ \&\ \texttt{finish} \rightarrow \texttt{STOP}\\
&\qquad\qquad \Box\ (S \neq \texttt{Blks})\ \&\ (\ \Box\ n : \texttt{enable}(S)\ @\\
&\qquad\qquad\qquad\qquad \texttt{blk}.n \rightarrow \texttt{Seq}(S \cup \{n\}))
\end{aligned}
$$

図 5 仕様 SeqCtrl の CSP による形式記述（先行研究 [6] の図 7）

$$
\begin{aligned}
\texttt{Sim} &= \textstyle\bigcup_{i \in \{1,\ldots,6\}} \texttt{Sim}_i\\
\texttt{Sim}_1 &= \{(\texttt{ParaCtrl}, \texttt{SeqCtrl})\}\\
\texttt{Sim}_2 &= \{((\texttt{Bfr}(1,S)\ [|\ \texttt{ComFin}\ |]\ \texttt{Bfr}(2,S))\backslash\texttt{Com}, \texttt{Seq}(S))\\
&\qquad \mid S \subseteq \texttt{Blks}\}\\
\texttt{Sim}_3 &= \{((\texttt{Aft}(1,S,n)\ [|\ \texttt{ComFin}\ |]\ \texttt{Bfr}(2,S))\backslash\texttt{Com}, \texttt{Seq}(S'))\\
&\qquad \mid S' = S \cup \{n\}, S' \subseteq \texttt{Blks}, n \notin S, n \in \texttt{Asn}(1)\}\\
\texttt{Sim}_4 &= \{((\texttt{Bfr}(1,S)\ [|\ \texttt{ComFin}\ |]\ \texttt{Aft}(2,S,m))\backslash\texttt{Com}, \texttt{Seq}(S'))\\
&\qquad \mid S' = S \cup \{m\}, S' \subseteq \texttt{Blks}, m \notin S, m \in \texttt{Asn}(2)\}\\
\texttt{Sim}_5 &= \{((\texttt{Aft}(1,S,n)\ [|\ \texttt{ComFin}\ |]\ \texttt{Aft}(2,S,m))\backslash\texttt{Com}, \texttt{Seq}(S'))\\
&\qquad \mid S' = S \cup \{n,m\}, S' \subseteq \texttt{Blks}, n \notin S, m \notin S,\\
&\qquad\quad n \in \texttt{Asn}(1), m \in \texttt{Asn}(2)\}\\
\texttt{Sim}_6 &= \{((\texttt{STOP}\ [|\ \texttt{ComFin}\ |]\ \texttt{STOP})\backslash\texttt{Com}, \texttt{STOP})\}
\end{aligned}
$$

図 6 プロセスの二項関係 Sim（先行研究 [6] の図 8）

3.4 実行順序維持の証明

多門等 [6] は，Simulink モデルの依存関係に反する順番でブロックを実行しないことを保証するために，図 4 の並列処理モデル ParaCtrl が図 5 の仕様 SeqCtrl のトレース詳細化（$\texttt{SeqCtrl} \sqsubseteq_T \texttt{ParaCtrl}$）であることを次の手順で証明した：

(1) 2 つの CSP プロセス P と Q の間に弱模倣関係 $\mathcal{S}$（各 $(P,Q) \in \mathcal{S}$ について，$P \xrightarrow{\alpha} P'$ ならば $Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q'$ かつ $(P',Q') \in \mathcal{S}$ となる Q' が存在する）を定義し（定義 3.8[6]），

(2) P と Q が弱模倣関係にあるならば，P は Q のトレース詳細化であることを証明し（命題 3.1[6]），

(3) 図 6 の $(\texttt{ParaCtrl}, \texttt{SeqCtrl})$ を含む関係 Sim が弱模倣関係であることを証明した（補題 5.2[6]）．

トレース詳細化の定義では，P の各遷移列 $P \xrightarrow{s}$ に対応する Q の遷移列の存在が要求されており，この関係はトレース s の長さに関する帰納法を用いて証明できる．一方，弱模倣関係 $\mathcal{S}$ では，適切な弱模倣関係 $\mathcal{S}$ を用意する必要はあるが，P の 1 回の遷移 $\xrightarrow{\alpha}$ ごとに Q との対応関係を示せばよく，よりわかりやすい証明が可能になる．

4. 並列制御モデルの全ブロック実行性

先行研究 [6] では，実行順序の逆転が起こらないことを証明するためにトレース詳細化（$\texttt{SeqCtrl} \sqsubseteq_T \texttt{ParaCtrl}$）を証明した．トレース詳細化は許可されないトレースが実行されないことを要求するための関係であり，実行すべきイベントを実行することは要求しない．例えば，停止プロセスは何も実行しないため，

