形式仕様記述言語 Alloy からの C#コードの生成

Generating C# code from fomal method language Alloy

電気電子・情報工学専攻　　AES1813　渡邊 優樹　（指導教員　西村　俊二）

Key Words:Alloy,C#,formal methods

１．緒　　言

近年，ソフトウェアは様々な分野で利用されており，私達の生活と密接な関係がある．従って，プログラムのバグによりソフトウェアが停止する等の不具合があると多くの人に迷惑がかかる．場合によっては人命に関わることもあり，バグの無い，信頼性の高いソフトウェアを開発する必要がある．

現在ソフトウェア開発では，自然言語により記述された仕様書の曖昧さによるバグが度々発生している．仕様書の読み手によって認識にずれが生じてしまう事がバグの原因となる．自然言語の曖昧さによるバグの発生を抑えるために，形式手法と呼ばれるプログラム開発手法が注目を集めている．形式手法とは，数理論理学に基づき，効率よく高品質なソフトウェアを開発する手法の総称である．

形式手法の一つに形式仕様記述がある．ソフトウェアの仕様を形式仕様記述言語で記述することで，仕様の厳密な定義を行う．記述した仕様を専用のツールを用いて検証することで，ソフトウェアの品質や安全性を確保する．

実際の製品開発に用いられた例[1]もあり，形式仕様記述の有用性は既に示されている[2]．しかし，形式仕様記述を用いる開発方法は，用いない開発方法と比べて開発工程が多くなることや，形式仕様記述言語を扱える技術者が少ないことなどを理由に一般的にはあまり普及していない．形式仕様言語からコードを直接生成できれば，開発工程が減るため，プログラムの開発効率もあがり，ヒューマンエラーの予防も可能になるため，形式仕様記述の導入もより容易になると考えられる．形式仕様記述言語の一つ，VDM++は vdmtools という専用のツールを用いることで，Java や C++のコードを生成することができる．また，C#のコード生成に関する研究[3]もおこなわれている．

しかしながら，形式仕様記述言語 Alloy[4]の専用ツール Alloy Analyzer[4]には他言語のコード生成機能はない．これまで，Alloy から JML 仕様の生成の研究[5]や，Alloy を用いた Java プログラムの開発に関する研究[6]は行われている．しかし，C#プログラムの生成に関しては報告されていない．ソフトウェア開発において用いられる事の多い C#等のコードを Alloy から直接生成できれば，プログラムの開発効率もあがり，ヒューマンエラーの予防も可能である．また，Alloy では，クラス図等の UML から Alloy 記述の生成に関する研究[7]が行われており，Alloy から生成可能な言語が増加すれば，形式仕様記述を用いたソフトウェア開発が現在よりも普及するのではないかと考えられる．

２．目的

本研究では，Alloy から C#コードを生成する機能がないことに着目し，Alloy で記述された仕様のうち，ソフトウェアの実装に関わる部分から C#コードを生成する手法の提案とコード生成ツールの開発を行う．

３．Alloy の概要

3.1　形式仕様記述言語　仕様の厳密な定義を行うために用いられる言語である．ソフトウェア開発では，開発したソフトウェアを仕様と比較して，開発者の意図通りに実装できているかをテストなどによって確認するが，仕様自体に矛盾や曖昧な点があれば，ソフトウェアが正しく動作しているか確認することが出来ない．そこで，仕様を形式仕様記述言語で厳密に定義することにより，仕様の矛盾や曖昧さを取り除く．本研究で用いる Alloy の他に，VDM や Z 記法，B メソッド，SPIN 等がある．

3.2　Alloy について　Alloy は形式仕様記述言語である．手続きの機能仕様やデータの静的な構造の表現を目的とする．基礎となるロジックとして 1 階関係論理を採用し，言語の意味定義を簡単化した．有限スコープ仮設を導入することにより，有限モデルに限定した範囲での解析で検証を行う．Alloy では，1 階関係論理に推移閉包オペレータを加えることにより再帰的な定義の取り扱いを可能にした．有限解析であるため限界はあるが，解析できない場合であっても仕様の厳密な定義等は可能である．

3.3　Alloy の構文要素　Alloy では主に，以下の構文要素を用いて仕様を記述する．

・open…他のモジュールの呼び出し
・module…モジュール名の宣言

・sig…集合，関係の定義
・fact…関係に対する制約の表現
・pred…動作を論理式で記述
・assert…検査する性質を記述
・check…指定した assert に対する反例を生成
・run…指定した制約を満たす例を生成

