

ソフトウェアシンポジウム 2016

事例報告

VDM++仕様からC#コードを 生成するツールの開発と評価

仙台高等専門学校

千坂優佑 大友楓雅 力武克彰 岡本圭史

目次

1. はじめに
 - 背景、課題、アプローチ
2. コード生成の方法と結果
 - コード生成の方法
 - コード生成の結果の例
 - テストの結果
 - 従来開発との比較
3. まとめと課題

背景

- 形式手法
 - 数学的な理論に基づいた方法によって開発成果物の品質を向上させようとする手法の総称
- 形式仕様記述
 - 仕様書を形式言語で記述することで仕様の厳密化や計算機による検証を可能とする手法
- VDM
 - いわゆる「ライトウェイトな形式手法」
 - 事前・事後・不変条件を用いて仕様を記述
 - FeliCaの開発や宇宙開発等で利用されている^{[1][2]}

[1] 栗田太郎, 携帯電話組込み用モバイルFeliCa IC チップ開発における形式仕様記述手法の適用, 情報処理, Vol. 49, No. 5, pp. 506.513, 2008.

[2] JAXAにおける形式手法に対する取組み～ソフトウェアIV&Vにおける形式手法の活用事例～,
<http://cfv.jp/cvs/event/workshop/2012/09/pdf/jaxa.pdf>.

背景

- VDM++
 - VDMの記法の1つ
 - オブジェクト指向 + 関数型 (+ 手続き型)
 - 仕様を事前・事後・不変条件として記述可能
 - 構文チェックや型チェックによる仕様の検証が可能
- 自動コード生成
 - VDM仕様からJavaやC++のコードを生成する機能
 - VDMTools^[3]、Overture^[4]で利用可能

[3] SCSK, VDM information web site, <http://www.vdmttools.jp/>.

[4] Overture Tool, <http://overturetool.org/>

背景

- CodeContracts [5]
 - .NET Framework言語で事前・事後・不変条件を記述
 - 静的検証の実施、テストの強化

The screenshot shows a Visual Studio window with a CodeContracts project. The main window displays the source code for `Test.cs` with the following content:

```
12 public int Sum(int n)
13 {
14     return Range(1, n).Sum();
15 }
16 public List<int> Range(int start, int count)
17 {
18     Contract.Requires(count >= 0);
19 }
```

Annotations and callouts include:

- A red box around `count` in `Range(int start, int count)` with the text "引数countは0以上".
- A red box around `Contract.Requires(count >= 0);` with the text "呼び出し".
- A red box around `n` in `Sum(int n)` with the text "nは0以上であるべき".
- A blue box with a warning icon and text: "1 CodeContracts: Missing precondition in an externally visible method. Consider adding `Contract.Requires(n >= 0);` for parameter validation".
- A blue box with an info icon and text: "2 CodeContracts: Checked 8 assertions: 7 correct 1 unknown".

Below the code, a table titled "静的検証" (Static Verification) shows test cases:

n	result(target)	result
0		0
1		1
2		3
7		28

A red box around the row where `n=1` and `result=1` has the text "自動生成されたテストケース" (Automatically generated test case).

At the bottom, the text "テストケース生成ツール連携" (Test case generation tool integration) is displayed.

[5] Microsoft Research, Contracts, <http://research.microsoft.com/en-us/projects/contracts/>.

関連研究

- B. Meyer: Design by Contract, Technical Report (1986)
 - 事前・事後・不変条件で仕様を記述する
 - 設計からコーディング、検証まで利用可能
- Mohammadsadegh Dalvandi: From Event-B Models to Dafny Code Contracts (2015)
 - Event-BモデルをDafny(検証向けプログラミング言語)に変換
 - 形式仕様を検証可能コードに変換する研究は非常に少ない
- Néstor Cataño: A Machine-Checked Proof for a Translation of Event-B Machines to JML (2013)
 - Event-BからJMLへの変換の正しさの証明