仕様 SeqCtrl のトレース詳細化である（$\texttt{SeqCtrl} \sqsubseteq_T \texttt{STOP}$）．

仕様 SeqCtrl は，Simulink モデルの依存関係に反しない順番でイベント $\texttt{blk}.n$ を実行し，全イベントを実行後に終了イベント finish を実行するプロセスであるが，仕様自身が途中で停止する（デッドロックする）と失敗詳細化関係 $\texttt{SeqCtrl} \sqsubseteq_F \texttt{ParaCtrl}$ を証明しても，並列制御モデル ParaCtrl が全ブロックを実行することを証明したことにはならない．

本節では，まず，4.1 小節で，仕様 SeqCtrl がデッドロックフリーであることを証明し，次に，4.2 小節で，並列制御モデル ParaCtrl は仕様 SeqCtrl の（トレース詳細化だけでなく）失敗詳細化でもあること（$\texttt{SeqCtrl} \sqsubseteq_F \texttt{ParaCtrl}$）を証明する．

4.1 仕様のデッドロックフリー性の証明

先行研究 [6] では，実行順序の逆転が起こらないこと（実行順序維持）を証明するために仕様 SeqCtrl を定義した．その証明では仕様 SeqCtrl が途中で停止しないことは重要ではないが，全てのブロックが実行されることを保証するためには，仕様 SeqCtrl が停止しないことを証明する必要がある．

図 5 の仕様 SeqCtrl は逐次的なプロセスであり，その振舞いは明確であるが，その振舞いは関数 $\texttt{enable}(S)$ により定まる．まず，実行済みのブロック ID の集合 S が全ブロック ID 集合 Blks の真部分集合（$S \subset \texttt{Blks}$）ならば，$\texttt{enable}(S)$ が空集合にならないことを意味する命題 4.1 を与える．

命題 4.1 $(\texttt{Blks}, \texttt{Deps})$ は有向非巡回グラフ，$S \subset \texttt{Blks}$（$S \neq \texttt{Blks}$）とする．このとき，$\texttt{enable}(S) \neq \{\}$ である．
証明 背理法によって証明するため，$\texttt{enable}(S) = \{\}$ を仮定する．すなわち，enable の定義より，全ての $n \in \texttt{Blks} - S$ について，$(n', n) \in \texttt{Deps}'$ となる $n' \in \texttt{Blks} - S$ が存在する．ここで，$\texttt{Deps}' = \texttt{Deps} - ((\texttt{Blks} \times S) \cup (S \times \texttt{Blks}))$ である．このとき，次に示す補題 4.2 より，$(\texttt{Blks} - S, \texttt{Deps}')$ は閉路をもつ．しかし，$(\texttt{Blks} - S, \texttt{Deps}')$ は $(\texttt{Blks}, \texttt{Deps})$ の部分グラフであるので，これは条件の有向非巡回グラフであることに矛盾する．よって，$\texttt{enable}(S) \neq \{\}$ である． ∎

補題 4.2 N は空ではない有限集合とする．このとき，有向グラフ (N, E) について，全ての $n \in N$ に対し，$(n', n) \in E$ となる $n' \in N$ が存在するならば，(N, E) は閉路をもつ．
証明 N のサイズに対する帰納法で証明する．ノード $n \in N$ について，有限集合 $M = \{m \mid (m, n) \in E^+\}$ を定義する（E^+ は E の推移閉包）．もし $n \in M$ ならば，$(n, n) \in E^+$ であるので N は閉路をもつ．もし $n \notin M$ ならば，(N, E) の部分有向グラフ $(M, E \cap (M \times M))$ は本補題の仮定を満たす．M のサイズは N よりも小さいため，帰納法の仮定より，$(M, E \cap (M \times M))$ は閉路をもつ．以上，帰納法により (N, E) も閉路をもつ． ∎

命題 4.1 より，仕様 SeqCtrl のデッドロックフリー性に関する次の定理 4.3 が成り立つ．

定理 4.3 $(\texttt{Blks}, \texttt{Deps})$ は有向非巡回グラフ，Blks は有限集合とする．このとき，SeqCtrl は $\{\texttt{finish}\} \cup \{\texttt{blk}.n \mid n \in \texttt{Blks}\}$ についてデッドロックフリーである．
証明（概要）仕様 SeqCtrl は，実行済みのブロック ID の集合 S が全ブロック ID 集合 Blks の真部分集合ならば，$n \in \texttt{enable}(S)$ を満たすブロック $\texttt{blk}.n$ を実行でき，実行後に n を S に追加する動作を繰り返す．命題 4.1 より，途中で $\texttt{enable}(S)$ が空集合にならないため，S は増え続け，いつかは S は Blks に等しくなり，（全てのブロックが実行済みになり），finish を実行して終了する． ∎