3.4　Alloy Analyzer　Alloy の記述を図式化，自動解析するツールである．AlloyAnalyzer の解析方法は図 1 の通りであり，Alloy の記述をコンパイルし CNF ファイル標準形式に変換後，充足性を SAT ソルバーで検証する．

図 1．Alloy Analyzer の仕組み

4．生成方法

4.1　概要　Alloy からの C#コード生成は，おおまかに以下の手順で行う．

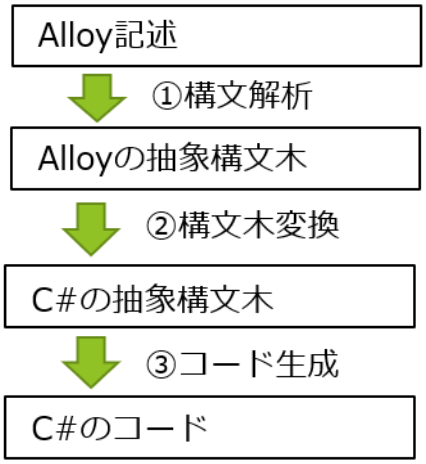

図 2．C#コード生成方法

最初に，Alloy 記述の構文解析を行い，Alloy の抽象構文木へ変換する．次に Alloy の抽象構文木を C＃の抽象構文木へ変換, 最後に C#の抽象構文木から C#コードの生成を行う．

4.2　Alloy 記述の構文解析　Alloy 記述から Alloy の抽象構文木を生成する処理である．Alloy Analyzer には，Alloy 記述を解析する機能は備わっているが，出力できないため，他の言語で扱えるファイルとして出力する機能の実装を行う．この時，Alloy 記述のうち，ソフトウェアの動作を制御するのに必要ない要素は取り除き，残った物から Alloy の抽象構文木を生成する．取り除くのは以下の要素である．

・assert

ソフトウェアの性質の検査を行う部分であり，ソフトウェアの動作には必要ない．但し，変換後のプログラムが所望の性質を満たしていることをテストする場合には有用であると考えられ，今後検討が必要である．

・check，run

ソフトウェアが所望の性質を満たしているのかを実際に検証する要素であり，ソフトウェアの動作には特に必要ないと考えられる．

また，今回のプログラムではパターンマッチを利用しており，処理を簡単にするために“this/”といった記述の削除も行っている．

4.3　C#の抽象構文木の生成　Alloy の抽象構文木から C#の抽象構文木へ変換する処理である．Alloy と C#では型や関数の定義に違いがある．変換前後でソフトウェアの動作が観測等価であるために，正確な変換を行う必要がある．4.5 で詳しく説明する．

4.4　C#コードの生成　C#の抽象構文木から C#コードを生成する処理である..引数の変数名からメソッドの返り値の型を推測する．変数名にアポストロフィが使われている変数（例：b’）は，事後変数を呼ばれるものであり，事後状態の変数が格納されている．そのため，今回は事後状態の変数の型を返り値の型にする．ない場合は Boolean 型を返り値の型にする．

4.5　抽象構文木の変換　Alloy は 3 章で記述したように，一階の関係論理をベースにしているテキストモデリング言語であり，sig(シグネチャ)や pred，fact，assert などから構成される．Alloy の各要素のうち，ソフトウェアの動作に必要な物を表１のように，対応する C#の要素へと変換する．

表１．変換対応表

Alloy	C#
sig	クラス
pred	メソッド
fun	メソッド
parameter	引数
field	フィールド
variable	フィールド

Alloy の sig は C#におけるクラスと同じであると考えられる．pred や fun ではプログラムの動作を記述するため，メソッドに変換するのが相応しい．但し pred に関しては，プログラムの処理ではなく制約を示している場合があり，その場合はメソッドには変換しない．Field や variable は Alloy 記述には直接は出てこず，C#におけるフィールドを表している記述の時に，抽象構文木に記述される．

各要素に含まれる論理式や文は全て C#の文へ変換する．変換について，一部を表 2 に示す．

表 2．変換対応表

Alloy	C#
open	using
extends	extends
call	関数呼び出し
set	Hushset
->	Dictionary
a => b	if(a){b}
Int	int
++	Add

