Vol.2011-SE-171 No.8
2011/3/14

VDM記述からのPromela記述生成における変換手法の提案

宮下 怜[†2] 大森洋一[†1]
日下部 茂[†1] 荒木啓二郎[†1]

開発早期での検証の手段として形式手法が注目されている．形式仕様記述は開発対象の機能側面に対して厳密な仕様記述を行うことを可能とする．しかし，開発対象の振る舞いの検証において完全性を欠く．一方，形式検証に含まれるモデル検査は開発対象の状態を網羅的に探索することによって振る舞いを検証する．そこで，本論文では形式仕様記述によって得られた厳密な仕様記述をモデル検査が可能な記述へと変換する手法を提案する．

A Proposition of A Transformation to Generate Description in Promela from Description in VDM

Ryo Miyashita,[†2] Yoichi Omori,[†1]
Shigeru Kusakabe[†1] and Keijiro Araki[†1]

Formal methods attract attention as a way of verification at early statge of development. Formal specification enables you to describe strict specification for functional aspects of development objects. But,it lacks completeness in verifying actions of development objects. In contrast, model checking included formal verification verifies actions of development objects by exploring in their state exhaustively. Then, to solve formal specification's problem, we suggest transform method to generate a description for model checking from a strict specification described with formal specification.

†1 九州大学 大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University
†2 九州大学 大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University

1. はじめに

1.1 研究の背景

利用可能な計算機能力の飛躍的な向上に伴って，形式手法によるソフトウェア検証の現実的な問題に対する応用が期待されている．特に有限状態機械の遷移を網羅的に検証するモデル検査は，記号モデル検査[1)] などのアルゴリズム改良，並列処理[2)]， SAT/SMT の利用[3)] による探索効率の改善，有界モデル検査[4)] などにより，自動化および適用範囲を拡大しつつあり，産業界からの期待も大きい．その背景には，ソフトウェアの開発工程において発見される不具合のうち，半数以上が開発早期で作り込まれるという問題がある[5)]．このような問題に対し，開発早期に要件分析を十分に行い，仕様を検証することが可能な形式手法は不具合の排除を支援できる．計算機のソフトウェアは有限状態機械としての性質を備えているはずであり，数理的にあらゆる検証が可能といえる．

しかしながら，ソフトウェアの設計品質を向上させるためには，検証技術を自動化するだけでなく，対象システムの本質的な性質を抽出し検証する仕様記述技術が重要である．利用者からの要求を外した品質項目やシステムの本質を表さないモデルについて検証しても，そこで保証されるモデルの性質は対象システムの評価に関して実効的な意味をもたないからである．ソフトウェアを含むシステム利用者からの要求は，そのシステムにより新たに実現して欲しい機能やサービスがまず最優先である．すなわち，少なくとも開発の初期段階における要求は機能単位で記述されており，最終的な製品評価おいても機能の観点は欠かせない．

これらの長所を組み合わせるために，複数の視点，とりわけ要求から導かれた機能に関する仕様記述のモデルと状態の遷移に関する制約に関する振る舞い仕様記述のモデルを併用した，対象システム検証が設計の安定性を増すために有用であることが知られている[6)]．もちろん，その場合はモデルの数に応じて適用コストが増加するため，そのコストに見合う安全性が要求されるシステムに対象が限定されていた．

1.2 研究目的

前節の理由から，形式手法を含むこれまでのソフトウェア開発手法の多くは機能を対象としたものであった．形式手法においては，その初期に Dijkstra による段階的詳細化手法が既に提案されており[7)]，数理的な証明を用いて，形式的に仕様記述された機能要求から実行可能なプログラムを演繹的に得られる手順が知られていた．しかし，このような証明は数学的な訓練と知識を必要とし，誰にでもできるものではなかったので広く利用されるには至らなかった．つまり，形式手法の利点は理解されていたものの，適用コストの高さが普及を妨