4.2 全ブロック実行の証明

本小節では失敗詳細化関係を証明するために，まず，弱失敗模倣関係を定義 4.1 のように定義する．

定義 4.1 プロセス上の二項関係 $\mathcal{S} \subseteq \texttt{Proc} \times \texttt{Proc}$ が**弱失敗模倣関係**であるとは，$(P, Q) \in \mathcal{S}$ ならば，任意の $\alpha \in \texttt{Events}_\tau$ と $P' \in \texttt{Proc}$ について，次の 2 つの条件を満たすことである．

$$(i) \quad P \xrightarrow{\alpha} P' \;\Rightarrow\; (\exists Q'.\ Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q' \wedge (P', Q') \in \mathcal{S}),$$
$$(ii) \quad (P \xrightarrow{\tau}\ ) \vee (P \stackrel{\alpha}{\not\longrightarrow} \;\Rightarrow\; Q \stackrel{\alpha}{\longrightarrow})$$

命題 4.4 に示すように，$P \sqsubseteq_F Q$ を証明するためには，$(Q, P) \in \mathcal{S}$ となる弱失敗模倣関係 $\mathcal{S}$ の存在を示せば十分である．

命題 4.4 $(P, Q) \in \mathcal{S}$ となる弱失敗模倣関係 $\mathcal{S}$ が存在するならば，$Q \sqsubseteq_F P$ である．
証明 $(P, Q) \in \mathcal{S}$ となる弱失敗模倣関係 $\mathcal{S}$ が存在すると仮定する．このとき，先行研究 [6] の命題 3.1 より，条件 (i) から $\texttt{traces}(P) \subseteq \texttt{traces}(Q)$ は成り立つため，ここでは，$\texttt{failures}(P) \subseteq \texttt{failures}(Q)$ を証明する．
$(s, X) \in \texttt{failures}(P)$ とする．すなわち，s は τ を含まず，$s = \widehat{t}$，$P \xrightarrow{t} P'$ となる t と P' が存在し，$P' \stackrel{\tau}{\not\longrightarrow}$，$X \subseteq \texttt{refs}(P')$ である．このとき，$(P, Q) \in \mathcal{S}$ であるので，トレース t の長さに関する帰納法を適用して，条件 (i) より，$Q \stackrel{\widehat{t}}{\Longrightarrow} Q'$ かつ $(P', Q') \in \mathcal{S}$ を満たす Q' が存在する．すなわち，$Q \xrightarrow{t'} Q'$ かつ $s = \widehat{t'}$ を満たす t' も存在する．また，$(P', Q') \in \mathcal{S}$ と条件 (ii) より，もし $P' \stackrel{a}{\not\longrightarrow}$，$a \neq \tau$ ならば，$Q' \stackrel{a}{\longrightarrow}$ である．すなわち，$\texttt{refs}(P') \subseteq \texttt{refs}(Q')$ を得る．以上，$\texttt{failures}(P) \subseteq \texttt{failures}(Q)$ が成り立つ． ∎

本小節では，Simulink モデルの全てのブロックが実行されることを保証するために，図 4 の並列処理モデル ParaCtrl が図 5 の仕様 SeqCtrl の失敗詳細化（$\texttt{SeqCtrl} \sqsubseteq_F \texttt{ParaCtrl}$）であることを証明する．まず，図 6 の弱模倣関係 Sim は弱失敗模倣関係であることを意味する補題 4.5 を与える．

補題 4.5 $\texttt{Asn}(1) \cap \texttt{Asn}(2) = \{\}$，$\texttt{Asn}(1) \cup \texttt{Asn}(2) = \texttt{Blks}$ とする．このとき，図 6 の関係 Sim は弱失敗模倣関係である．
証明 $(P, Q) \in \texttt{Sim}$ とする．このとき，弱失敗模倣関係の定義 4.1 の条件 (i) が満たされることは先行研究 [6] の補題 5.2 に証明されているため，ここでは，条件 (ii) が満たされることを示す．まず，(P, Q) が図 6 の関係 $\texttt{Sim}_2$ に含まれる場合，すなわち，$S \subseteq \texttt{Blks}$，

Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

$$P = (\mathtt{Bfr}(1,S)\ [|\ \mathtt{ComFin}\ |]\ \mathtt{Bfr}(2,S))\backslash\mathtt{Com},$$
$$Q = \mathtt{Seq}(S)$$