Open は C#でにおける using と同様の動作を行う要素であり，他のモジュール（ライブラリ）の呼び出しを行う時に用いられる．継承に関しては Alloy，C#の両方とも extends を使用する．call は field や variable と同様，抽象構文木のみに記述される．“->”は，前後の関係を表すものであり，sig で用いる場合は Dictionary に変換するが，pred で用いられる場合は“++”等と共に使われることが多く，その場合直接変換は行わない．“a=>b”は，a ならば b を意味しており，if 文への変換が相応しいと考えられる．

表には載せていないが，基本的な四則演算（+, -, *, /）や比較（<, <=, >, >=），真偽値（true, false）に関しては手を加えず，C#コードで使用する．

５．生成ツールの概要

5.1　生成ツールについて　今回開発したツールは，以下の手順でコード生成を行う．

① Alloy Analyzer で表示可能な構文木から必要な情報を抜き出し，Alloy の抽象構文木としてテキストファイルで出力．
② ①で出力した抽象構文木を C#の抽象構文木に変換．
③ ②で生成した C#の抽象構文木から C#コードを生成．

①は，Alloy Analyzer はコードが公開されているため，4.1 で述べた機能を Alloy Analyzer に直接組み込んだ．

②と③は，.NET コンパイラプラットフォーム Roslyn を用いて C＃で開発した．

5.2　例１　今回開発したツールによるコード生成について具体例を挙げて説明する．

リスト 1．変換前（Alloy）

```
Sig Button
{
    type : Type,    //ボタンの種類
    width : Int,    //ボタンの幅
    height : Int    //ボタンの高さ
}
pred on_button (x : Int, y : Int, b : Button)
{
    x >= 0 && x   < b.width
    y >= 0 && y < b.height
}
pred mouse_pressL (x : Int, y : Int, b : Button, b' :
                                        Button)
{
    !on_button[x,y,b] => b'.type = Normal
    on_button[x,y,b] => b'.type = Press
    b'.width = b.width
    b'.height = b.height
}
fun GetType [b : Button] : Type
{
    b.type
}
```

リスト 2．変換後（C#）

```
class Button
{
    Type type;
    int width;
    int height;
}
Boolean On_button(int x, int y, Button b)
{
    return x>0&&x<b.width&&
                y>=0&&y<b.height;
}
Button Mouse_pressL(int x, int y, Button b,
                                    Button b')
{
    if(!On_button(x,y,b)
    {
        b'.type = Normal;
    }
    if(On_button(x,y,b)
    {
        b'.type = Press;
    }
    b'.width = b.width;
    b'.height = b.height;
    return b';
}
Type GetType(Button b)
{
    return b.type;
}
```

リスト 1 は変換前の Alloy のプログラムである．初めに Button という sig の定義を行っている．Button は type（ボタンの種類），width（ボタンの幅），height（ボタンの高さ）の 3 つを持つ．On_button は 3 つの引数を受け取り，int 型の引数が指定の範囲内に入っているか否かを Bool 型で返すメソッドである．Mouse_pressL は，On_button に引数を設定して呼び出し，false を返した場合は Button の Type を Normal に，true を返した場合は Press を返す．そして，返り値として Button を返すメソッドである．GetType は引数として受け取った Button の Type を返す．

リスト 2 は変換後のリスト 1 のプログラムである．分かりやすくするために，リスト 1 から直接変換された部分のみを記述している．4.5 の表 1 の通り，Alloy の sig は C#では class に変換する．pred はメソッドに変換するが，返り値の型は，引数のインスタンス名に’を持った物が有る場合は引数の型を返り値とする．ない場合は Boolean 型を返すとしており，On_button の場合は Boolean 型, Mouse_pressL の場合は Button 型が返り値になっている事が確認できる．Pred 内に記述されるプログラムの動作に着目すると，比較や代入は特に手を加えられずそのまま C#に変換されている事，“=>”が if 文に変換されている事が確認できる．また，fun も pred と同様にプログラムの動作を表す部分であるため，表 1 の通り，メソッドに変換されている．また，変換前のプログラムの変数名にあるアポストロフィに関しては，C#では使用できないため，アンダーバーに置き換えている．

5.2　例 2

リスト 3．変換前（Alloy）

```
sig BirthdayBook
{
      known : set Name,
      birthday : Name -> Date
}

pred AddBirthday (b,b' : BirthdayBook,
                                n : Name, d : Date)
{
      b'.birthday = b.birthday ++ n -> d
}
```