げる大きな要因であった．

数理的証明に代わる手段として，特定の値に関する評価により仕様検証を行なうアニメーションや特定の数式に関する自動定理証明といった，計算機上のソフトウェアツールによる形式手法適用補助が試みられてきた[8]．これらは限定された，しかし頻出するパターンについて，ツールが証明を代行するといった性格のものであったが，検証用演算能力の向上は，限定される範囲を小さくし，完全な検証に近づいている．

この点に着目し，ツールの役割をより積極的に捉え，ツールで証明可能な範囲で形式手法を利用する軽量形式手法 (light weight formal methods) が提唱された[9]．ツールの利用により，複数のモデルを利用する場合も，現実的な検証コストに抑えられると期待できる．モデル検査はツールによる自動化に適しており，完全な証明を与えられることから，振る舞い検証をモデル検査により行なう手法が特に注目されている．

本研究では，軽量形式手法を前提とし，機能仕様に関するモデルから振る舞い仕様に関するモデルを生成し，より低いコストで複数のモデルを用いた検証を可能とする手法を提案する．本稿では，具体的な組込みシステムを対象として，機能に関する記述を得意とする仕様記述言語である VDM++によるモデルから，振る舞いに関する記述を得意とする仕様記述言語である Promela によるモデルへの変換手法について述べる．VDM++ および Promela は，それぞれ長期に渡って使用されている安定した検証ツールと多くのユーザをもち，本研究の試行環境として適切な特性を備えているからである．

2. 仕様記述言語

2.1 VDM

形式手法の一つである VDM(Vienna Development Method) はシステムのモデル化と分析，詳細設計やコーディングへ至るための技術の集合である．形式手法の中ではモデル規範型と呼ばれており，形式仕様記述の方針としてモデルの状態を記述することに重点が置かれている．

2.2 VDM++

VDM++はモデル指向の形式仕様記述言語であり，同じく VDM で用いられる形式仕様記述言語 VDM-SL をオブジェクト指向拡張した言語でもある．VDM++は開発対象のデータを型として定義し，機能を操作・関数として定義することによってモデルを構築する．開発対象の機能的側面を仕様として厳密に，かつ適切な抽象度で記述することが出来る．さらに，VDM++の特徴として支援ツールの VDM++ Toolbox が存在することを挙げる．本ツールは仕様の構文や型のチェック，インタプリタによる機能仕様の検証（仕様アニメーション）を可能としている．

2.3 クラス・型定義

VDM++による仕様記述では，開発対象のデータを型やクラスとしてモデル化を行う．型やクラスの候補は主に開発対象の仕様中の名詞句から抽出する．また，不変条件と呼ばれる型や状態変数の不変の性質を記述することができる．図 1 に，VDM++による型定義の例を示す．

```
public Thermister = real
    inv thermister == -10 <= thermister and thermister < 110;
```

図 1 VDM++のデータ例
Fig. 1 A data example of VDM++

2.4 操作・関数定義

VDM++による仕様記述では，クラス・型定義を行ったデータに対する動作を操作又は関数として定義する．この操作と関数は開発対象の仕様中の動詞句から主に抽出する．

操作や関数はシグネチャ，本体，事前条件，事後条件によって構成される．事前条件・事後条件はそれぞれ，操作および関数を適用する前に満たすべき条件と適用した後に満たすべき条件である．また，操作として定義した場合は状態変数を変更することが可能であるが，関数の場合は不可能である．図 2 に，VDM++による操作定義の一部を例として示す．

```
operations
public heating : () ==> ()
   heating() == (
      while pot_status.thermister <> 100 do
               (省略)
      pre    pot_status.cover = <ON> and
               (省略)
      post   pot_status.thermister = 100 and
             pot_status.timer = 3;
```

