Vol.2017-ARC-225 No.6
Vol.2017-SLDM-179 No.6
Vol.2017-EMB-44 No.6
2017/3/9

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

山本 尚平[1,a)]　鈴木 悠太[2]　峰田 憲一[2]　森 裕司[2]　枝廣 正人[1,b)]

概要：組込みシステムにおいて近い将来必要とされるマルチコア向けソフトウェアの開発手法として，我々はSimulink等を用いたモデルベース開発に着目したモデルベース並列化フローを提案している．しかし，モデルベース並列化フローでは逐次動作するプログラムを分割することで並列プログラムを構成するため，その過程でデッドロックや実行順序逆転などの並列化を起因とした問題が発生する可能性がある．そこで本研究では，モデルベース並列化フローの途中でCSPモデルを生成し，それを利用した形式検証を適用することで，並列化が原因で生じる問題の有無を検出する手法を提案する．

Applying formal verification using CSP Model to Model-Based Paralellizer

Shohei Yamamoto[1,a)]　Yuuta Suzuki[2]　Kenichi Mineda[2]　Hiroshi Mori[2]　Masato Edahiro[1,b)]

1. はじめに

組込みシステムの多機能化により，近い将来にマルチ・メニーコアを利用することが予測されている．しかし，一般に並行システムの開発は難しく，適切なタスク分割粒度とタスク配置が必要とされている．

そこで，我々は組込みシステムの開発現場において採用されているモデルベース開発に着目し，モデルベース並列化という並行システム開発フローを提案している．このフローでは，モデルベース開発に利用されるモデルからタスク同士の依存関係を自動抽出し，性能見積もりから適切な粒度で逐次コードを分割し．タスク配置を行うことで，システムの自動並列化が可能である．

しかし，このフローでは逐次実行の時には起きない並列化を起因としたデッドロックなどの不具合が発生する可能性がある．そのため，そのような問題が発生するかどうかを検証する必要があるが，マルチコアで動作する並行システムの多様な動作パターン全てを検証することは難しい．

1 名古屋大学大学院情報科学研究科
2 株式会社デンソー
a) yamasefu@ertl.jp
b) eda@ertl.jp

そこで本研究では, 形式検証と呼ばれる検証手法をモデルベース並列化フローに適用することで，並列化を起因とした問題の有無を特定する手法を提案する．この手法は，形式検証の特性によりシステムの動作パターンを網羅的に探索するため，人手の検証では難しいとされる並行システムの検証を自動で行うことが可能である．重要な前提は，本研究では並列化を起因とした問題のみに着目して検証するということである．逐次コードはモデルベース開発によって生成されたコードであり，そのソフトウェアは十分に検証され論理的に正しく動作して不具合がないことを前提として，その上で分割による並列化が原因となり起きる問題を検証する．つまり，並列動作が逐次動作の動作結果と同様になるよう動作するかどうかを検証することが本研究の検証目的である．本研究の前提の下では，検証のために並列動作の振る舞いのみに注目すればよく，各タスクの実際の演算処理を抽象化して動作パターンのみを検証するよう問題を単純化することで，検証コストを抑え実用的な時間での検証を実現する．

本研究で対象とするのはMATLAB/Simulinkを入力としたモデルベース並列化フローであり，形式検証に利用するのはCSPモデルと，それを扱う形式検証ツールPATで

ある．本研究ではモデルベース並列化フロー上で Simulink モデルの動作仕様を表現した CSP モデルと検証項目となる Assert 式を自動生成し，PAT 上での形式検証を行う．

2. 準備

2.1 MATLAB/Simulink

MATLAB/Simulink[1] は,MathWorks 社が開発しているソフトウェアである．Simulink モデルは，各処理を行うブロックと，それらのデータ伝播を表した信号線を組み合わせたブロック線図によって構成される．Simulink モデルはモデルとしてのモジュール管理や可視性向上の役目だけではなく，モデルに表現された演算を利用したシミュレーション機能も充実しており，これらの機能から，MATLAB/Simulink は現在モデルベース開発環境として実際の開発現場で広く利用されているツールである．また，MATLAB/Simulink は Simulink モデルから逐次コードを自動生成する機能も有し，我々が提案するモデルベース並列化フローではこの機能で生成した逐次コードを分割，再構築することで並列化を行っている．

