並行システムにおけるコンポーネント間の状態の一貫性に関する研究

2011SE038 林寛人　2011SE219 酒部晃基
指導教員：張漢明

1　はじめに

並行システムのソフトウェア開発において，分散する状態間の一貫性を検証することは重要である．しかし，プログラムの段階で検証することは難しい．とりうる全ての振舞いのもとで，分散したコンポーネント間の状態不変条件を満たすことを検証する必要がある．このことから，設計段階で振舞い検証だけでなく状態に関する仕様記述を明確に検査する必要がある．

本研究の目的は，並行に動作するコンポーネント間の状態不変条件に関する一貫性検証支援である．ソフトウェア設計段階における仕様記述に対する効率的な検証方法を提示し，アーキテクチャ記述から設計段階において，各コンポーネントに分散する状態間の不変条件を系統的に検証する方法を提案する．

本研究の基本的なアイデアは，振舞い検証と状態に関する検証の分離をおこなう．分離をおこなうことで状態に関する検証をする際に，デッドロックなどの振舞いに関する不具合はない，という前提で検証することができる．また，検証が失敗した際には，追加した状態に関する記述の間違いである可能性が高くなる．形式仕様言語を用いて状態に関する記述をおこない，それを用いて SPIN[1] で検証をおこなう．状態変数を考慮しないアーキテクチャ記述の振舞い検証では，FDR を用いることでイベントの実行順序を検証し，アーキテクチャ記述の振舞いを検査する．状態に関する検証では，形式的な記述方法として VDM++[3] をもちいる．VDM++ は形式手法の 1 つであり仕様の段階で要求から実現すべき機能や制約などを記述することができるので，仕様に間違いなく作られているかどうかの確認が可能になる．

本研究では，振舞い記述と形式仕様言語からモデル検査言語へ変換して検証する方法を提示する．対応関係から VDM++ から Promela の変換をし，実際に自動販売機システムの事例を用いて，対応関係から検証における有効性の確認をした．

2　基本的なアイデア

2.1　状態の一貫性

状態の一貫性とは，並行システムにおけるコンポーネント間で変数が記述されている際に，以下のことを満たすことである．

- イベントの振舞いに関する仕様を満たす
- ある状態に関してコンポーネントの変数間でつねに成り立つべき条件を満たす

2.2　アーキテクチャ記述と振舞い検証

ソフトウエアの状態を考慮しない設計コンポーネントを本研究で用いるモデル検証ツールにより，イベントの送受信の順序関係に誤りがないかを検証する．

2.3　状態の一貫性検証について

一貫性における検証では変数を考慮した記述をおこなうが変数を考慮するには状態に関する記述を検査する必要がある．そのためには，振舞い検証と状態に関する検証の分離をおこない，状態に関する検証として変数を考慮した記述を追加し，検証をおこなう．本研究の概念図を以下の図 1 にまとめた．

1. アーキテクチャ記述の振舞いを FDR を用いて検証をおこなう
2. 状態に関する記述を VDM++ を用いて記述する
3. 振舞い記述と状態に関する記述をもとに SPIN で検証をおこなう

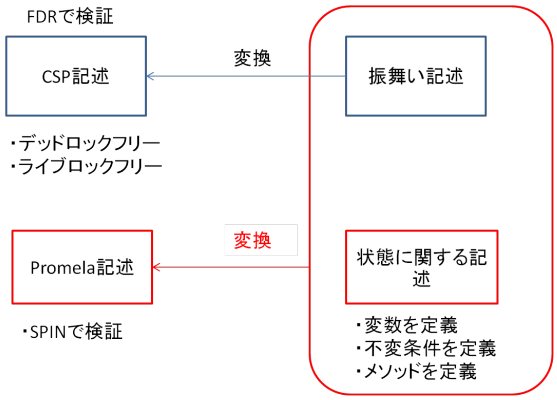

図 1　本研究の概念図

3　状態の一貫性検証

本章では，背景技術として VDM++，Promela を記述し，それぞれの記述の方法について説明する．

3.1　背景技術

3.1.1　VDM++[2]

VDM++ は，変数によって表現されたシステムの状態やデータ構造，関数定義，操作定義によって表現されるシステムの機能をモデル化の対象としている．検証の対象となる性質としてシステムの状態（変数）に関して成り立つべき条件式を不変条件として記述し，望ましい状態か否かの表示し，関数，操作についての事前，事後条件が成り立っているかを検査しながらインタプリタ実行をおこなうこと

ができる．つまり，必要な機能について詳細な記述が可能となり，様々なインスタンス変数の不変条件や操作に対する事前，事後条件を正確に記述することで制約が開発工程で失われないようにする．インスタンス変数や UML の関係に対する宣言の型を検査することや自動化支援を容易にする．以下は，記述例を示したものである．