図 2 VDM++の関数例
Fig. 2 A function example of VDM++

2.5 モデル検査

モデル検査とは，形式検証の一つであり，システムの取り得る状態をすべて生成してそれぞれの状態が正当性仕様を満足するかどうか検査する検証方法である[10]．この手法を用いることによって，システムがデッドロックに陥らないことや到達可能性（ある特定の状態に到達すること）などを検証できる．

2.6 Promela

Promela（Process Meta Language）は並行動作システムの整合性，特にデータ通信のプロトコルを分析するための形式仕様記述言語である．Promela の記述において，特に本論文の中で重要なものを以下に示す．

プロセス型

Promela での動作単位となる．Promela のモデルはプロセスの集まりで構成され，各プロセスの相互作用によって動作が決まるという特徴がある．また，変数の変化が他のプロセスに影響を与える可能性があるため，プロセスの並行性については注意しなればいけない．

mtype 型

この型を使用することによって，値に記号名をつけることが出来る．また，Promela における列挙型にもあたり，複数の値を表現することも可能である．ただし，列挙型にタグをつけて複数に分類することができないという制限を持つ．

if 文

if 文は Promela における条件分岐である．複数の条件を定義できるが，そのうち真となる条件がない場合は if 文の手前でプロセスが停止する．また，真となる条件が複数ある場合はその中からどれか一つが選ばれて実行される．さらに，条件を示さないことによって意図的に非決定的な処理を行わせることも可能である．

assert 文

assert 文は表明を行うための文である．これは，記述した部分に実行の制御が来たときに指定した条件が成立していなければならないことを示す．条件が満たされていれば実行時には何も表示されない．条件が満たされない場合のみエラーが表示され，実行が停止する．

インライン展開

インライン展開とは，複数の文の列に名前をつけることである．プロセスの中にインライン展開の名前が出てくると，その部分にインライン展開の名前に該当する複数の文がコピーされる．コピーの際，インライン名にある仮パラメータは，呼び出し時の実パラメータの置き換えられる．

never 要求　ここでは，本来満たしてはいけない条件を記述する．検証が実行されたときに，この条件が満たされた場合はエラーが表示される．

2.7 SPIN

SPIN（Simple Promela Interpreter）はモデルチェッカと呼ばれるソフトウェアツールである[10]．実システムとして使われるソフトウェアのモデル検査を目的としている．SPIN では以下の手順に従ってモデル検査を行う．

(1) システムの振る舞いを表すモデルを作成する
(2) その振る舞いの正しさを時相論理などを用いて記述し，システムの要求定義を表す
(3) モデルチェッカで検証する
(4) 検証の結果から以下のどちらかの対応を取る
  (a) 記述した正しさをモデルが満たしていれば，検証を終了する
  (b) 記述した正しさをモデルが満たしていなければ反例が示されるので，結果の検討を行いモデルを修正し，再度の検証を行う

この手順を踏むことで，最終的にシステムの正しさを確保できる．ただし，取り得る全ての状態を生成するために状態数が多くなり，検証が不可能になる場合がある．このような問題を避けるために，取り扱うデータをシステムの影響がない範囲に絞り込んだり，実際のデータを有限個のデータにマッピングするなど抽象化が必要となってくる[11]．

3. 提 案 手 法

形式手法は，それぞれの基礎となる数理体系によって，表記法や検証能力が異なる．例えば， VDM++ は集合論および一階述語論理に基づく言語であり，データとしての集合および集合の関連により機能を表現する．Promela は，状態変数の値により識別される状態と状態間の遷移条件を記述する言語であり，状態変数によりデータを，遷移条件と次状態により機能を表現する．事前条件は遷移の元状態，事後条件は遷移の先状態である．

VDM++で記述したモデル（以下 VDM 記述）の振る舞いは支援ツール VDM++ ToolBox を用いて特定の値に対するアニメーションは可能だが，網羅的に検証することは困難である．本章では，VDM 記述を Promela で記述したモデル（以下 Promela 記述）に変換する手法を提案する．これによって，SPIN を用いたモデル検査を利用した振る舞いの形式的な検証が可能となる．