2.2 モデルベース並列化

この節では，我々が提案しているモデルベース並列化 [2] について説明を行う．

BLXML

BLXML（Block-Level XML）[3] は XML 形式のファイルであり，モデルベース並列化フローにおいて Simulink モデルから生成される．BLXML の情報には，ブロック名，ブロックの種類，接続信号名，接続先などの，並列化対象の Simulink モデルについての基本的な情報が入っている他に，ブロックごとのコード情報，性能情報，コア割当て情報などを埋め込み，最終的な並列コード生成に利用される．本研究においても，BLXML 内のブロック接続関係やコア割当て情報などを元に CSP モデルを生成する．

並列化の流れ

(i) 入力とする Simulink モデルを BLXML へ変換する．変換ツールは，並列化向けにブロック名の変更やモデルの設定の変更を加えた後，モデルの構造を解析して BLXML を生成する．
(ii) MATLAB/Simulink の機能を利用し，逐次コードを生成する．
(iii) 自動で BLXML 内のブロックに対応するコード情報を生成逐次コードから切り出し，BLXML 内に埋め込む．
(iv) 自動で埋め込んだブロックごとのコード情報から SHIM[4] を用いた性能見積もり [5] を行い，ブロックごとの性能情報を BLXML に埋め込む．
(v) 自動で性能情報から負荷分散を考慮した並列化を行って割当てコア配置を決定し，コア割当て情報として BLXML に埋め込む．
(vi) 最後に，ブロックごとに決められた割当てコア配置より，対応するコードを結合して並列化コードを生成する．

2.3 CSP と PAT

CSP（Communicating Sequential Processes）[6] は Antony Hoare によって提唱されたプロセス代数であり，並行システムを形式的にモデリングするための仕様記述言語である．CSP モデルはプロセスとイベントによって構成され，チャネルや共有イベントなどで並行合成したプロセス同士を協調動作させることで，並行に動くシステムの仕様を表現する．

PAT（Process Analysis Toolkit）[7] はシンガポール国立大学において開発されたソフトウェアであり，CSP 理論をメインに据えたモデル検査ツールである．PAT は統合環境であり，PAT 上で CSP モデルの作成編集，検証，シミュレーションを行うことが可能である．

PAT では CSP モデルが書かれたファイルに，表明式と呼ばれる`#assert` から始まる式を記述することで検証を行う．PAT は，CSP の動作による全ての状態を網羅的に探索し，表明式に違反する状態を検出するとそこに至るまでの反例トレースを出力する．様々な表明式が用意されているが，本研究では deadlockfree，LTL，詳細化検証の 3 つを取り扱うので，それらを説明する．

- deadlockfree
  プロセス P がデッドロックフリーか検証する．
- LTL（線形時相論理）
  PAT では LTL と呼ばれる，時間関係を扱う表明式を書ける．「これ以降はずっと真」「いつかは真になる」といったような表明式を利用することが可能である．
- 詳細化検証
  CSP 特有の検証手法として，詳細化検証が存在する．これは，あるプロセスはあるプロセスの詳細化であるといった動作に関する包含関係を検証するものであり，トレース詳細化，安定失敗詳細化，失敗発散詳細化関係がそれぞれ存在する．本研究においては，トレース詳細化を扱う．

Timed-CSP

PAT は，本来の CSP 理論に時間要素を加えた Timed-CSP をサポートしている．PAT で記述できる Timed-CSP には通常の CSP 記法に加え，特有の記法を用いて時間に関する情報や制約を CSP モデルに導入する．以下に，本論文で扱う二つの記法を紹介する．