の場合に定義 4.1 の条件 (ii) を満たすことを示す．
条件 (ii) の「$P \xrightarrow{\alpha}\!\!\!\!\!\!/\ \ $ ならば $Q \xrightarrow{\alpha}\!\!\!\!\!\!/\ \ $」の対偶を証明するため，$Q = \mathtt{Seq}(S) \xrightarrow{\alpha}$ を仮定する．この遷移 $\mathtt{Seq}(S) \xrightarrow{\alpha}$ の導出に適用された可能性がある遷移規則を網羅的に確認して，この遷移が存在するための必要十分条件として，次の (a) または (b) を得る．

(a) $S = \mathtt{Blks} \wedge \alpha = \mathtt{finish}$,
(b) $\exists n \in \mathtt{enable}(S).\ S \neq \mathtt{Blks} \wedge \alpha = \mathtt{blk}.n$

- 上記の条件 (a) が成り立つ場合：遷移規則を適用して，次の遷移を導出できる．

$$P = (\mathtt{Bfr}(1,S)\ [|\ \mathtt{ComFin}\ |]\ \mathtt{Bfr}(2,S))\backslash\mathtt{Com} \xrightarrow{\mathtt{finish}}$$

- 上記の条件 (b) が成り立つ場合：$n \in \mathtt{enable}(S)$ となる n が存在する．ここで，$\mathtt{Asn}(1) \cup \mathtt{Asn}(2) = \mathtt{Blks}$ であるので，$n \in \mathtt{enable}(S) \cap \mathtt{Asn}(1)$ または $n \in \mathtt{enable}(S) \cap \mathtt{Asn}(2)$ が成り立つ．このとき，外部選択等の遷移規則を適用して，$\mathtt{Bfr}(1,S) \xrightarrow{\mathtt{blk}.n}$ または $\mathtt{Bfr}(2,S) \xrightarrow{\mathtt{blk}.n}$ を導出できる．さらに並行合成等の遷移規則を適用すると，どちらの場合でも，$P \xrightarrow{\mathtt{blk}.n}$ が得られる．

以上，$(P,Q) \in \mathtt{Sim}_2$ ならば条件 (ii) を満たすことを証明した．$(P,Q) \in \mathtt{Sim}_2$ 以外の場合については，$(P,Q) \in \mathtt{Sim}_{3,4,5}$ ならば，遷移規則を適用して $P \xrightarrow{\tau}$ を導出できるため，定義 4.1 の条件 (ii) を満たす．また，$(P,Q) \in \mathtt{Sim}_6$ ならば P も Q も遷移をもたないため，明らかに条件 (ii) を満たす．最後に，$(P,Q) \in \mathtt{Sim}_1$ は $\mathtt{Sim}_2$ の特殊な場合（初期状態）である． ∎

これまでに証明してきた補題と命題より，次の定理を得る．

定理 4.6 $\mathtt{Asn}(1) \cap \mathtt{Asn}(2) = \{\}$ かつ $\mathtt{Asn}(1) \cup \mathtt{Asn}(2) = \mathtt{Blks}$ ならば，$\mathtt{SeqCtrl} \sqsubseteq_F \mathtt{ParaCtrl}$ である．
証明 補題 4.5 と命題 4.4 より証明できる． ∎

定理 4.7 $(\mathtt{Blks}, \mathtt{Deps})$ は有向非巡回グラフ，$\mathtt{Blks}$ は有限集合，$\mathtt{Asn}(1) \cap \mathtt{Asn}(2) = \{\}$ かつ $\mathtt{Asn}(1) \cup \mathtt{Asn}(2) = \mathtt{Blks}$ ならば，$\mathtt{ParaCtrl}$ は $\{\mathtt{finish}\} \cup \{\mathtt{blk}.n \mid n \in \mathtt{Blks}\}$ についてデッドロックフリーである．
証明 命題 2.1 と定理 4.6 より証明できる． ∎

5. 定理証明器 Isabelle による形式検証

Isabelle[7] は定理の証明を支援するための対話型のツール（**定理証明器**）であり，Isabelle で証明済みの様々な定理が**理論ファイル**（拡張子は.thy）に保存され，提供されている．利用者は新しい定理（証明対象）を**ゴール**として Isabelle に入力し，証明指示を**証明コマンド**によって与えると，Isabelle は，その証明コマンドにしたがい，証明済みの定理を可能な限り自動的に適用して，新しい定理の証明を試みる．Isabelle は自動証明に行き詰まると，証明の途中結果を**サブゴール**として表示して待機状態になる．利用者はそのサブゴールを検討し，次の証明コマンドを与える．証明完了した定理は新たに理論ファイルに保存でき，次以降の他の定理の証明に適用できるようになる．このように，新たな理論ファイルは既存の（複数の）理論ファイルの上に作成されるため，Isabelle の理論ファイルは階層構造をもつ．

