並行システム設計技術を利用した衝突防止システムの研究

～コース形状の変更の影響を受けにくいモデルの開発～

橋浦研究室　1125426 水沼 祥太

1. はじめに

先行研究[]において並行システム設計技術を用いて, 検証可能なソフトウェアの開発を行うことによってこのような問題を未然に防ぐことが可能であることを証明している.

しかし, 実装したプログラムではあらゆるコースに対応しているとは言い切れず, 様々なコースに応じて検証と実装をしなくてはならない.

2. 研究目的

先行研究において一つの形のコースを使用して検証, 実装を行ったがそれだけでは様々なコースにおいても問題を未然に防ぐことが可能かを示せたわけではない.

そこで本研究では先行研究をもとにコース形状を変更し, それに伴って検証やモデルへの影響を調査する.

3. 研究内容

本研究では, 先行研究同様にライントレースロボットを対象とし, CSP でシステムのモデルを構築し, ツールによって欠陥がないことを検証した後で, 実際に動作するプログラムは世の中で広く用いられている Java 言語で実装することとした.

3.1. 並行プログラミングの難しさ

並行プログラミングにおける特徴として, 実行順序に非決定性が存在することが挙げられる. 非決定性とは, あるステートマシンを考えたときに, ある状態から同じイベントによって複数の状態へ遷移を持つ場合や, イベントが無くても遷移してしまうような遷移を持つ場合に該当する（図 1）.

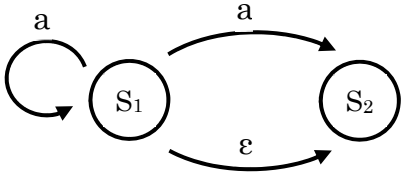

図 1. 非決定性の例

3.2. 並行プログラミングにおけるバグの例

並行プログラミングにおけるバグの例として, 変数の値が壊れてしまう場合について, 例を挙げて説明する.

例えば, 変数 X に 1 を足すプログラムがあるとする. このようなプロセスが 2 つあり, プロセス間で変数 X が共有されているとする.

このような場合, 2 つのプロセスが直列に実行されるならば, 変数 X は 1 が 2 回足されるため, プロセスの終了時には変数 X の値は 2 になることが期待される. しかしながら, 並列実行時には図 2 で示したような実行順序となることがある. このような場合には, 2 つのプロセスが変数 X から 0 を読み込むことになり, それぞれが 1 を足して書き込むため, 実際に得られる値は 1 となる. これは期待結果と一致しない.

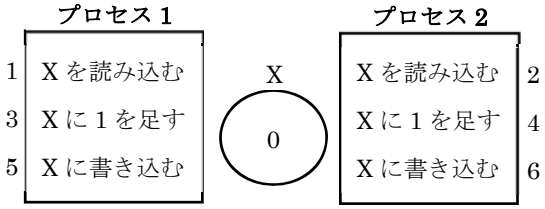

図 2. 変数の値の破壊

3.3. 検証とは

並行プログラム実行時に発生する非決定性によって生じる欠陥を取り除くために, 検証を用いる. 検証を用いることで, プログラムを実行しなくても欠陥を発見することが可能である. 同時に反例を得ることができるため, 反例に基づいて実装前のモデルから欠陥の除去が可能になる.

検証する際には, システムが満たすべき性質をあらかじめ表明として記述しておき, 検証ツールを用いて表明を満足するかどうかを判定することができる.

3.4. CSP について

CSP(Communicating Sequential Processes)とは, 1978 年に Hoare[1]によって考案された形式手法の一つで, 並行システムを形式的に記述し,

解析する理論である．

表現力や検証方法の理論研究が進められるとともに，ソフトウェア（マルチスレッドコネクションプール），組み込みシステム（航空電子システム），セキュリティ（認証プロトコル）等，幅広い分野の検証に使われている．

```
CTtest(x) = CT \ diff(Events,{|move.0.x,move.1.x,

CTspec(0) = move.0.0 -> put.0 -> CTspec'(0)
CTspec(2) = move.1.2 -> put.2 -> CTspec'(2)
CTspec(x) = |~| i:Id @ Cspec(i,x) ; CTspec(x)
CTspec'(x) = |~| i:Id @ Cspec(i,x) ; CTspec'(x)
Cspec(i,x) = get.x -> move.i.x -> (move.i.x -> pu

assert  CTspec(0) [T=   CTtest(0)
assert  CTspec(1) [T=   CTtest(1)
assert  CTspec(2) [T=   CTtest(2)
assert  CTspec(3) [T=   CTtest(3)
assert  CTspec(4) [T=   CTtest(4)
```

図 X. CSP 記述（検証部分を抜粋）

3.5. JCSP[2]

JCSP は Java 言語で CSP モデルの実装を支援するために Welch と Brown により開発されたライブラリである．

JCSP には並行動作するプロセスや同期型の通信チャネルなど CSP モデルを実装するためのクラスが豊富に提供されている．

3.6. モデルと変更点

本研究では図 4 に示すコースを基にモデルを構築した．コース上にはチェックポイント（図 4 コース上の点部分）があり，コースをいくつかの区間に区切っている．

走行体の衝突を避けるためには 1 つの区間に 2 台以上の走行体が進入しないようにしなければならない．このため，チェックポイントに信号を置き，走行体がチェックポイントに到達するまでに走行体直前の信号と通信し，進入可能ならば通行し，信号が進入不可の信号を出している場合には停止して進入可能の信号が出るまで待つこととした．

また，変更点としてコース形状を先行研究の長円形から 8 の字に変更し，先行研究にはない交差点を設け．これを通過しなければならないように変更し，交差点における信号待ちが実装可能かどうかを確認する．

4. 実装

走行体はレゴマインドストーム EV3[3]を 2 台使用し，前述の JCSP で実装したプログラムを搭載するためにファームウェアとして leJOS[4]を採用した．

コースは A0 でポスター印刷し赤色のビニールテープでコース上の任意の点にチェックポイントを配置した．

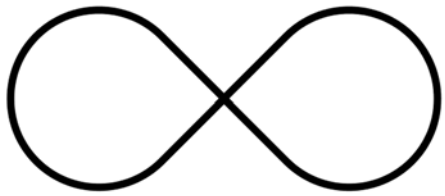

図 4. 使用したコース

5. 結果と考察

走行体に開発したプログラムを搭載し，コースを走行させてみたところ，直前の区間に他の走行体がいる場合にはチェックポイントで停止し，他の走行体が直前の区間から離れてから直前の区間に進入するという期待結果どおりの実結果が得られることを確認した．

5.1. 今後の課題

8 の字コースを走行させることに成功し，交差点での信号待ちができることを確認したが，他の条件でも衝突しないことを証明できたわけではない．

そこで新しい形のコースを作り，それに合わせた検証を作成し，衝突しないことの確認が必要である．そのためにも今回のコースでは起こり得ない状況が発生するコースを作り，それに合わせて検証を作り，モデルを作成，実装することが必要である．

参考文献

[1]

[2] C. A. R. Hoare, “Communicating Sequential Processes,” Prentice Hall, 1985.

[3] P. Welch, N. Brown, “JCSP,” http://www.cs.kent.ac.uk/projects/ofa/jcsp/ (2016/1/18 閲覧).

[4] The LEGO Group, “教育版レゴマインドストーム EV3,” https://education.lego.com/ja-jp/preschool-and-school/secondary/mindstorms-education-ev3 (2016/1/18 閲覧).

[5] leJOS, “leJOS EV3,” http://www.lejos.org (2016/1/18 閲覧).