課題とアプローチ

- 目的: 事前・事後条件を後工程まで利用可能に
 - 課題: コードに対する事後条件の検証ができない
 - 方針: .NETのCodeContractsを利用
 - 実装後の検証に事前・事後条件が利用可能
 - 操作定義の事後条件も記述可能
- 目的: 自動コード生成の対応言語を増やす
 - 課題: 既存ツールではJavaとC++のみ生成可能
 - 方針: VDM++仕様^[6]からC#コード^[7]を生成するツール作成
 - プロトタイプをC#で実装した

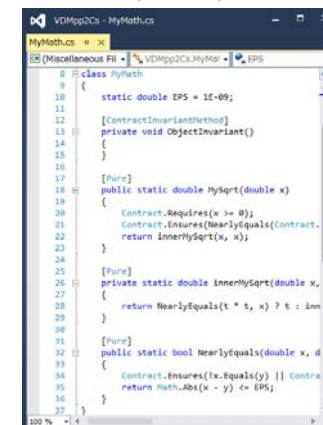
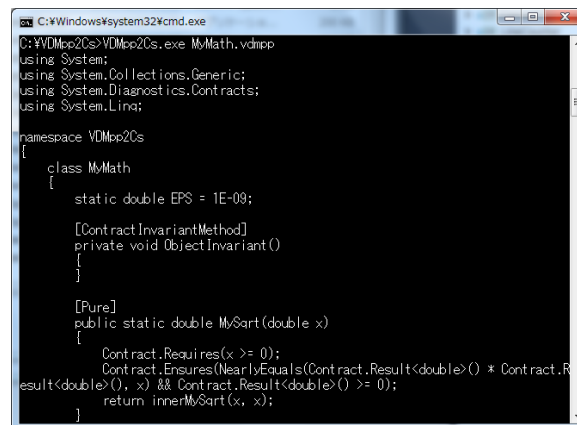
[6] SCSK, The VDM++ Language Manual, http://www.vdmttools.jp/uploads/manuals/langmanpp_a4E.pdf.

[7] ISO/IEC 23270:2003 C# Language Specification.

本ツールの利用イメージ

記述、検証(VDMTools、Overture)

実装、検証(CodeContracts)



コード生成の方法

- 大きく3段階に分けて処理する
 1. 構文解析 (VDM++仕様→VDM++抽象構文木)
 - 実装: 既存ツールVDMJ [\[8\]](#) の一部を利用
 - VDMJはJava製なのでC#に変換して利用
 2. 変換 (VDM++抽象構文木→C#抽象構文木)
 - 実装: 要素ごとに変換
 3. コード生成 (C#抽象構文木→C#コード)
 - 実装: コンパイラプラットフォームRoslyn [\[9\]](#) を利用

[\[8\] Nick Battle, VDMJ, https://sites.google.com/site/nickbattle.](https://sites.google.com/site/nickbattle)

[\[9\] GitHub, dotnet/roslyn, https://github.com/dotnet/roslyn.](https://github.com/dotnet/roslyn)

コード生成の方法

- VDM++の構文要素に対応するC#の要素を定める

対応の例

VDM++	C#
式	式
加算	加算
forall	All(LINQ)
文	文
代入文	代入文
let文	変数宣言と式
型	型
seq of T	List<T>

VDM++	C#
クラス	クラス
型定義	クラス
不変条件	CodeContracts
値定義	フィールド
関数定義	メソッド
事前条件	CodeContracts
事後条件	CodeContracts
操作定義	メソッド
インスタンス変数定義	フィールド

プロトタイプの実装

- コード生成ツールのプロトタイプを実装した
- VDM++構文要素の変換の実装範囲
 - クラスと型・値・関数・操作・インスタンス変数定義
 - 事前・事後・不変条件
 - pre・post・invをContract.Requires・Ensures・Invariantに変換
 - VDM++、CodeContractsのいずれも条件の記述は式
 - 従来ツールにはない操作の事後条件の旧名称の変換が可能
 - if式、forall(集合束縛)等多くの式と代入文等一部の文
- 変換の未実装範囲
 - forall(型束縛)は無限の要素を扱うため検証できない
 - パターンマッチやトークン等は未実装