3.1 状態の対応付け

本手法適用の前提として，Promela 記述の元となる VDM 記述について，振る舞いが記述されていることを挙げる．また，VDM Toolbox の構文チェックおよび型チェックを実行済みであることも前提とする．

VDM 記述と Promela 記述はともに設計対象となるシステムを状態機械としてモデル化する．ただし，それぞれの表記法と基盤とする数理体系の違いにより，モデル構築およびモデル検証のしやすい性質が異なる．

このため，以前の研究[12] では， VDM 記述と Promela 記述で，適切と考えられる抽象度や状態変数の選択が異なり，必ずしも機械的な変換が可能ではなかった．このため，特徴的な状態変数について注目することで扱う状態数の削減を試みたが，多くのシステムでは状態変数は互いに依存しあっているので，これだけでは有効な効果が得られない場合が少なくなかった．

この問題を解決するためにラベル遷移システムによる橋渡しを行なうというのが，本研究の基礎的なアイデアである．すなわち，特にソフトウェア設計の上流工程に注目し，状態を状態変数の集合として表現する前に，状態の識別に専念し， VDM 記述と Promela 記述で共通の状態に共通のラベルをつけてモデル化し，両者の対応づけを保障する．

状態とラベルの対応は任意に付けられるので，これだけでは形式的な検証とならない．次のステップとして，ラベルを媒介として VDM 記述の状態変数の集合と Promela 記述の状態変数の集合を表 1 のような表形式により対応付ける．従来の手法と異なるのは，両者のあらゆるの状態の対応付けを考えるのではなく，対応付けられる可能性がある状態の組み合わせのみについて考慮することで，これによりモデル変換に必要な作業量を大幅に削減できる．

表 1 VDM++の状態と Promela の状態の対応例
Table 1 An example of map of states in VDM++ and Promela

ラベル	VDM++の状態	Promela の状態
沸騰中	int: therm ¿= 100 therm ¿= 100	waterstate : offPID, onPID, boiling waterstate = boiling
ミルクモードで沸騰行為中	int : TargetTherm targettherm = 60 cover = on	TargetTherm : int TargetTherm = 60

表 2 VDM++の型と Promela の型の対応 (一部)
Table 2 Map of types in VDM++ and Promela (partial)

VDM++の型定義	Promela の型定義	値
bool	bool,bit	0,1,true,false
nat,nat1,int	byte,short,int,unsigned	整数値
引用，列挙	mtype	値につけた名前の集合 A,B,C,...
複合	タイプ定義	typedef Newname{...}
列	配列	a[index]

3.2 状態変数の変換

記述の変換は VDM++と Promela の記法の共通点を基に，対応を取っていく形で行う．変換は以下の手順で進める．

A. 型, 定数の変換

型，変数の変換については以下の**表 2** のように対応を取る．VDM 記述と Promela 記述は視点が異なるため，対応付けによる自動的な変換だけでなく，適宜仕様記述の追加を行なう．この際，どちらかに追加した情報は両方のモデルに追加するようにし，あまり両者が乖離しないように注意するのが望ましい．適当な時点で再度変換することで，確実な一貫性を保障する．

また，具体的な値が決まっていないデータ（0 ≦ x といった範囲を指定する形で設定してあるデータ）がある場合には，これらに対し，値をいくつかの代表値に分けて範囲とマッピングし，新しく mtype 型として定義し直す．代表値としてではなく，通常の数値として扱いたいのならば，より少ないビット数を用いる．これによって，状態数を減らす．

定数の変換については，#define を用いることによって値の記号化として実現する

B. インスタンス変数から大域変数への変換

VDM 記述においてインスタンス変数として定義されているものは，Promela 記述では大域変数として扱う．また，型およびインスタンス変数に対する不変条件は never 要求として，この不変条件を否定する条件を記述することで実現する．

