# Gears OSでモデル検査を実現する手法について

東恩納　琢偉[1,a)]　奥田　光希[2,b)]　河野　真治[3,c)]

**概要**：GeasOS は CbC で記述されており処理単位である codeGear の間に自由にメタ計算をはさむことができる。ここに dataGear の状態を記録することにより、ユーザプロセスあるいはカーネルそのもののモデル検査が可能になる。一般的なモデル検査では状態数の爆発は避けられない。記録する状態を抽象化あるいは限定する手法について考察する。

## 1. 並列プログラムの信頼性

現在、CPU の処理性能はクロック数の向上は電力消費の増大の問題から伸び悩んでおり、マルチコア CPU や GPU を利用した並列化処理を行うことによって処理速度の向上を図る事が多く、また画像処理や機械学習の分野では並列化処理は重要な役割を果たしている。しかし処理を並列化する場合、個々のプログラムが正しく動作する事が証明されていてもそれらを並列に実行したとき、全体の動作の正しさは保証されない。これは並列化されたプログラムの非決定性によるものである。また非決定性を含むプログラムは、逐次型のプログラムに有効な二分法などによるうデバック手法ではデバックする事が困難である。そのため、非決定生を含むプログラムに対して有効なデバック手法や検証手法の確立が重要な課題となっている。本研究ではモデル検査を用いる事でプログラムの信頼性を保証する手法として、GearsOS におけるモデル検査手法について提案する。

モデル検査とはプログラムのの設計から導出されたモデルが形式仕様を満たすかを検証することで信頼性を保証するもので、全ての状態を数えあげ、その状態について仕様が常に死んであることを確認する事で保証される。、しかしプログラムの規模が大きくなると導出されるモデルの状態数が爆発的に増えるためにそれら全てを検証する手法は好ましくない、そのため記録する状態を抽象化、または限定することによって検証の計算を減らす方法について考える。

[1] 琉球大学大学院理工学研究科情報工学専攻
[2] 琉球大学工学部工学科知能情報コース
[3] 琉球大学工学部
[a)] ikkun@cr.ie.u-ryukyu.ac.jp
[b)] Koki.okuda@cr.ie.u-ryukyu.ac.jp
[c)] kono@ie.u-ryukyu.ac.jp

## 2. 既存のモデル検査手法

モデル検査の方法としてよく利用される物として、SPIN と java path finder(以下 JVM) というツールがある。

SPIN は Promela という仕様記述言語で記述する事で C 言語の検証器を生成する事で、コンパイルまたは実行時に検証する事ができる。チャネルを使っての通信や並列動作する有限オートマトンのモデル検査が可能である。
SPIN では以下の性質を検査する事ができる。

- アサーション
- デッドロック
- 到達生
- 進行性
- 線形時相論理で記述された仕様

Java Path Finder(JPF) は java プログラムに対するモデル検査ツールで、java バーチャルマシン (JVM) を直接シミュレーションして実行している。そのため、java のバイトコードを直接実行可能である。バイトコードを状態遷移モデルとして扱い、実行時に遷移し得る状態を網羅的に検査する。バイトコードの実行パターンを網羅的に調べるために、膨大な CPU 時間を必要とする。また JVM へースであるため、複数のプロセスの取り扱いが出来ず、状態空間が巨大になる場合は直接実行は出来ず、一部を抜き出してデバックをするのに使用される。
JPF では以下の事ができる。