リスト 4．変換後（C#）

```
Class BirthdayBook
{
      Hushset<Name> known;
      Dictionary<Name，Date> birthday;
}
BirthdayBook AddBirthday(BirthdayBook b,
              BirthdayBook b_，Name n，Date d)
{
      b_.birthday = b.birthday;
      b_.birthday.Add(n，d);
      return b_;
}
```

リスト 3 は変換前の Alloy のプログラムである．初めに BirthdayBook という sig の定義を行っている．Name（名前）の集合と，Name と Date（日付）が関係づけられた birthday を持つ. AddBirthday は BirthdayBook 型の変数 2 つと Name，Date を引数に受け取り，BirthdayBook に Name と Date の追加を行う pred である．変数名に‘’がついてる変数は事後変数と呼ばれる．

リスト 4 は変換後のリスト 4 のプログラムである．表 2 にある通り，Set は Hushset に，“->”は Dictionary に変換されている事が確認できる．AddBirthday では，“++”は Add，“->”は特に変換されていないことが分かる．変換前のプログラムの“b’”が BirthdayBook 型であるため，返り値は BirthdayBook 型になっている．また，等号（=）は同じように C#でも使用できるため, 手を加えずそのまま C#に変換されている．

６．結言

Alloy 記述の他言語へのコード変換は JML 仕様の生成の研究[3]や，Alloy を用いた Java プログラムの開発に関する研究[4]は行われている．しかし，C#プログラムの生成に関しては報告されていない．Alloy 記述から実際に使われている開発言語の自動生成が出来れば，開発工程の増加等の欠点がなくなり，ソフトウェア開発への Alloy の導入も比較的容易になる．そこで本論文では，Alloy から C#コードの生成方法について提案を行い, プロトタイプを試作した．開発したツールは，①Alloy 記述の構文解析を行い，Alloy の抽象構文木へ変換，②Alloy の抽象構文木を C＃の抽象構文木へ変換，③C#の抽象構文木から C#コードの生成という方法により C#コード生成を行う．

Alloy 記述の構文解析については，現在の段階では出力された構文木に一切手を加えず C#の抽象構文木へ変換，コード生成を行うのは難しい．順番を並び替えるといった手動での処理が必要なくなるように今後改良していく必要がある．また，構文木変換に関して，Alloy では関係に対する制約を fact で表現するのだが，fact の変換は検討不足で実装できていない．fact の変換方法として考えられるのは, 制約を表現しやすい JML 等の言語に変換しコメントにして残す，.NET の codeconstracts を用いる等の方法が考えられる．また，if 文の変換をした際に出力後のコードが左に寄ってしまうため，Ctrl＋K，D を行い修正するという作業が必要になる点も今後の課題に挙げられる．

謝辞

本研究に際して，様々なご指導を頂きました西村俊二講師に深謝いたします．また，この研究の機会をくださった電気電子情報工学専攻の先生方に厚く御礼申し上げます．

参　考　文　献

1） T. Kurita: New Trends in Formal Methods: Application of a Formal Method in the Development of the “Mobile FeliCa” IC Chip Firmware for Embedding in Mobile Phone，2008

2） Philipp Berger，Joost-Pieter Katoen, Erika Abraham, Md

Tawhid Bin Waez, and Thomas Rambow: Verifying Auto-Generated C Coed from Simulink -An Experience Report in the Automotive Domain-, 22nd International Symposium FM 2018

3) 千坂　優佑，岡本圭史: VDM++仕様から C#コードを生成するツールの開発（情報処理学会第 78 回全国大会），2016.3.

4) Alloy Analyzer，http://alloytools.org/

5) Daniel Grunwald，Christoph Gladisch, Tianhai Liu, Mana Taghdiri, and Shumuel Tyszberowicz: Generating JML Specifications from Alloy Expressions，2014

6) Aboubakr Achraf El Ghazi, Mattias Ulbrich, Christoph Gladisch, Shmuel Tyszberowicz, and Mana Taghdiri: JKelloy: A Proof Assistant for Relational Specifications of Java Programs，2014

7) Kyriakos Anastasakis，Behzad Bordbar, Geri Georg, and Indrakshi Ray: UML2Alloy: A Challenging Model Transformation, 10th International Conference, MoDELS 2007