先行研究 [6] の実行順序維持や本論文 4 節の全ブロック実行に関係する補題や定理の証明は全部で 70 ページ以上になり，その証明にミスがないことを人手で確認することは容易ではない．そこで，本研究に関する全ての補題と定理の証明の正しさを Isabelle で形式的に検証した．

その形式検証のために作成した理論ファイルの階層構造を図 7 に示す．この理論ファイルは**汎用部分（LW-CSP-Prover）**と**並列化アルゴリズム部分（MBP-Prover）**から構成されている．LW-CSP-Prover の理論ファイルの行数の合計は約 1,000 行，MBP-Prover の理論ファイルの行数の合計は約 2,000 行である．以下，5.1 小節と 5.2 小節で，各々，汎用部分と並列化アルゴリズム部分の理論ファイルについて説明する．

5.1 汎用部分（LW-CSP-Prover）

定理証明器 Isabelle で CSP の全ての構文論と表示的意味論を定義し，トレース詳細化や失敗詳細化に関する証明済みの定理（CSP 規則）等をまとめた理論ファイルの既存研究として CSP-Prover[10] がある．一方，4.2 小節の補題 4.5 の証明にみられるように，本研究では証明に操作的意味論（遷移規則）を利用している．本研究でも，CSP-Prover に表示的意味論を追加定義して使用することも検討したが，本研究で必要な構文論は CSP の一部（部分言語）であるため，CSP-Prover を参考に必要最小限の構文論と操作的意味論を定義して，軽量（lightweight）な CSP-Prover（LW-CSP-Prover と呼ぶ）の理論ファイル（図 7 左側）を作成した．

図 8 に示すように，CSP のプロセスの各演算子（→, □ 等）は，再帰型 `('p,'a) Proc` の型構成子（`Prefix`, `ExtCh` 等）として，Isabelle のキーワード **datatype** により定義できる．ここで，`'a` はイベントの型変数，`'p` はプロセス名の型変数である．また，図 8 中の `("_ -> _")` は，(`Prefix` a P) を (a `->` P) のように記述することを可能にしており，LW-CSP-Prover でも CSP に似た構文（図 2 の Isabelle 版参照）を利用できる．なお，CSP ではプロセス名 A もプロセスの一部として定義されているが，LW-CSP-Prover ではプロセス名の型`'p` とプロセスの型 `('p,'a) proc` が異なるため，プロセス名をプロセスに変換する型構成子`$`が付けられている．

2.2 小節で説明した CSP の表示的意味論（図 3 の遷移規則）は，図 9 に示すように，LW-CSP-Prover では帰納的な述語 `trn` として，Isabelle のキーワード **inductive** によって定義できる．ここで，`'a event` はイベントの型であり，通常のイベント `(Ev a)`[*3] と内部イベント `Tau` から構成されている．また，図 9 の推論規則 `PName` の `PNfun` はプロセス名 `p` のプロセスを定義するための関数である．

