Automated Generation of Decision Table and Boundary Values from VDM++ Specification

Hiroki Tachiyama*, Tetsuro Katayama*
and Tomohiro Oda**
* University of Miyazaki, Japan
**Software Research Associates, inc. , Japan

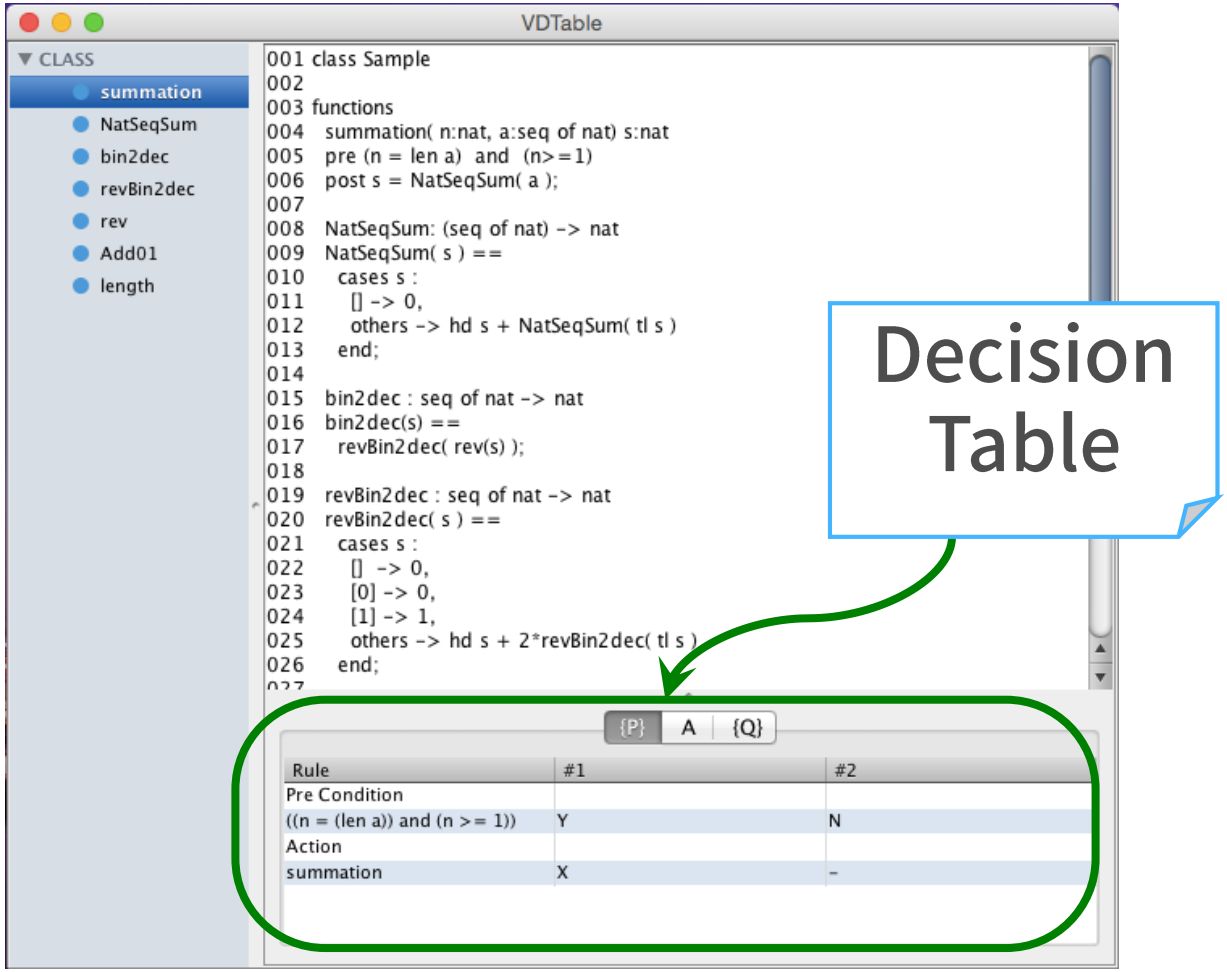
The 15th Overture Workshop 15, September 2017

agenda

1. background
2. VDTable
3. BWDM
4. Conclusion

VDM + Decision Table = VDTable

VDM++ Specification



an example of decision table

class name : Sample						
function name : SampleFunction						
		#1	#2	#3	#4	
Condition	a < 5	T	T	F	F	1
Condition	12 <= a	T	F	T	F	
Action	"a < 5"	T	T	F	F	2
Action	"12 <= a"	F	F	T	F	
Action	"5 <= a < 12"	F	F	F	T	