C. 操作および関数からインライン展開への変換

VDM 記述の操作および関数のシグネチャ，本体，事後条件をインライン展開へと変換する．事前条件については後述のプロセスの記述において説明する．まず，シグネチャは関数名はインライン展開の名前に，入力パラメータは仮パラメータとして表現する．出力パラメータについては，Promela の記法では表現できないため，大域変数に出力

用の変数を用意するなどして対処する．本体については Promela の記法に従って同様の処理を行うように書き換える．事後条件は assert 文に置き換え，インライン展開の最後に記述する．これによって，事後条件と同じ働きが得られる．

D. プロセスの記述

最後に，クラス中の操作および関数を全て含むように，インライン展開を利用してプロセスを記述する．各操作，関数毎の事前条件を考慮するために，if 文の条件式の後にインライン展開名を記述する．事前条件は，操作や関数が記述する前に成り立つべき条件なので，if 文の条件式とする．これにより，対応するインライン展開に対して事前条件を満たさない限り実行できないことを表現することができ，VDM++における事前条件と同じ働きを持たせることが出来る．

3.3 動的な性質の追加

VDM++は時間の概念や動的に変化する型や値の表現に向いていない．しかし，Promela であれば，if 文などを利用した非決定的な処理によって動的な性質を素直に記述できる．そこで，VDM++では表現が難しかった，動的に値が変化するデータや任意のタイミングで発生するイベントなどを Promela 記述に追加する．これらは，該当する変数の値を非決定的に書き換える処理を繰り返すプロセスとして記述する．

4. 提案手法の適用事例

本章では，3 章で提案した手法を VDM 記述に実際に適用した事例について述べる．

4.1 適 用 対 象

適用対象として「話題沸騰ポット（GOMA-1015 型）要求仕様書 第 7 版」[13] に記述されている話題沸騰ポット（以下ポット）の要求仕様を扱う．ポットは**図 3** の状態遷移図に示す動作を行う． 本論文では，ポットの沸騰行為について着目した．沸騰行為は以下のような機能側面を持っていることが示されている．

- 100 ℃まで水を加熱し，3 分間カルキ抜きを終えるまでを示す
- 目標温度 ON/OFF 方式でヒータを制御して沸騰させる
  - 目標温度 ON/OFF 方式は目標温度≦温度でヒータを OFF，目標温度＜温度でヒータを ON する
  - 操作量はヒータ ON 時は 100%,OFF 時は 0%とする
- エラーが検知されるか，蓋センサが OFF もしくは全ての水位センサが OFF になった場合，沸騰行為をやめる

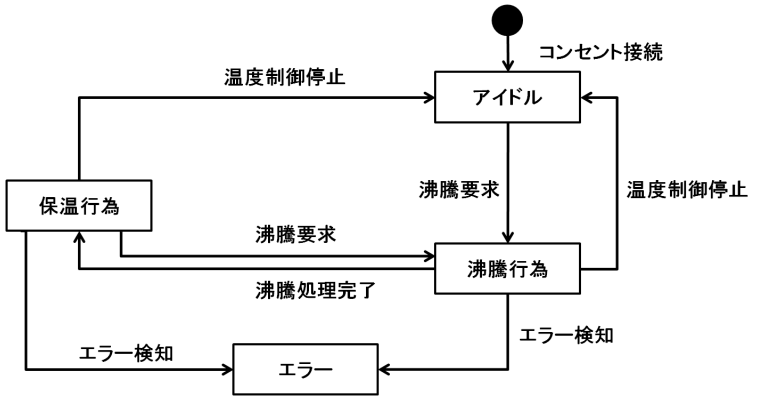

図 3 ポットの状態遷移図
Fig. 3 statemachine diagram for a pot

- サーミスタが 100 ℃になったら，さらに 3 分間ヒータで加熱を続ける（ヒータを ON し続ける）
- カルキ抜きの加熱を終えたら，沸騰行為を中止し，保温行為に遷る