- スレッドの可能な実行全てを調べる
- デッドロックの検出
- アサーション
- Partial Order Reduction


図 1　CodeSegment と DataSegment

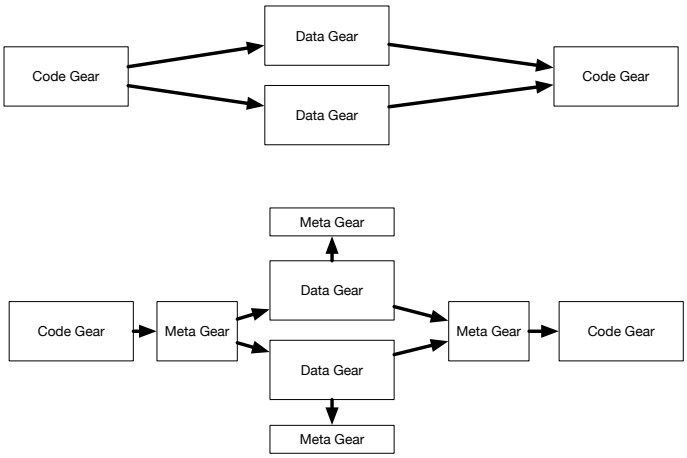


図 2　Gears OS のメタ計算

## 3. Continuation based C

GearsOS は Continuation based C (以下 CbC) という言語を用いて拡張性と信頼性を両立させることを目的として本研究室で開発されている。CbC は C 言語と似た構文を持つ言語であるが、CodeSegment と DataSegment を用いるプログラミングスタイルを提案している。CodeSegment は処理の単位である。CodeSegument は値を入力として受け取り処理を行ったあと出力を行う、また他の CodeSegment を接続していくことによりプログラムを構築していく。DataSegment は CodeSegment が扱うデータの単位であり、処理に必要なデータが全て入っている。DataSegmen は Input DataSegment と呼ばれ、出力は Output DataSegment と呼ばれる。CodeSegment A と CodeSegment B を接続したとき、A の Output DataSegment は B の入力 Input DataSegment となる。

CodeSegment の接続処理はメタ計算として定義されており、実装や環境によって切り替えを行なうことができる。検証を行なうメタ計算を定義することにより、CodeSegment の定義を検証用に変更せずプログラムの検証を行なう。

CbC における接続は goto を用いて行われる。got は関数呼び出しのような環境変数を持たず goto の直後に遷移先を記述することで、遷移先に接続される。これを軽量継続と言い、遷移元の処理に囚われず、遷移先を自由に変更する事が可能でり 遷移元の code gear の goto 先以外に変更する事なく、処理の間にメタレベルの計算を挿入する事が可能である。CbC における遷移記述はそのまま状態遷移記述にすることができる。**??**

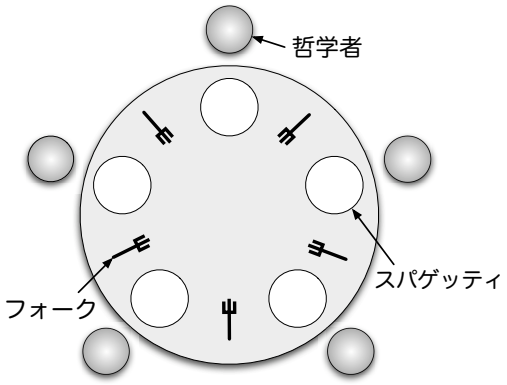


図 3　Dining Philosohers Ploblem

GearsOS では CodeSegment と DataSegment はそれぞれ CodeGear と DataGear と呼ばれている。マルチコア CPU 環境では CodeGaer と CodeSegment は同一だが、、GPU 環境では CodeGear には OpenCL/CUDA における kernel も含まれる。kernel とは GPU で実行される関数のことである。

## 4. DPP

検証用のサンプルプログラムとして Dining Philosohers Ploblem (以下 DPP) を用いる。これは資源共有問題の 1 つで、次のような内容である。

5 人の哲学者が円卓についており、各々スパゲッティーの皿が目の前に用意されている。スパゲッィーはとても絡まっているので食べるには 2 本のフォークを使わないと食べれない。しかしフォークはお皿の間に一本ずつおいてあるので、円卓にフォークが 5 本しか用意されていない。図 **??**哲学者は思索と食事を交互に繰り返している。空腹を覚えると、左右のオークを手に取ろうと試み、2 本のフォークを取ることに成功するとしばし食事をし、しばらくするとフォークを置いて思索に戻る。隣の哲学者が食事中でフォークが手に取れない場合は、そのままフォークが置かれるのを待つ。

各哲学者を 1 つのプロセスとすると、この問題では 5 個のプロセスが並列に動くことになり、全員が 1 本ずつフォークを持って場合はデッドロックしていることになる。プロセスの並列実行はスケジューラによって制御することで実現する。