2 節で例示した鍵プロセス `Key` の Isabelle/LW-CSP-Prover による形式記述と遷移規則の導出例を図 10 に示す．最初の 2 行

[*3] `(Ev a)` の `a` の型は`'a` である

Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

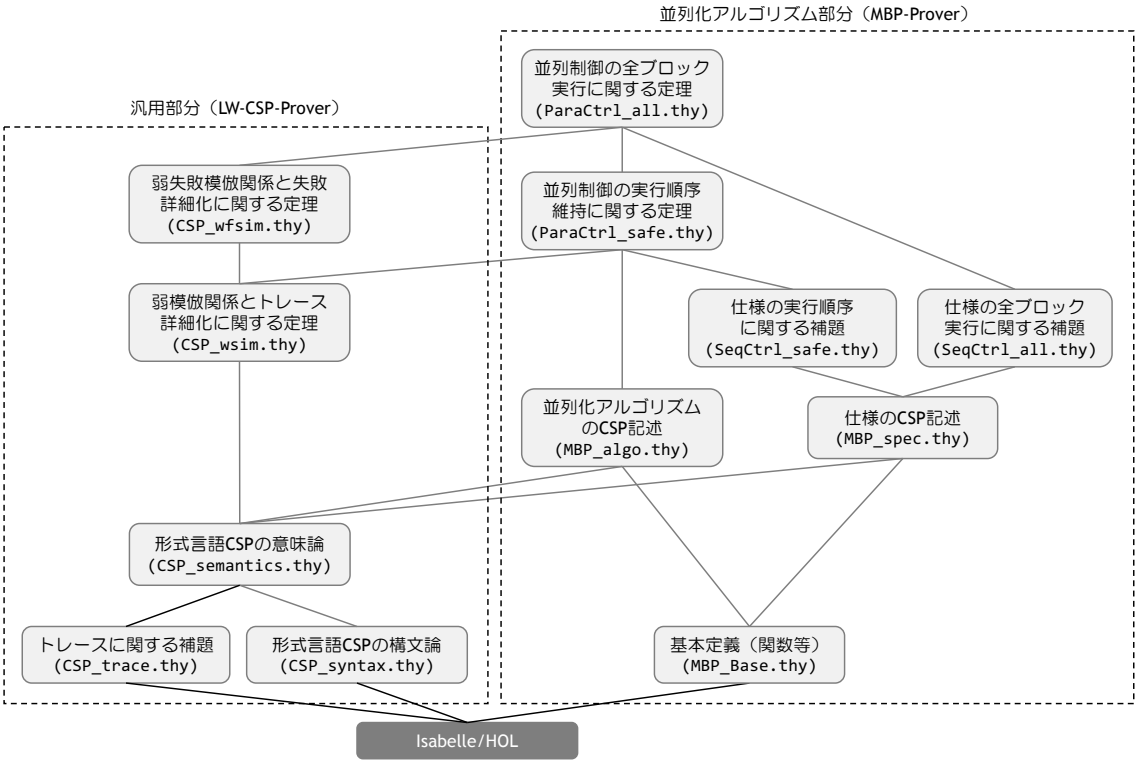

図 7 並列化アルゴリズムに関する理論ファイルの階層構造

```
datatype ('p,'a) proc
 = STOP
 | Prefix "'a" "('p,'a) proc" ("_ -> _")
 | ExtCh "('p,'a) proc" "('p,'a) proc" ("_ [+] _")
 ⋮
 | Hide "('p,'a) proc" "'a set" ("_ ~~ _")
 | PName "'p" ("$_")
```

図 8 Isabelle による CSP のプロセスの型 `('p,'a) Proc` の定義

```
inductive
 trn :: "('p,'a) proc ⇒ 'a event ⇒ ('p,'a) proc
        ⇒ bool" ("_ ---_---> _")
 where
   Prefix: "a -> P ---(Ev a)---> P"
 | ExtCh1: "P ---(Ev a)---> P'
           ⟹ P [+] Q ---(Ev a)---> P'"
 ⋮
 | Hide2:  "P ---(Ev a)---> P' ∧ a ∈ X
           ⟹ P ~~ X ---Tau---> P' ~~ X
 | PName:  "PNfun p ---e---> P'
           ⟹ $p ---e---> P'"
```

図 9 Isabelle による CSP のプロセスの遷移規則 `trn` の定義

で，`Key` で使用するイベントとプロセス名を，各々型 `K_Events` と `K_PName` として定義している．その次の **primrec** では，プロセス名 `Key` にプロセスを割り当てるための関数 `Def` を定義した後，**overloading** によって，`Def` を `PNfun` に多重定義している．これによって，プロセス`$Key` をプロセス `(lock -> unlock -> ···)` に展開できるようになる．

図 10 の **lemma** （*trn_exists* はこの補題の名前）は，これから証明する補題（"···"がゴールを表す）を宣言するための Isabelle のキーワード[*4]であり，**apply** は証明コマンドを与えるためのキーワードである．証明が完了（**done**）した補題や定理は，他の補題や定理の証明に適用できるようになる．

一つめの補題 *trn_exists* は，ゴールの遷移が存在することを証明するために，図 9 の推論規則 `PName`, `ExtCh1`, `Prefix` を順番に適用するための証明コマンドを与えている．`PName` に付属する `simp` は，（サブ）ゴールを簡単化する証明コマンドであり，ここでは，プロセス名の定義 `(PNfun Key)` を展開するために使われている．

二つめの補題 *if_trn_exists* では，遷移が存在すると仮定したときのイベントと遷移先についての条件を証明している．このような証明では，全ての推論規則を適用した場合を網羅的に確認する必要があるため，証明コマンド `(erule trn.cases)` が使われている．これは，全ての推論規則を仮定に対して適用する証明コマンドである．また，`auto` は Isabelle の半自動証明コマンドであり，集合や論理式等に関する証明を可能限り自動で実行する．`apply` の一番右の `+` は可能な限りこの証明コマンドを繰返し適用することを意味する．このような機械的な証明によって，複雑な CSP のプロセスについても，その遷移の正しさを厳密に確認することができ，証明の抜けや間違いを排除する