- Wait[t]
  t に指定した単位時間待つプロセスであり，イベントの直後等に挿入することでイベントにかかる時間を表現することが可能である．
- P deadline[t]

t 単位時間以内にプロセス P が終了することを強制するプロセスである．仮に何らかの理由で満たされない場合，失敗終了とみなされる．この項目は，デッドラインミス検証に利用される．

3. CSP モデルの生成

3.1 基本的な構造

CSP モデルにおいては，Simulink モデルのブロックに対応するプロセスをブロックプロセスと定義し，ブロックの配置コアごとに分割してブロックプロセスを逐次合成したプロセスをコアプロセスと定義する．

基本的なブロックプロセスはイベント部，入力チャネル部，出力チャネル部，駆動条件部から構成され，プロセスの動作が終了すると成功終了する．イベント部はブロック名と同じ名前のイベントが記述され，そのイベントが実行されるとそのブロックの演算が行われているとみなす．入力チャネル部と出力チャネル部は，コア割当ての結果ブロック間のデータ送信にコア間通信が必要となった時，それに対応するよう生成される．つまり，コア間通信が存在しない時はそれに対応するチャネル通信も CSP モデル上には存在しない．入力チャネル部はイベント部の直前に配置され，必要なデータが揃ってからブロックの演算を表すイベントを実行するように動作する．出力チャネル部はイベント部の直後に配置され，動作が終わってからそのデータを必要とする後続のプロセスに対してデータを送るよう動作する．駆動条件部は，EnabledSubsystem などの条件付きサブシステムの内部に存在するブロックに対応するプロセスに現れ，プロセスの実行非実行を条件変数によって制御するように記述される．駆動条件が成立した時に入力チャネル部，イベント部，出力チャネル部を実行するよう記述し，不成立の時にはそのブロックが不成立となったことを表すイベントが発生するように記述する．

コアプロセスは，ブロックプロセスを逐次合成し，並列動作の各コアの動作を表す．つまり，コアプロセスは割り当てコアの種類と同じ数だけ生成され，チャネル通信を行うときは別々のコアプロセス内ブロックプロセス同士で行われることとなる．

最後に，コアプロセス全てを並行合成した System プロセスを定義する．このプロセスが，並行システム全体を表すプロセスとなる．

3.2 遅延ブロックの変換

Simulink モデルのライブラリブロックには，Delay や UnitDelay といったデータを保持して一定周期遅延させる遅延ブロックが存在する．遅延ブロックの挙動は，Simulink モデルの実行周期が始まった時に，まず自分が保持しているデータを出力し，周期終盤にアップデート処理として今回値を受け取り保持データを更新する．このブロックを CSP のブロックプロセスとして表すとき，3.1 節で表した基本の構造では，実際の挙動を表すことができない．

そこで，今回作成する CSP モデルでは，遅延ブロックの出力処理とアップデート処理を別プロセスとして区別し，出力処理は BLXML 内実行順情報に従った位置，アップデート処理はその遅延ブロックが配置されているコアプロセスの末尾に付加するといった形で表現している．BLXML 内実行順序情報は，遅延ブロックの出力処理に対しての順序関係を表すものであるので，このブロックプロセス記述で遅延ブロックの動作を表すことが可能である．

3.3 条件付きサブシステムの変換

Simulink には EnabledSubsystem,TriggeredSubsystem などの信号により実行/非実行を制御する条件付きサブシステムが用意されている．これらのサブシステムは，その内部ブロックがある周期において実行される場合とそうでない場合があるため，3.1 節で示した基本構造のうち駆動条件部を設定し，条件変数を参照してその実行非実行を制御する必要がある．

この条件変数を分岐させるために，条件付きサブシステムの特殊ポートブロック（EnablePort ブロック，TriggerPort ブロックなど）のブロックプロセスでそれを表現する必要がある．これらのブロックプロセスでは，イベント部において内部選択を用いて非決定的に条件変数値を設定することで，その条件変数を駆動条件とするその他のブロックの動作を制御する．条件変数値は-1 で初期化されており，0 となったら非実行，1 となったら実行を表す．