このポットは沸騰行為の状態において，さらに状態が細分化されている．この細分化された沸騰行為の状態遷移図を 4 に示す．沸騰行為は「加熱」から「カルキ抜き」と，温度変化や時間を発火条件として状態が遷移という仕様になっている．また，沸騰行為のどちらの状態からもエラー状態もしくはアイドル状態へと遷ることがある．

提案手法の適用に当たって，まず，このような機能側面の VDM 記述を作成した．

4.2 モデルの変換

機能側面の VDM 記述に対して 3.2 節で提案した手法を用いて，VDM 記述を Promela 記述に変換した．

4.2.1 型 の 変 換

型の変換において，引用型に対しては mtype 型への変換を行った．引用型以外の型については int 型が使用されており，値の範囲を示す形で使用されていた．そのため，データの抽象化を行って代表値を割り当て，mtype 型として定義し直した．その際に着目したデータと抽象化の結果を以下の**表 3** に示す．

サーミスタは VDM 記述の加熱処理において 100 という目標値が事後条件で示されてい

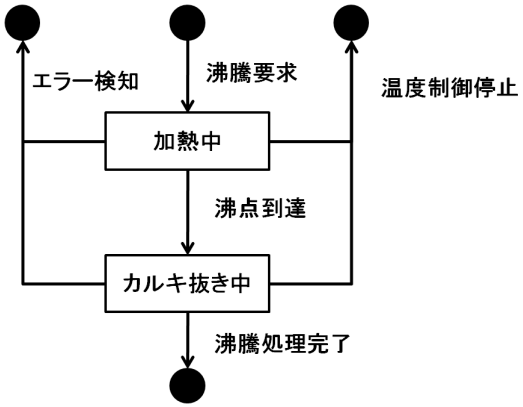

図 4　沸騰行為の状態遷移図
Fig. 4　statemachine diagram for boiling

る．また，目標温度 ONOFF 方式において目標値未満と目標値以上という 2 パターンについての処理が言及されている．そのため，mtype 型の 3 つの値として定義した．時間についても，VDM 記述のカルキ抜きの処理において 0,0 から 3，3 の 3 つのパターンに言及してあったため，これらを示す 3 つの値を定義した．操作量については，実際に値を見るのは 0 と 100 の 2 値だけであったため mtype 型として定義した．

4.2.2　操作の変換

操作の変換についての結果として，Promela 記述において事前条件と操作の呼び出しを表した部分を**図 5** に，操作の本体と事後条件を表した部分を**図 6** に示す．これは**図 7** の VDM 記述から変換したものであるが，型の変換において対応ができないものがなかったため，同様の性質を表した記述になっている．

表 3　データの抽象化の結果
Table 3　data abstraction

データ名	VDM 記述での値の使用範囲	Promela 記述での値の範囲
サーミスタ	-10 ≦ t ＜ 110,100	Under_Target_Temp,Boiling_Point,Over_Target_Temp
操作量	0,100	MIN,MAX
時間	0,0 ≦ t ＜ 3,3	Zero,Zero_to_Three,Three

```
if/*保温行為の事前条件*/
:: atomic{pot_status.cover == ON && pot_status.water_level == Exist_ON &&
          pot_status.error_status == None &&
          pot_status.temp_ctrl.ctrl_method == Not_Use &&
          pot_status.timer == Zero -> trans_keep_warming()};
:: else -> skip
fi;
```

図 5　保温行為への遷移 (事前条件)
Fig. 5　Transition for heat retention(precondition)

```
/*保温行為への遷移*/
inline trans_keep_warming(){
/*操作*/
        pot_status.temp_ctrl.ctrl_method = PID;
/*事後条件*/
        assert(pot_status.cover == ON &&
                pot_status.water_level == Exist_ON &&
                pot_status.error_status == None &&
                pot_status.temp_ctrl.ctrl_method == PID);
}
```