3.1.2 SPIN[4]

G.J.Holzmann が開発，公開している自動検証法のひとつであるモデル検査ツールである．記述言語として Promela を用いる．Promela とは対象システムの各プロセス間をチャネル通信を用いて記述する．プロセスとは対象システムの振る舞いを表現したものである．SPIN 検証方法として以下のふたつが挙げられる．本研究では．表明を用いて検証をおこなう．

- 表明 (assert)
  プロセスの任意の箇所に assert 文（条件式) を記述することで不変条件を満たすかを検証
- 線形時相論理 (LTL)
  プロセスが時系列に沿って常に成り立つべきであることを記述

3.2 状態に関する検証

3.2.1 データ定義

VDM で使用する型は複数存在し，以下で分類する．また，VDM++ の基本形および本研究との関連を以下の表 1 にまとめた．

- bool 型
  仕様記述の計算が終了しない場合や結果がでないシステムを対象にし，値は「true(真)」,「false(偽)」を示す．
- 数値型 (nat1 型，nat 型，int 型，rat 型，real 型)
  これらは, 整数除算，法算，剰余算を除いて演算対象としての混在が可能である．
- 文字列型 (char 型)
  単一の全ての文字であり，改行や空白も単一文字として捉える．
- token 型
  異なる値の加算無限集合からなる．トークン型は相当または不等の演算しか用いることが出来ない．役割として，型の詳細をモデル化しない場合に使用する．

VDM++ と Promela における基本データ型を対応して考察すると，Promela 記述では主に数式をあつかうが実数すべてを使用することができないが，整数，自然数に関しては対応づけることができる．文字列型は文字を数式に変換することで対応は可能である．

タイプ	値	研究との関連
bool	true，false	○
nat1	1，2，3，4，5，...	○
nat	0，1，2，3，4，....	○
int	.....，-2，-1，0，1，2，...	○
rat	...，-1/2，...，1/2，..	×
real	...，-23.23，...,12.23，...	×
char	‘a’，‘b’，..，‘1’，‘2’，...，‘+’，‘-’，...	○
token	mk __ token	×

表 1 VDM++ の基本形

3.2.2 状態不変条件

VDM ＋＋では不変条件を検査するデータクラスを作成してインスタンス変数定義個所に各クラスで定義されている変数を定義し，不変条件式 (inv) を記述する．

```
class データクラス名

instance variables
  public 参照するクラスの変数 A : 型 ;
  public 参照するクラスの変数 B : 型 ;

  inv ( 変数 A と変数 B に関する論理式 );

end クラス名
```

3.3 Promela をもちいた記述

3.3.1 データ定義

Promela のデータ型を以下の表 2 のようにまとまた．また，検証の途中で状態が爆発しないように，なるべく小さいビット数を使う．

タイプ	値	サイズ（ビット数）
bit,bool	0，1，true，false	1
byte	0，...255	8
short	-32768，...，32768	16
int	-2^{31}，....，2^{31}，-1	32

表 2 Promela の基本形

3.3.2 状態不変条件

分散システムにおけるあるひとつの状態の時に成り立つべき条件を記述する．以下の点をもとに記述する．

- つねに成り立つべき条件を状態に記述する
- assert 文に成り立つ条件の論理式を記述する

```
active proctype クラス名 (){
do
 .....
```

```
  assert(論理式);
  .....
od;
}
```

3.3.3 UML から Promela への変換

- 変数の型定義，チャネルの定義はグローバルで定義する
- do 文は繰り返し文，if 文は条件分岐をあらわし，本研究では並行システムにおける検証をおこなうので，上記のふたつをよく用いる
- ->はガード文を示し，前の条件が真ならば後の条件をおこなう

以下は図 2 の UML 記述から Promela 記述への記述例を示す.

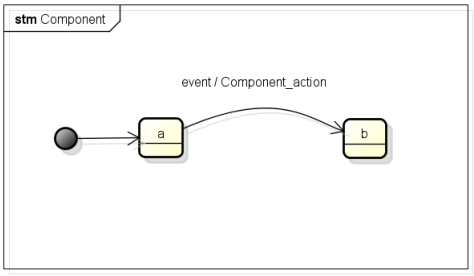

図 2 UML 記述

```
mtype = {a,b}
mtype = {event , action}

chan chan_Name1 = {キューの大きさ} of {mtype}

int state = a;

active proctype Name1(){
do
::state == a ->
  chan_Name1?event ->
           chan_Name2!action; state = b;
::state == b
od;
}
```

4 事例検証：自動販売機システム

自動販売機システムにおける検証では商品の販売された価格の合計と売上金額の合計がつねに等しくならなければいけない．振舞い記述から状態に関する記述を追加し，不変条件の検証をする例として売上を保持する Safe クラスを用いて以下のように追加した.