さらに，SwitchCase ブロックと If ブロックを用いた時には条件変数の扱い方が異なる．これは，上記のような条件変数の値は 0 か 1 の二択では表すことができず，分岐の数だけの値を取る必要があり，CSP モデル生成ツールは内部でその分岐一つ一つに対し対応番号を割り振る．例えば，SwitchCase ブロックの出力が 3 つあった場合，条件変数の取りうる値は 0,1,2 で制御を行うことになり，接続先の ActionSubsystem は条件変数が自身に紐付いている対応番号を取るときに動作するように条件が設定される（このときの条件変数の 0 への遷移は非実行ではなく，ひとつの駆動対応番号となる）．この時，Merge ブロックへの通信があった場合にそれは動作した ActionSubsystem の出力のみを受け取るように択一的に動作しなければならない．このため，CSP モデルではそのような通信も条件変数に応じ択一的に動作するように工夫している．

3.4 時間情報付きプロセス

本研究においては，通常の CSP モデルに加えて Timed-CSP を利用した時間情報付き CSP モデルも出力することができる．時間情報はモデルベース並列化フローにおいて予め見積もった worst,best,typical のサイクル値を持つ

performance 情報を利用し，それらの情報を Wait プロセスとして CSP 内に展開する．ブロックの種類や具体的な処理にもよるが，BLXML 内の performance 情報はおおよそ数百サイクルの値を持っている．この値をそのまま CSP に埋め込むと，単位時間経過イベントが多数発生し状態数の増加へとつながるので，今回は暫定的に 50cycle を 1 単位として丸めることとしている．

CSP モデル内では，ブロックプロセスのイベント実行を表すプロセスの後に Wait プロセスを挿入することで実行時間を表現する．この時，BLXML 内の performance 情報のうち best 値と worst 値を用いて，実行時間の揺れを表現している．best 値と worst 値を内部選択を用いて非決定的に選択することで，あるブロックが最も良い効率で動いた場合と最も悪い効率で動いた場合の動作パターンを両方表現することができる．

4. 検証項目と検証方法

4.1 検証が必要な問題

我々が提案しているモデルベース並列化フローでは，逐次コードを生成した後それを分割，再構成して並列化を行うため，生成した並列コードには逐次実行時と動作が異なるような並列化を原因とする問題が発生する可能性がある．本研究では，ソフトウェアの論理的正当性は保証されていて逐次動作には問題がないという前提を置き，それが崩れるような並列動作の振る舞いが起きるかどうかを検証する．本研究で検証の対象としているそのような 3 つの問題について，ここで説明する．

- デッドロック
  デッドロックは，共有リソースの排他制御や同期のための通信待ちが競合することによりシステムが停止してしまう問題である．
- 実行順序逆転
  並列化を行うと，各コアでは同期通信を必要に応じて取りながらも，独立して処理を行う．この際，逐次実行時には確定していた各処理の順序が，並列化により一部入れ替わる可能性がある．モデルベース並列化フローにおいては，入れ替わることで問題が起こることがないように Simulink モデルの信号線をタスク間の依存関係として並列化を行っている．しかし，DataStoreMemory を利用したモデルにおいては，DataStoreRead と DataStoreWrite によるメモリの読み書きの順序が依存として現れることがない．結果，逐次実行においては保証されていた順序が並列化により崩れてしまい，読み書き順序の逆転によってモデルの動作が不正となってしまう場合がある．
- デッドラインミス
  通信の同期待ちや通信自体のオーバーヘッドなどの要因により，実際の性能向上幅の予測は難しいものとなっている．並列化の実行パターンは多く，プログラムにおけるクリティカルパスの静的な見積もりは難しいが，Timed-CSP を用いた形式検証を利用することでデッドラインミスに関して検証可能である．

4.2 問題に対する検証方法

デッドロックの検証

デッドロックは，CSP においてこれ以上状態遷移ができない状態に陥るか，プロセスが Stop（失敗終了）したときに検出される．PAT においては，deadlockfree にてプロセスがデッドロックしないかどうかの表明式を作成することができるため，これによってデッドロック検証が可能である．