## 5. タブロー展開と状態数の抽象化

GearsOS におけるモデル検査はタブロー展開を用いることでデッドロックを調べる。タブロー法は生成可能な状態の全てを生成する手法である。反例を探す場合は反例が見つかった時点で状態の生成を停止してもよいが、証明

を行う場合は全ての状態を生成する必要がある。状態の生成は初期状態から非決定的に生成される全ての次の状態を生成することにより行われ、これを状態の展開という。証明はプログラムの状態の数に比例し、またプログラムが含む変数の数の指数状の計算量がかかる。この展開の際に仕様うも同時に展開することでプログラムに対する仕様の検証を行う事が可能である。

タブロー法は実行可能な状態の組み合わせを深さ優先探索で調べ、木構造で保存する方法である。この時、同じ状態の組み合わせがあれば共有することで状態を抽象化する事で、状態数が増えすぎる事を抑える。

## 6. GearsOS を用いたモデル検査

DPP は哲学者 5 人が同時に行動するので、5 つのスレッドで同時に処理することで状態を生成する事ができる。まず Gears OS の並列構文の par goto が用いることでマルチスレッド処理の実装を行う。 par goto は引数として、data gaer と実行後に継続する`__exit` を渡す。par goto で生成された Task は`__exit` に継続する事で終了する。これにより Gears OS は複数スレッドでの実行を行う事が可能である。　また Gears OS には　Synchronized Queue というマルチスレッドでのデータの一貫性を保証する事ができる Queue があり、これを使い 5 つのフォークの状態を管理する。　Syncrhonized Queue は CAS(Check and Set) を用いて実装されており、値の比較、更新をアトミックに行う命令である。CAS を使う際は更新前の値と更新後の値を渡し、渡された更新前の値を実際に保存されているメモリ番地の値と比較し、同じデータ今日がないため、データの更新に成功する。異なる場合は他の書き込みがあったとみなされ、値の更新に失敗し、もう一度 CAS を行う。
5 スレッドで行われる処理の状態は以下の 6 通りで、think のあと Pickup Right fork に戻ってくる。

- Pickup Right fork
- Pickup Left fork
- eating
- Put Right fork
- Put Left fork
- Thinking

この状態は goto next によって遷移する。またこの状態遷移は無限ループするので MemoryTree に保管し、保管されている状態とは stat DB によって保管される

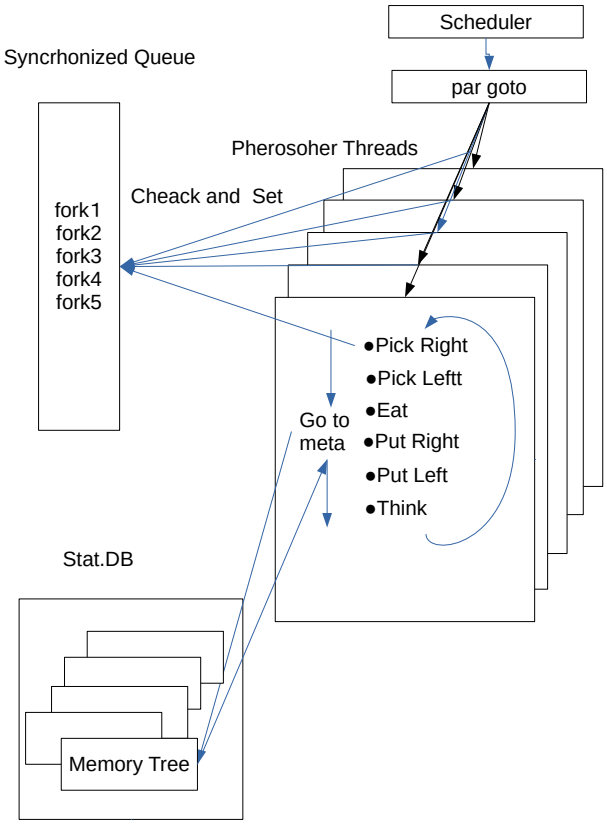


図 4　DPP chacking