A prototype of generating a decision table from formal specification and its application examples

Oct./31/2014

Kenta Nishikawa*, Tetsuro Katayama*

*University of Miyazaki
Agenda

1. Background
2. Goal
3. A prototype of a decision table generation tool
4. A process to generating a decision table of the prototype
5. Application examples
6. Discussion
7. Conclusion
8. Future issues
Background

Interests in the formal methods and test techniques to realize a high reliability system is increasing.

The testing with the formal methods is few reports.

- It’s difficult to introduce the formal methods into software process.
- The advantage of the testing with the formal methods is not clear.

It’s needed to clarify the advantage of the testing with the formal methods.
Issue

To clarify the advantage of the testing with the formal methods

Implement a prototype of a decision table generation tool
- This prototype generates a decision table from the specification described in VDM++, which is one of formal specification languages.

This prototype performance was unknown
- It was necessary for us to evaluate this prototype performance
Performance evaluation the prototype of a decision table generation tool

We applied some general specifications to the prototype

These specifications are selected from examples of the textbook
A prototype of a decision table generation tool

Displays conditions and actions in a decision table form from the specifications of VDM++, on GUI.

- The decision table is available for design of test cases
- We can remove individualistic characteristics
- We can grasp the complicated logic relations
- We reduce a high threshold to introduce the formal methods
A process to generating a decision table of the prototype

- VDM++ Spec
- Decision Table

Parser

Internal expression data for analysis

Pre condition

Post condition

Module

Conversion

Datos de expresión interna para análisis

Condición pre

Condición post

Módulo

Conversión

Actions extraction

Condición de extracción

Truth-values generation

Generación de verdades
- Analysis result of a VDM++ specification which a user inputs to Parser
  - We use a parser of Overture Toolset as a Parser
  - The parsing data has an abstract syntax tree and tokens

```java
new OmlFunctionDefinitions(
    new OmlFunctionDefinition(
        new OmlAccessDefinition(
            ...
        new OmlLexem(19,5,375,"public",1),
        new OmlLexem(19,12,433,"FizzBuzz Spec",2),
        new OmlLexem(19,23,58,:",0),
        new OmlLexem(19,23,58,"nat1",1),
        ...

Abstract syntax tree (AST)

Tokens
```
Internal expression data for analysis

- Information suitable for analysis such as the division of a module or the correspondence of "if" and "else"

```java
if1
BinaryExpression(
    Name("s",)
BinaryOperator(27,11,10),
SeqEnum@
    then
NumericLiteral(0,11,20),
else1
BinaryExpression(
```

```java
new OmlFunctionDefinitions([
    new OmlFunctionDefinition(
        new OmlAccessDefinition(
            new OmlLexem(19,5,375,"public",1),
new OmlLexem(19,12,433,"FizzBuzz Spec",2),
new OmlLexem(19,23,58,":",0),
new OmlLexem(19,25,353,"nat1",1),
```
Extraction of conditions and actions

- We can extract the conditions from branch condition of the specifications such as "if" syntax or "cases" syntax.
- The associations between conditions and actions:
  
  IF “Condition” THEN “Action”
  ELSE IF “Condition” THEN “Action”
  ELSE “Action”
  CASE “Condition” \(\rightarrow\) “Action”
  OTHERS “Action”


**Extraction rules**

- **DT-Generator stores conditions and actions in an array of String type**

<table>
<thead>
<tr>
<th>Pattern of condition extraction</th>
<th>Pattern of action extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>if &quot;condition&quot; then elseif &quot;condition&quot; then</td>
<td>then &quot;action&quot; elseif then &quot;action&quot; else else &quot;action&quot; if else &quot;action&quot; elseif else &quot;action&quot; else else &quot;action&quot; EOF</td>
</tr>
<tr>
<td>cases &quot;condition&quot; -&gt;</td>
<td>-&gt; &quot;action&quot; cases others &quot;action&quot; EOF</td>
</tr>
<tr>
<td>pre &quot;condition&quot; post pre &quot;condition&quot; EOF post &quot;condition&quot; EOF</td>
<td></td>
</tr>
</tbody>
</table>

※EOF(End Of File)
CA-Table

DT-Generator makes CA-Table, when DT-Generator extracts conditions and actions

<table>
<thead>
<tr>
<th>Condition index</th>
<th>Token</th>
<th>Action index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>elseif</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>elseif</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>else1</td>
<td>2</td>
</tr>
</tbody>
</table>

class Sample

functions

revBin2dec : seq of nat -> nat

revBin2dec( s ) ==
if s=[] then 0
elseif s=[0] then 0
elseif s=[1] then 1
else hd s + 2*revBin2dec( tl s );

end Sample

Condition array

<table>
<thead>
<tr>
<th>Index</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s = []</td>
</tr>
<tr>
<td>1</td>
<td>s = [0]</td>
</tr>
<tr>
<td>2</td>
<td>s = [1]</td>
</tr>
</tbody>
</table>