実行順序逆転の検証

実行順序逆転の検証は，LTL（線形時相論理）式を用いて表明式を作成する．今回，順序逆転の検証は DataStoreRead,DataStoreWrite の本来の実行順が逆転しないかどうかを検証する．DataStoreRead と DataStoreWrite の本来の実行順は BLXML 内の実行順情報から得ることができ，ひとつの DataStoreMemory ブロックに紐付けられる DataStoreRead もしくは DataStoreWrite の順序の連続する二つの順序についての表明式を生成し，それを複数作成することにより全体の順序を検証する．例えば，DataStoreRead ブロック α,γ と，DataStoreWrite ブロック β があって，それら全てが同一の DataStoreMemory に紐づくブロックであり，本来の実行順が $\alpha \to \beta \to \gamma$ であった時，LTL 表明式は $\alpha \to \beta$ と $\beta \to \gamma$ の二つが生成され，これらが全て Valid となった時に検証した並行システムには順序逆転の可能性がないことがわかる．

上述の例の $\alpha \to \beta$ の順序逆転検証のための実際の表明式は以下のようになる．

$$\#assert\ System() | = (!\beta U \alpha) \&\& <> \beta$$

$(!\beta U \alpha)$ は「α が発生するまで β が発生しない」ことを表し，$<> \beta$ は「いつかは β が発生する」ことを表す．この二項が同時に成り立つとき，$\alpha \to \beta$ の実行順序が守られることが保証される．

デッドラインミスの検証

デッドラインミス検証は Timed-CSP を用いて行われる．2.3 節で述べた時間に関する演算子のうち，並行システムを表すプロセスシステムに deadline 制約をかけたプロセスを用意する．deadline 制約はプロセスが指定時間内に終了しなかった場合失敗終了するため，$deadlockfree$ により検証が可能である．

5. 階層分割

形式手法には大きな課題として状態爆発と呼ばれる問題が存在する．この問題に対し，これまでに数々の状態数圧

縮手法が先行研究でなされている [8].

本研究では，状態爆発への対策として階層分割を提案する．階層分割は，Simulink モデルの AtomicSubsystem の内外を別の CSP モデルとして独立させることで検証を分け，状態爆発を防ぐ．逐次実行においては，AtomicSubsystem 内部の処理が実行されている間に他の処理が割り込むことはない．この特性を利用し，並列分散されたタスクのうち AtomicSubsystem 内部のまとまった部分をまとめて別の CSP として切り出すことで分割を行う．この分割は AtomicSubsystem の階層構造に対して行うため，同一のモデルに再帰的に適用することで適切なサイズまで分割を行うことが可能である．

分割手法は，AtomicSubsystem 内部のブロックを全て別の CSP モデルとして生成し，外側のモデルに対しては抽象化したダミープロセスを配置することで，外側にあるプロセスの動作のみに着目した際に元のモデルと動作が不変であるようにする．

不変であるかの検査方法は，元のモデルと分割後の外側モデルのうち互いに存在するイベント以外を全て CSP の隠蔽を利用して観測できないようにした後，二つのモデルの動作トレースが同一であるかを双方向のトレース詳細化検証によりトレース等価性を検査する．これを示すことにより，外側モデルでの検証は元のモデルの同一部分に対する検証と同様のトレースを検証することができ，また内側モデルも独立して検証することで状態爆発への回避策となる．

この分割は，現状デッドロック検証にのみ対応している．順序逆転検証については，片方が外側モデルに，片方が内側モデルに入っている場合に元のモデルで検証することと同義の検証を行う方法を検討中である．デッドラインミス検証については，内側モデルにてかかる時間を外側モデルに反映させる必要があり，その方法の考案についても今後の課題としている．