図 6　保温行為への遷移 (本文，事後条件)
Fig. 6　Transition for heat retention(body, postcondition)

4.2.3　プロセスの追加と検証

4.2.3.1　プロセスの追加

追加プロセスの対象として，水温，時間，温度制御停止条件の発生の 3 つを記述した．水温，時間については値の変化に対する VDM 記述が存在しなかったため，これらの値を変化させるプロセスを追記した．温度制御停止条件の発生については，蓋センサと水位センサ，エラー検知の状態を非決定的に選択して変化させることを繰り返すプロセスを追加した．

4.2.3.2　検　　証

これらの追加プロセスを用いて SPIN での検証を行った．本論文では，各操作において事

```
--保温行為への遷移
public trans_keep_warming : () ==> ()
      trans_keep_warming() ==    pot_status.temp_ctrl.ctrl_method := <PID>
      pre    pot_status.cover = <ON> and
             pot_status.water_level = <Exist_ON> and
             pot_status.error_status = <None> and
             pot_status.temp_ctrl.ctrl_method = <Not_Use> and
             pot_status.timer = 0
      post pot_status.temp_ctrl.ctrl_method = <PID> and
             pot_status.cover = <ON> and
             pot_status.water_level = <Exist_ON> and
             pot_status.error_status = <None>;
```

図 7 保温行為への遷移（VDM）
Fig. 7 Transition for heat retention(VDM)

前条件が満たされた場合に，事後条件が満たされているかどうかの検証を行った．この検証を行うことによって，事前条件が満たされている状態の下で操作を実行し，事後条件が満たされていればその操作が正しいということができる

本論文の検証では，Promela 記述のインライン展開において処理の最後に記述している assert 文の条件が成り立っているか成り立っていないかを確認する．インライン展開の処理が実行されている時点で，if 文の条件として記述した事前条件が満たされており，事後条件である assert 文の条件が満たされていれば，SPIN はエラーを返すことなく検証を終了するはずである．事前条件が満たされない場合には if 文の後に続くインライン展開が実行されず，assert 文が実行されないために事後条件も確認されないことになる．

検証の結果，加熱処理の実行中にエラーが発生したため，事前条件が満たされても事後条件が満たされない状態が成り立つことが判明した．原因は加熱処理の実行中に温度制御停止の発生プロセスが蓋センサの状態変数を OFF に変更したため，加熱処理の事後条件のうち蓋センサ$\bar{\text{O}}$N という条件が成立しない状況が発生したことである．そのため，VDM 記述の事後条件を変更するか，操作中に他のプロセスが状態変数の値を書き換えることが出来ないようにする必要がある．

5. 考　　察

5.1 型, 変数の変換について

本論文における VDM 記述への手法の適用の範囲内では問題なく行うことができた．これは，今回の適用事例では VDM 記述で使用している型について，Promela で規定されている型との対応が容易なものであったためである．しかし，VDM++によって記述されたモデルならば，寧ろ提案手法では対応できていない写像型や集合型の変数が含まれていることが一般的であると考えるため，提案した手法の適用は限定的になると考える．この点が課題として残っている．提案手法の作業は，型の変換を基礎として進めるものであるため，ここでの問題を解消できない限り以降のすべての変換作業に対して支障が生じるのは明らかである．現状のままで本提案手法を用いるのであれば，機能側面の VDM 記述の段階で写像型や集合型を使わずに記述できる範囲に留めておく必要がある．

5.2 操作および関数の変換について

VDM++における操作および関数の記述については，その記法に関しては変換が可能であることが提案手法の事例への適用の結果から確認できる．しかし，型や変数の変換と同様に対応できていない型については，その型に対する式や文の記述ができないため，ここでも課題が残っている．インライン展開の使用についても，今回はを用いているがこの部分はプロセスとして置き換えることも可能であると考える．プロセスに置き換えることによって，局所変数を持つことが出来るなど，表現可能な範囲が広がる．そのため，操作および関数についてはプロセスに変換する方が適当であると考える．