Action array

<table>
<thead>
<tr>
<th>Index</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>hd s + 2*revBin2dec( tl s )</td>
</tr>
</tbody>
</table>
Truth-values generating process

1. Make an array to store truth-values
2. Select the first column of this array
3. Select a row of CA-Table
4. Compare a token of the selected row of CA-Table
   a. If this token matches “if”, “elseif”, or “cases”
      i. Store “Y” into the condition index row of this column, then store “N” from the next column to the last column
      ii. Store “X” into the action index row of this column
   b. If this token matches “else”, or “others”
      i. Store “N” into the condition index row of this column, then store “ー” from the next column to the last column
      ii. Store “X” into the action index row of this column
5. If there is a row that we have not yet selected, we select the next column of this array and return to third step. Otherwise truth-values is filled
Truth-values generating process

1. Make an array to store truth-values
2. Select the first column of this array
3. Select a row of CA-Table

Two-dimensional array of String type

<table>
<thead>
<tr>
<th>Condition index</th>
<th>Token</th>
<th>Action index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>elseif</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>elseif</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>else1</td>
<td>2</td>
</tr>
</tbody>
</table>

condition size + number of false(else, others)
Truth-values generating process

4. Compare a token of the selected row of CA-Table
   a. If this token matches “if”, “elseif”, or “cases”
      i. Store “Y” into the condition index row of this column, then store “N” from the next column to the last column
      ii. Store “X” into the action index row of this column

<table>
<thead>
<tr>
<th>Condition index</th>
<th>Token</th>
<th>Action index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>elseif</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>elseif</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>else1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>N</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. If there is a row that we have not yet selected, we select the next column of this array and return to third step

<table>
<thead>
<tr>
<th>Condition index</th>
<th>Token</th>
<th>Action index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>elseif</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>elseif</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>else1</td>
<td>2</td>
</tr>
</tbody>
</table>

```
0 | Y | N | N | N
---|---|---|---|---
   | --|--|--|--
   | --|--|--|--
   | --|--|--|--

1 | X | --|--|--
2 | --|--|--|--
   | --|--|--|--
   | --|--|--|--
   | --|--|--|--
```
# Truth-values generating process

4. Compare a token of the selected row of CA–Table
   
   a. If this token matches “if”, “elseif”, or “cases”
      
      i. Store “Y” into the condition index row of this column, then store “N” from the next column to the last column
      
      ii. Store “X” into the action index row of this column

<table>
<thead>
<tr>
<th>Condition index</th>
<th>Token</th>
<th>Action index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>elseif</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>elseif</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>else1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Y</th>
<th>N</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>X</th>
<th>—</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Compare a token of the selected row of CA-Table
   a. If this token matches “if”, “elseif”, or “cases”
      i. Store “Y” into the condition index row of this column, then store “N” from the next column to the last column
      ii. Store “X” into the action index row of this column
Truth-values generating process

4. Compare a token of the selected row of CA-Table
   b. If this token matches “else”, or “others”
      i. Store “N” into the condition index row of this column, then store “—” from the next column to the last column
      ii. Store “X” into the action index row of this column

<table>
<thead>
<tr>
<th>Condition index</th>
<th>Token</th>
<th>Action index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>elseif</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>elseif</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>else</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Y</th>
<th>N</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>—</td>
<td>—</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>X</th>
<th>—</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td>—</td>
<td>X</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>X</td>
</tr>
</tbody>
</table>
Truth-values generating process

5. Otherwise truth-values is filled

<table>
<thead>
<tr>
<th>Condition index</th>
<th>Token</th>
<th>Action index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>elseif</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>elseif</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>else1</td>
<td>2</td>
</tr>
</tbody>
</table>
Overview

D-Tree

FS-Screen

DT-Panel

Sample
- summation
- NatSeqSum
- bin2dec
- revBin2dec
- rev
- Add01
- length

```
001 class Sample
002
003 functions
004 summation( n:nat, a:seq of nat) s:nat
005 pre (n = len a) and (n>=1)
006 post s = NatSeqSum( a );
007
008 NatSeqSum: (seq of nat) -> nat
009 NatSeqSum( s ) ==
010 cases s :
011   [ ] -> 0,
012   others -> hd s + NatSeqSum( tl s )
013 end;
014
015 bin2dec : seq of nat -> nat
016 bin2dec(s) == ...
```
DT-Panel