6. 評価

6.1 デッドロックと順序逆転

この節では，実際のモデルへの本研究の提案手法の適用による問題の発見と，その解決策の評価を行う．まず，MathWorks 社 MATLAB/Simulink ドキュメンテーション内に存在する DataStoreMemory を利用したモデル [9] を参考に，同様の動作となるサンプルモデルを用意した．

このモデルは，入力に sin 波形を入力し，入力値が 0.8 以下の場合モデル上部のサブシステム内で入力値を 1.5 倍した値が出力されるが，0.8 を超えた場合は出力値がモデル下部のサブシステム内で 0.5 倍になり出力されるといった処理になっている．そして，上下のサブシステムどちらの値を使うかと言った制御を，一度 DataStoreWrite によってメモリに boolean 値を書き込み，両サブシステムの接続先である Switch ブロックの制御端子につながった DataStoreRead からその boolean 値を入力することで出力する値を切り替えている．

このサンプルモデルに対し，モデルベース並列化フローを通して並列化コードを生成し，また通常の CSP モデルと時間情報付き CSP モデルを作成した．二つの CSP モデルについて，デッドロック検証と順序逆転検証を行ったところ，デッドロックは検出されなかったが，両方のモデルにおいて順序逆転が検出された．実際に逐次コードと並列化コードの動作結果を比較したものが表 1 である．

表 1 逐次動作と並列動作の結果とその比較 (一部抜粋)
Table 1 Result and comparison of the serial execution and the parallel execution (Excerpt)

時間 (s)	逐次動作の出力	並列動作の出力	比較
2.0	0.454	0.454	1
2.1	0.431	0.431	1
2.2	0.404	0.404	1
2.3	1.118	0.372	0
2.4	1.013	1.013	1

表 1 を見ると，時間が 2.3 秒の時に出力されている値が逐次動作と並列動作で異なっていることがわかる．この原因は，逐次動作とは異なり，並列動作では DataStoreRead が DataStoreWrite よりも先に実行してしまっているためである．本来は 2.3 秒時点では出力が下のサブシステムの計算結果から上のサブシステムの計算結果に移るのが正しい動作だが，本来と異なる下のサブシステムの計算結果を利用してしまうため，逐次動作と並列動作の結果が不整合となる問題が生じてしまう．

なお，この問題に対する解決策は以下の 2 点が考えられるが，いずれの解決策も対応後の正常動作を確認した．

(i) DataStoreWrite と DataStoreRead を同じコアに割り当てる．
(ii) DataStoreRead 実行前に一定時間処理を待たせてから実行させる．

これらの結果より，提案手法が並行動作の問題を発見し，また解決策を適用したものが安全であることが確認できたことで，並行システムに対する検証として有用なものであることが示せた．

6.2 階層分割の適用

本節では，5 章で説明した階層分割を実際に適用した例について述べる．フローを適用した Simulink モデルは約 10000 ブロックから構成されるエンジン制御の一部を表したモデルである．このモデルに対し提案手法を用いたところ，3370 個のブロックプロセスを含んだ CSP モデルが生成され，デッドロック検証は完了することなくリソース不足で PAT が停止した．この CSP モデルに対して，階層分

表 2　階層分割適用結果 (一部抜粋)
Table 2　Result of applying hierarchical division (Excerpt)

CSP	ブロックプロセス数	検証にかかる時間 (s)
元のモデル	3370	検証不可 (1 時間で打ち切り)
分割 1	28	0.01
分割 2	27	0.01
分割 3	66	0.07
分割 4	27	0.01
分割 5	272	55
⋮	⋮	⋮
分割 34	27	0.01
合計	3440	528.79

割を適用して 34 個の CSP モデルに分割を行い，それぞれに対してデッドロック検証を行った．その結果のうち，一部抜粋したものを表 2 にまとめる．

表 2 では，34 個の分割された CSP モデルのうち一部を抜粋して掲示している．分割後の CSP モデルでは最も多いもので 281 個のブロックプロセスを含んだモデルができあがり，それらは約 1 分程度での検証が可能となる．分割された 34 個のモデルすべてにおいてデッドロック検証を行い，その全てでデッドロックフリーであることが確認できたため，このモデルに対する並列化においてデッドロック問題は発生しないということが示せた．このようにして，そのままでは状態爆発により検証できないモデルも階層分割で適切なサイズまで分割して検証することで，実用的な時間での検証が可能であることを示せた．