5.3 モデル検査の適用について

VDM 記述から Promela 記述へ変換することの目的は，VDM 記述へのモデル検査の適用を可能とすることによって振る舞いの検証においての欠点を補うことである．その点に関して，今回の適用事例においては目的を果たすことができたと考える．実際に VDM 記述中の不備 SPIN によるモデルチェックによって発見できている．ただし，提案手法における他の変換過程と同様に型の変換に基づく問題がここにも影響している．

6. おわりに

6.1 まとめ

本論文では，VDM 記述を Promela 記述へと変換する手法を示した．提案した手法では，VDM++と Promela の記法の共通点を基に変換を行う．この手法を VDM++で記述されたポットの機能仕様に実際に適用し，Promela 記述を生成した．その結果，VDM++で記述した機能仕様に対し，モデル検査が可能となることが確認できた．また，VDM++では検査することが出来なかった動的な変化に対応した検査を行うことが可能であることも確認できた．

6.2 今後の課題と方針

VDM 記述から Promela 記述へと変換する手法を提案したものの，現時点では VDM++と Promela との間で対応を取ることができない型や式，文が存在する．そのため，今回の手法の適用範囲は狭いものとなってしまっている．今後は，対応が取れない型への対処として，直接の対応ではなく，ラベルなどを用いて状態変数による状態の表現を代用した変換を行う方法を機械的に行なうアルゴリズムを考案し，ツールとして実装することにより形式手法に関する専門的な知識を不要とすることを目指す．これによって，より広範囲の事例について本手法を適用可能とする．

謝辞 本研究の一部は科学研究費補助金（21300009），基盤研究（B）「ソフトウェア開発の現場で使えるフォーマルメソッドに関する研究」の助成を受けたものである．

参考文献

1) McMillan, K.L.: *Symbolic Model Checking*, Kluwer Academic (1993).
2) Böhm, M. and Speckenmeyer, E.: A Fast Parallel SAT-Solver - Efficient Workload Balancing, *Annals of Mathematics and Artificial Intelligence*, Vol.17, No.3–4, pp. 381–400 (1996).
3) Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking, *Proceedings of the 14th International Conference on Computer Aided Verification*, pp.359–364 (2002).
4) Biere, A., Cimatti, A., Clarke, E.M., Fujita, M. and Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs, *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pp.317–320 (1999).
5) 佐原　伸：形式手法の技術講座 – ソフトウェアトラブルを予防する，ソフトリサーチセンター ((2008).
6) Farias, A., Mota, A. and Sampaio, A.: Efficient CSP Z Data Abstraction, *Proceedings of the 4th international conference on Integrated Formal Methods*, pp.108–127 (2004).
7) Wirth, N.: Program development by stepwise refinement, *Communication of ACM*, Vol.14, No.4, pp.221–227 (1971).
8) CSK システムズ：http://vdmtools.jp/.
9) Jones, C.B., Jackson, D. and Wing, J.: Formal Methods Light, *IEEE Computer*, Vol.29, pp.20–22 (1996).
10) Ben-Ari, M.: *Principles of the Spin Model Checker*, Springer-Verlag London Limited (2008).（中島 震，谷津 弘一，野中 哲，足立 太郎．訳 SPIN モデル検査入門オーム社 (2010)).
11) 吉岡信和，青木利晃，田原康之：SPIN による設計モデル検証，近代科学社 (2008).
12) Miyoshi, K., Kusakabe, S. and Araki, K.: Extracting State Machines from Model-based Formal Specifications by Focusing on Data Types, *Computer Software*, Vol.23, No.2, pp.2_211–2_224 (2006).
13) 組込みソフトウェア管理者･技術者育成研究会：話題沸騰ポット（GOMA-1015 型）要求仕様書. http://www.sessame.jp/workinggroup/WorkingGroup2/ POT_Specification.htm.