4.1 Safe クラスのクラス図

以下の図 3 は Safe クラスのクラス図を示している.

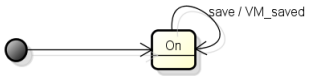

図 3 Safe クラス

4.2 VDM 記述

状態に関する記述を VDM を用いて記述する.

```
class Safe

instance variables
  public safe:nat:=0;
  public amount1:Coin;
  public stock1:Item;

  inv (stock1.stock=amount1.amount);

operations
  Saled()==
   (safe:=safe + amount1.amount;
   amount1.amount:=0);

end Safe
```

4.3 Promela 記述

振舞い記述と VDM 記述を用いて Promela で記述する.

```
mtype = {Off,On}
mtype = {save,saled}

chan ch_VM = [2] of {mtype,byte}
chan ch_Safe = [2] of {mtype}

int Safe_state = Off;
byte safe = 0;

active proctype Safe(){
do
::Safe_state == Off;
  if
   ::ch_Safe??on; Safe_state = On
  fi;

::Safe_state == On;
  if
   ::ch_Safe??save;
    safe = safe + coin;
```

```
    ch_VM!saled,0;
    Safe_state = Off;
  fi;
od;
}
```

4.4 アサーション記述

Item クラスに記述した商品購入金額 (stock) と Safe クラスに記述した売上金額 (safe) の不変条件を記述する．VDM++ ではデータクラスに作成し，つねに成り立つべき記述をした．

```
instance variables
  public item:Item;
  public safe:Safe;
   inv (item.stock=safe.safe);
```

Promela 記述は以下のように記述する．

```
proctype VM(){
.....
        assert(stock == safe);
.....
}
```

5 考察

5.1 検証の意義

VDM++ を用いることで状態に関する記述に着目することができ，定義に関する記述不足が解決される．システムにおける個々の操作において関数やメソッドを記述し，保持される値とコンポーネント間の関係を明記できる．以上の状態記述をもとに Promela 記述に対応づけることで状態の一貫性検証を可能にし，想定しうる振舞いにおいてある変数間でつねに成り立つべき性質を満たすことができる．並行システムの状態を考慮した一貫性検証を容易にすることを実現する．

5.2 VDM++ から Promela への変換

自動販売機システムの記述に関して，VDM++ は検証する際に商品を登録し，価格設定をおこないうが Promela 記述と対応するには，Promela では数式を扱うので登録や設定をおこなうのが動作が困難であるので，あらかじめ商品名（変数名）と価格（値）を宣言する必要がある．

5.2.1 VDM++ について

- 変数定義
  VDM++ ではインスタンス変数を直接参照することはできず，参照するための操作を用意し，それを外部へ公開しなければならない．また，インスタンス変数に不変条件を定義することもできる．
- 不変条件
  今回の検証ではコンポーネント間の状態の不変条件を記述することである．記述における問題点はイベントにおける状態遷移に考慮した不変条件記述が困難である．不変条件の記述箇所を考察しなければならない．

5.3 アーキテクチャの詳細化

今回，記述をする際の対応表を作成した．これにより，基本の型定義やメソッドの定義をするときにはこれを用いることができる．しかし，数値を用いる際には，Promela では大きな値を用いると，状態爆発が起きる可能性があるので，検証する際には小さな値に変換する必要がある．

5.4 検証の自動化について

本研究の目的である検証の自動化であるが，対応関係の調査だけでは不十分である．その理由を以下に述べる．

- 状態数の変化
  不変条件を記述した時，Promela は同期をとることができないので，状態を増やしてイベントを受信する必要がある
- 不変条件の記述
  VDM++ で，不変条件をどのクラスに記述するかを指定することができるが，そのクラスのどの状態に時に不変条件の記述をするかも指定をすることができない

以上の点から本研究では自動化の支援として対応づけをおこなった．

6 おわりに

本研究では，並行システムにおける状態の一貫性の検証として，振舞い記述から形式仕様言語とモデル記述言語を用いて，検証の自動化の支援をおこなった．今後の課題として，前項で挙げた点をどのようにして形式化するのかという方針が挙げられる．この点を改善することで検証の自動化が実現でき，一貫性検証を容易におこなうことができる．

参考文献

[1] G. J. Holzmann, “*The Model Checker Spin*” IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 23, no. 5, 1997.

[2] 石川 冬樹，荒木 啓二郎，“VDM++ による形式仕様記述 (トップエスイーシリーズ 実践講座)” 近代科学者出版，2011.

[3] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, Marcel Verhoef (酒匂寛 訳), “VDM++ によるオブジェクト指向システムの高品質設計と検証” 翔泳社，2010.

[4] Mordechai Ben-Ari(中島震 監訳，谷津弘一・野中哲・足立太郎 共訳), “SPIN モデル検査入門” オーム社出版，2010.