なお，現在分割は手動指定によって行っており，今回の分割は全て同程度の階層にある AtomicSubsystem を指定した分割となっている．そのため分割後のプロセス数にばらつきがあるが，このばらつきを均等化するような分割方法を自動的に決定する手法を考案することは，今後の課題となっている．

7. おわりに

7.1 まとめ

本研究においての成果は，モデルベース並列化フローの中で並行システムの動作を表した CSP モデルを自動生成し，またその CSP モデルを利用した検証により並列化を起因とした問題の特定が自動で行うことができることである．並列化を起因とした問題として，デッドロック，実行順序逆転，デッドラインミスを挙げ，それぞれに対する CSP と PAT を利用した検証方法を提案し，状態爆発への対策として階層分割の提案も行った．さらに，モデルベース並列化フローを適用した時に問題を発見し，それが実際の並列コードでも再現される問題であることを確認して，提案手法が並列化による問題を発見するために有用であることを示した．

7.2 今後の課題

階層分割適用時の検証

5 章で述べたとおり，現状階層分割を適用した際にはデッドロック検証のみにしか対応できていない．残る二つの項目に対し，階層分割へ工夫を加え，分割の正しさの検査を追加で行う必要があると考える．

自動階層分割

階層分割は現状手動で分割 AtomicSubsystem を指定しているが，適切なサイズの分割を行えるように予めモデルを確認して分割対象を決めておく必要がある．今後においては，CSP モデルのブロックプロセス数などの情報に対する検証時間の相関をまとめ，並列化フローのユーザが階層分割の指定などを気にすることなく自動で適切なサイズへの分割を行えるよう分割 AtomicSubsystem を選択できるようにすることが望ましい．

マルチレートモデルへの対応

今回の提案手法で生成している CSP モデルは，全てモデルベース並列化フローへの入力 Simulink モデルがシングルレートで動作していることを前提としている．Simulink モデルには，複数のレートで動作するマルチレートモデルが存在し，それらを表現するためには現在の CSP モデル生成方法では不十分である．よって，マルチレートモデルへの対応を行うために，レートによる周期あたりのブロックの実行回数の変化や，レート変換ブロックへの対応，そしてマルチレート動作に対する検証方法の考案などを行っていくことが必要である．

参考文献

[1] MATLAB. *version 8.3.0.532 (R2014a)*. The MathWorks Inc., Natick, Massachusetts, 2014.
[2] 山口滉平他. Simulink モデルからのブロックレベル並列化. 組込みシステムシンポジウム 2015 論文集, 第 2015 巻, pp. 123–124, oct 2015.
[3] 山口滉平. モデルベース開発におけるブロックレベル並列化のためのデータ形式. Master's thesis, Jan 2016.
[4] Masaki Gondo, et al. Establishing a standard interface between multi-manycore and software tools-shim. In *COOL Chips XVII, 2014 IEEE*, pp. 1–3. IEEE, 2014.
[5] 溝口裕哉. ソフトウェア向けハードウェア性能記述を用いたプロセッサ性能見積手法に関する研究. Master's thesis, Jan 2016.
[6] Charles Antony Richard Hoare and He Jifeng. *Unifying theories of programming*, Vol. 14. Prentice Hall Englewood Cliffs, 1998.
[7] PAT. `http://www.comp.nus.edu.sg/~pat/`.
[8] Andrew W Roscoe, et al. Hierarchical compression for model-checking csp or how to check 1020 dining philosophers for deadlock. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 133–152. Springer, 1995.
[9] MathWorks. データストアの作成によるグローバルデータのモデル化. `https://jp.mathworks.com/help/simulink/ug/model-global-data-using-data-stores.html`.