[*4] 重要なゴールの場合は **theorem**（定理）を用いるが機能は同じ

Vol.2022-ARC-248 No.54
Vol.2022-SLDM-198 No.54
Vol.2022-EMB-59 No.54
2022/3/11

```
datatype K_Events = lock | unlock | finish
datatype K_PName = Key

primrec
 Def :: "K_PName => (K_PName, K_Events) proc"
 where
 "Def Key =      (lock -> unlock -> $Key)
            [+] (finish -> STOP)"

overloading K_PNfun ==
 "PNfun :: K_PName => (K_PName, K_Events) proc"
 begin definition "PNfun = Def" end
 declare K_PNfun_def [simp]

lemma trn_exists:
 "$Key ---(Ev lock)---> (unlock -> $Key)"
  apply (rule PName, simp)
  apply (rule ExtCh1)
  apply (rule Prefix)
  done

lemma if_trn_exists:
 "$Key ---e---> P'
  ⟹    (e = (Ev in1) ∧ P' = unlock -> $Key)
      ∨ (e = (Ev finish) ∧ P' = STOP)"
  apply (erule trn.cases, auto)+
  done
```

図 10　鍵プロセス Key への Isabelle/LW-CSP-Prover の適用例

ことが可能になる．

意味論の理論ファイル（CSP_semantics.thy）には失敗詳細化や弱失敗模倣等が定義されており，4.2 節で説明した補題 4.1 の証明も理論ファイル（CSP_wfsim.thy）に与えられている．

Isabelle には証明支援エディタ（jedit）が付属しており，効率的に理論ファイル（証明コマンドの列）を作成できる．図 11 は，証明支援エディタで補題 4.2 を証明中（証明コマンド列の作成中）のスクリーンショットである．ここでは，有向グラフのノード数 k に関する帰納法を適用し，base-case を証明後，step-case を証明中の状態を表している．図 11 の上半分に補題（ゴール）と証明コマンドを入力すると，下半分にゴールに証明コマンドを適用した結果（サブゴール）が表示される．このサブゴールは，ノード数が k 以下の場合にこの補題が成立することを仮定して $k+1$ の場合に成立することを証明することを要求している．

5.2　並列化アルゴリズム部分（MBP-Prover）

図 7 の右側の理論ファイルには，我々の研究室で研究開発中のモデルベース並列化システム MBP の並列化アルゴリズムの実行順序維持と全ブロック実行に関する全ての定義，補題，定理

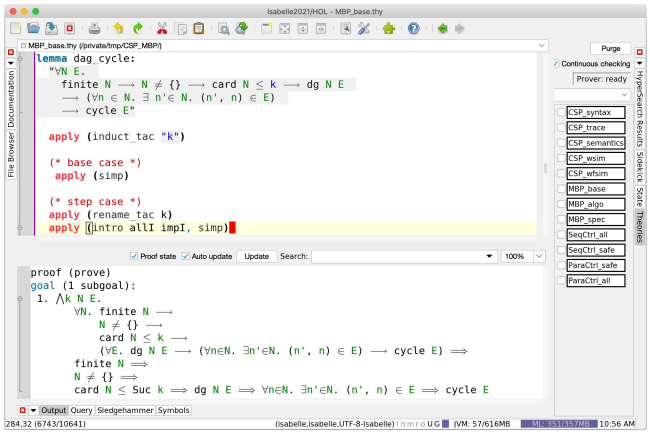

図 11　Isabelle の証明支援エディタ（jedit）

```
theorm SeqCtrl_refF_ParaCtrl:
 "Asn 1 ∩ Asn 2 = {} ⟹ Asn 1 ∪ Asn 2 = Blks
 ⟹ SeqCtrl Blks Deps [F= ParaCtrl Blks Deps Asn"
  apply (rule wfsim_refF[of "Sim Blks Deps Asn"])
  apply (simp add: Sim_wfsim)
  apply (simp add: Sim_def)
  done
```

図 12　並列制御モデルが仕様の失敗詳細化であることの証明

```
proof (prove)
goal (2 subgoals):
1. Asn 1 ∩ Asn 2 = {} ⟹ Asn 1 ∪ Asn 2 = Blks
   ⟹ weak_failures_sim (Sim Blks Deps Asn)
2. Asn 1 ∩ Asn 2 = {} ⟹ Asn 1 ∪ Asn 2 = Blks
   ⟹ (ParaCtrl Blks Deps Asn, SeqCtrl Blks Deps)
       ∈ Sim Blks Deps Asn
```