コード生成結果の例

VDM++仕様

```
class Counter
instance variables
  public Count : nat;
operations
  public Counter :
    () ==> Counter
  Counter() == (
    Count := 0;
  )
  post Count = 0;
  public Add : () ==> ()
  Add() == (
    Count := Count + 1;
  )
  post Count = Count~ + 1;
end Counter
```

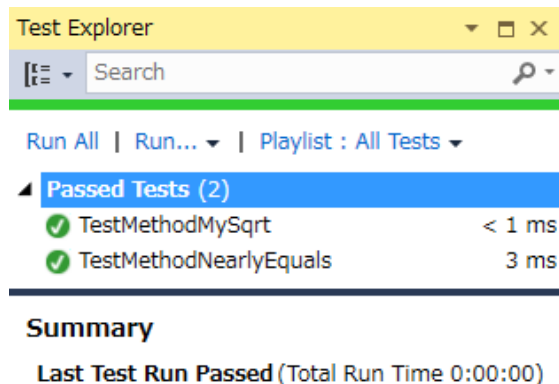
生成されたC#コード

```
class Counter {
  public uint Count;
  public Counter() {
    Contract.Ensures(
      Count.Equals(0));
    Count = 0;
  }
  public void Add() {
    Contract.Ensures(
      Count.Equals(Contract
        .OldValue(Count) + 1));
    Count = Count + 1;
  }
}
```

操作の事後条件も変換される

変換の妥当性確認

- VDM++仕様とC#コードに対してテストを実施
- 同一のテストケースに対して同一のテスト結果
 - 式(関数)や文(操作)の変換は妥当だと考えられる
- 事前・事後・不変条件の中身は式
 - 式の変換が正しければこれらも正しいと考えられる



```
**
** Overture Console
**
Start test - MyMath`NearlyEquals
End test   - MyMath`NearlyEquals
Start test - MyMath`MySqrt
End test   - MyMath`MySqrt
Succeed
TestRun`run() = ()
```

従来開発との比較

- 本ツールの有用性評価のため開発工程を比較
 - 開発対象: 3つのクラスから成る簡単なシステム

	(A)従来開発	(B)本ツールを利用
実施者のレベル	小規模の実用的なツールを設計・開発できる	(A)と同様 + 研究でVDM++を利用している
仕様記述言語	UML	VDM++
仕様の検証	レビュー	ツールによるチェック
実装(C#)	人手による実装	自動コード生成
実装の検証	テスト	テスト(事前・事後・不変条件利用)

- 各工程の時間や実施者が気づいたことを記録

従来開発との比較結果

	(A)従来開発	(B)本ツールを利用
モデル作成の時間	30分未満	1時間未満(VDM++からUMLを生成可能)
コード作成の時間	モデルからスケルトンコードの作成に15分(3クラス)	数秒程度
	→ コード作成の工数削減	
テストケース作成	事前条件を満たさないテストケースも作成	事前条件を満たさないテストケースは除外
	→ 有用なテストケースに集中できる	
仕様の検証	テスト段階で仕様の漏れ発見	VDM++仕様により漏れ発見
	→ 不具合の早期発見	

まとめと課題

- まとめ: C#コードを生成するツールを開発
 - VDM++仕様からC#コードを自動生成
 - CodeContractsで事前・事後・不変条件もC#コードに保持
 - 後工程の静的検証やテストに利用可能
- まとめ: 変換の妥当性とツールの有用性の評価
 - VDM++仕様と生成したC#コードでテストの結果が同一
 - モデルからコードを作成する期間削減
 - 事前・事後・不変条件によりテストケースが改善
- 課題: 変換可能なVDM++構文要素の拡充
 - 未実装の要素を実装しツールの適用可能範囲を拡大
- 課題: モデルの記述コストを含めた評価