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y (Yes)</td>
<td>The condition is satisfied</td>
</tr>
<tr>
<td>N (No)</td>
<td>The condition is not satisfied</td>
</tr>
<tr>
<td>-</td>
<td>The condition is no relation or is impossible logically</td>
</tr>
<tr>
<td>X (eXecute)</td>
<td>The action is executed</td>
</tr>
<tr>
<td>-</td>
<td>The action is not executed</td>
</tr>
</tbody>
</table>

{P} : pre condition tab
A : module tab
{Q} : post condition tab
D-Tree

Select any of the definition names

draw

redraw
Application examples

These specifications are selected from examples of the textbook, which can describe in VDM++

1. Sum of n natural numbers
2. Multiplication
3. Division
4. Factorial
5. Greatest common divisor
6. Raise
7. Inner product
8. Convert decimal to binary
9. Square root
10. Palindrome
11. Minimum subsequence sum
12. Maximum increase subsequence
13. Magic square
14. 8-Queen
class Dec2bin
functions

    dec2bin( X : nat ) b : seq of nat
pre  X >= 1

    post (forall i in set inds b & b(i)=1 or b(i)=0) and
    b(1) = 1 and
    X = NatSeqSum( [ b(i)*2**(len b - i) | i in set inds b ] );

NatSeqSum : seq of nat -> nat
NatSeqSum( s ) ==
    cases s :
        [] -> 0,
        others -> hd s + NatSeqSum( tl s )
    end;

end Dec2bin

✓ Stores it in binary expression array v of a positive integer X
✓ However, the highest-order digit become one

X = 25

\[
\begin{array}{c}
1 \\
1 \\
0 \\
0 \\
1 \\
\end{array}
\]

Expression of the partial legitimacy (textbook sec.4.1 p.73)
class Dec2bin
functions

dec2bin( X : nat ) b : seq of nat
pre X >= 1
post (forall i in set inds b & b(i)=1 or b(i)=0) and 
       b(1) = 1 and 
       X = NatSeqSum( [ b(i)*2**(len b - i) | i in set inds b ] );

NatSeqSum : seq of nat -> nat
NatSeqSum( s ) ==
  cases s :
    [] -> 0,
    others -> hd s + NatSeqSum( tl s )
edn;
end Dec2bin
We classified the states of the generated decision table into three types.

This prototype could generate the decision table:

\[
\frac{(5 + 5)}{14} \times 100 \approx 71.4\%
\]

Why? “A−” and “F”:
- (A−) Complex condition
- (F) Non-correspondence syntax
Non-correspondence syntax

- This prototype doesn’t support to all syntax in VDM++
  - (let, for, while, invariance condition etc..)
- This prototype can’t extract all conditions and actions
- This problem is caused by the conversion failure of the internal expression data for analysis from the parsing data

Solution

- It could be resolved in future by the prototype improvement
Complex condition

- Combined more than one condition with the logical operator

Form of the decision table which our prototype generates

<table>
<thead>
<tr>
<th>Rules</th>
<th>#1</th>
<th>#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Year mod 4 = 0) and (Year mod 100 = 0) and (Year mod 400 = 0)</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Actions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>X</td>
<td>–</td>
</tr>
<tr>
<td>29</td>
<td>–</td>
<td>X</td>
</tr>
</tbody>
</table>

- This prototype just extracts the description of the specifications
- This prototype doesn’t divide the compound conditions into a simple
- We will not usually write it in this way, manually
Complex condition

- Should be divided into a simple condition

Divided the decision table into a simple condition

<table>
<thead>
<tr>
<th>Rules</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Year mod 4 = 0)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>(Year mod 100 = 0)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>-</td>
</tr>
<tr>
<td>(Year mod 400 = 0)</td>
<td>Y</td>
<td>N</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>29</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

Solution

- We rethink an extraction rule, and improve CA-Table
- Correspondence is not clear at present
Conclusion

Issue & Approach
To clarify the advantage of the testing with the formal methods
- Implemented a prototype of a decision table generation tool

Goal & Approach
Performance evaluation of a decision table generation tool
- Applied 14 specifications to the prototype

Result
This prototype generated the decision table from the applied specifications of approximately 70%
- We reduce a high threshold to introduce the formal methods by using the prototype
- The testing with the formal methods is advanced
Future issues

- The usefulness improvement of the decision table generation tool

- The application of large-scale system specifications
  - The book “Validated Designs for Object-Oriented Systems”
  - ATM example

- Automatic generation of the test data
  - variable, type of input value

- Application to other test design techniques
  - equivalence analysis, boundary value analysis