図 13　証明コマンドで wfsim_refF を適用後に表示されるサブゴール

が含まれている．例えば，並列制御モデルの CSP のプロセス ParaCtrl（図 4）とその仕様 Spec（図 5）は，各々，理論ファイル MBP_algo.thy と MBP_spec.thy に定義されており，並列制御モデル ParaCtrl の全ブロック実行に関する定理 4.6 と定理 4.7 は理論ファイル ParaCtrl_all.thy で証明されている．

例えば，定理 4.6 のゴールの記述と証明コマンドを図 12 に示す．図中の最初の証明コマンドは補題 4.5（wfsim_refF）を適用することを意味しており，この証明コマンドによって，図 13 に示す二つのサブゴールが表示される．図 13 の第 1 のサブゴールは，図 6 の関係 Sim が弱失敗模倣であることを要求し，第 2 のサブゴールは，組 (ParaCtrl, SeqCtrl) が関係 Sim に含まれることを要求している．第 1 のサブゴールの証明は補題 4.5（図 12 の 2 番目の証明コマンド Sim_wfsim）によって完了し，第 2 のゴールの証明は，集合 Sim の定義（Sim_def）を展開して完了する．

定理証明器 Isabelle では，帰納法や背理法などの証明法を適用可能な強力な証明支援ツールである．些細な補題も厳密に証

明する必要があるため，その人的な証明コストは非常に高いが，並列制御モデルや仕様，各詳細化関係に関する補題や定理を証明し，理論ファイル（LW-CSP-Prover, MBP-Prover）として蓄積することによって，今後，並列化アルゴリズムの機能を拡張したときに必要となる新しい定理の証明支援にも，多くの定義，補題，定理を再利用可能となる．

6. おわりに

本論文では，我々の研究室で研究開発中のモデルベース並列化システム MBP の並列化アルゴリズムの正しさを保証するため，階層構造やフィードバックをもたないなどの制限はあるが，任意の Simulink モデルについて，そのモデルを並列化した制御モデルが全ての制御ブロックを実行することを証明し（実行順序逆転が発生しないことは先行研究 [6] にて証明済み），その証明の正しさを定理証明器 Isabelle で形式的に検証した．今回の Isabelle による形式検証では，手作業で証明していた内容に深刻なミスを発見することはなかったが，いくつかのミスを修正することはできた．例えば，定理 4.6 を手作業で証明したときは，「(Blks, Deps) は有向非巡回グラフである」という仮定が必要と考えていたが，その仮定がなくても証明できることを Isabelle で検証できた．証明の検証のために作成した Isabelle の理論ファイル（LW-CSP-Prover, MBP-Prover）は，第三者による証明の正しさの確認や，今後の新しい定理の証明に利用することができる．

今回証明した並列化アルゴリズムは，モデルベース並列化システム MBP の並列化アルゴリズムの簡略版であり，フィードバックをもたない（ブロック間依存関係の有向グラフは閉路をもたない）Simulink モデルに制限している．フィードバックをもつ場合についても，1 周期分を抽出・展開すれば本研究の成果を適用可能であると想定しているが，マルチレートの場合などを扱うには不十分であり，入力対象とする Simulink モデルの拡張は今後の課題である．また，並列化アルゴリズムに関する新しい定理に対する証明支援能力を向上するため，さらに補題や定理を証明し，理論ファイル（LW-CSP-Prover, MBP-Prover）に登録していくことも今後の課題である．

参考文献

[1] MathWorks Makers of MATLAB and Simulink. 入手先 ⟨http://www.mathworks.co.jp.⟩
[2] 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.
[3] 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.
[4] 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.
[5] 于 文博, 磯部 祥尚, 枝廣 正人: 共有メモリ付階層型制御モデルの並列化アルゴリズムの CSP による形式化と FDR による検証, ETNET2020, 電子情報通信学会技術研究報告, Vol. 2020-EMB-44, No. 40, pp.1-12, 2020.
[6] 多門俊哉, 磯部祥尚, 枝廣正人: 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, ETNET2019, 電子情報通信学会技術研究報告, Vol.2019-EMB-50,No.9, pp.1-8,2019.
[7] University of Cambridge and Technische Universität München, Webpage on Isabelle. https://isabelle.in.tum.de/
[8] C.A.R.Hoare: Communicating Sequential Processes, Prentice Hall (1985).
[9] 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
[10] Y. Isobe and M. Roggenbach: Webpage on Csp-Prover. http://staff.aist.go.jp/y-isobe/CSP-Prover/CSP-Prover.html