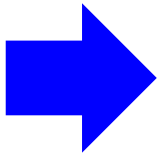
3

1. Condition part – Describe the conditional expressions in software and the possible combinations of these Boolean values
2. Action part – Describe the operation in the software and the action on the truth value of the condition part
3. Rule part – Describe Behavior of software between condition part and action part

Boundary Value + Vienna Development Method = BWDM

VDM++ Specification

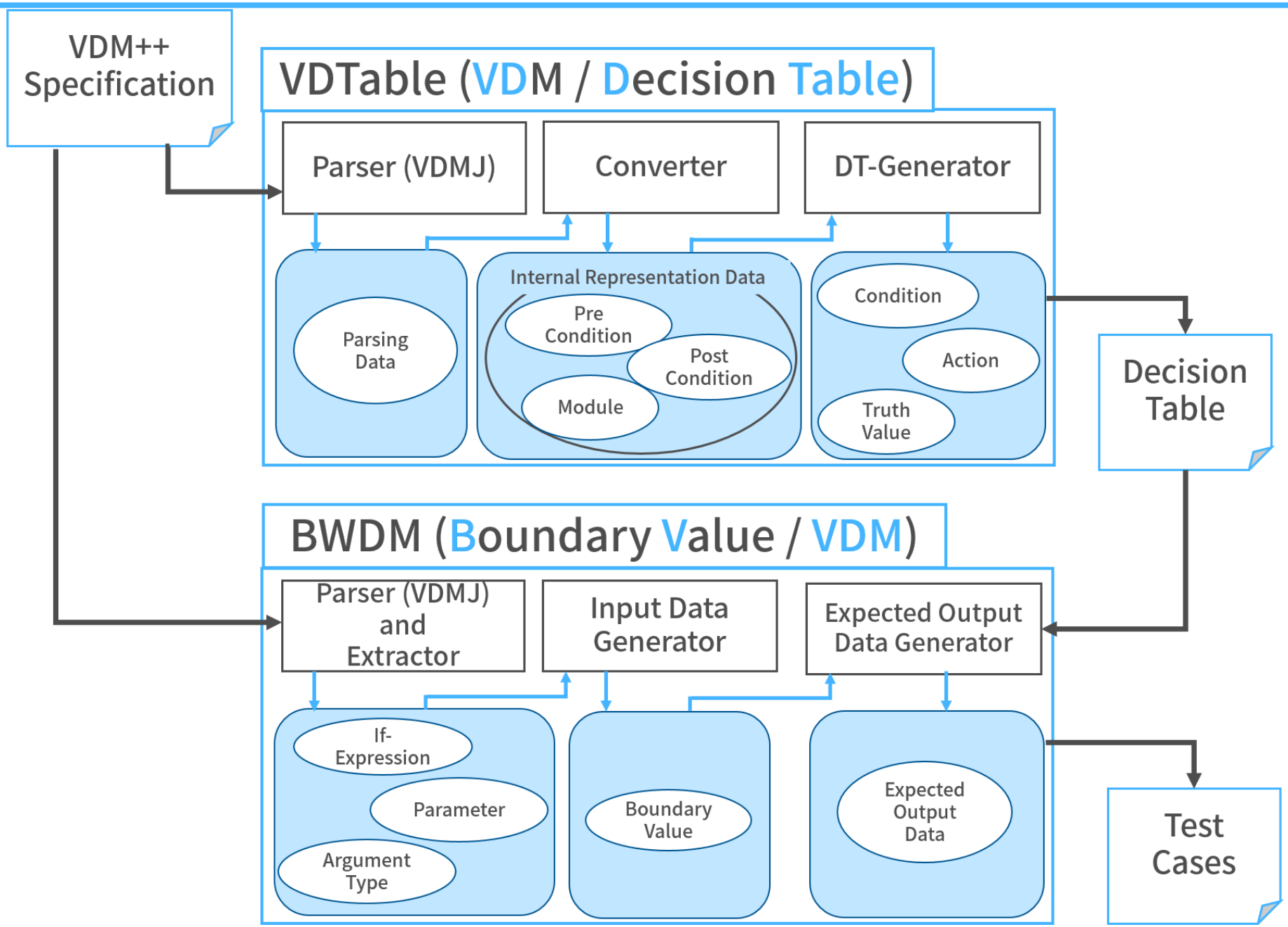
Decision Table

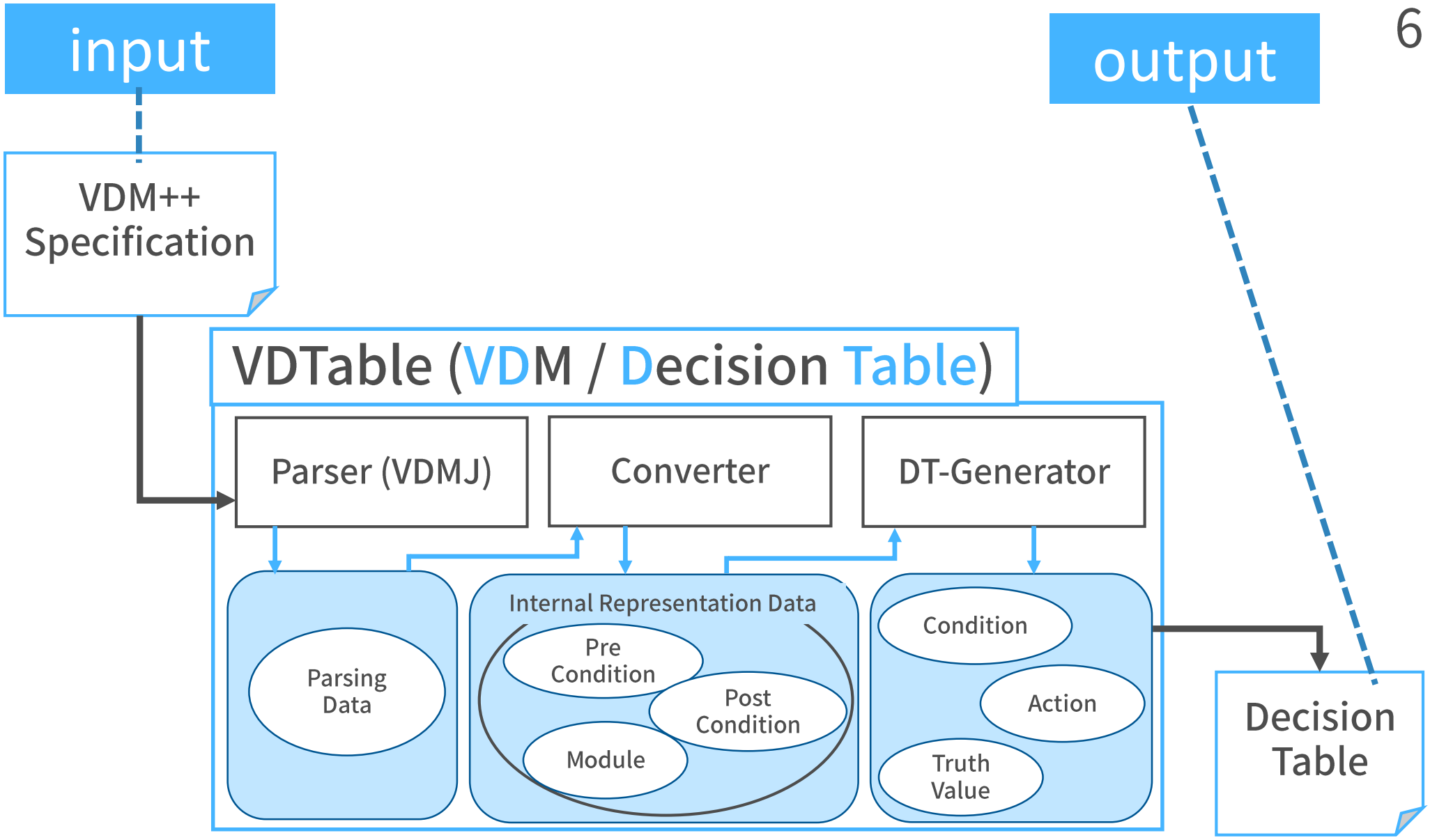


Test Cases

No.6	natMin-1	-1	-->	Undefined Action
No.7	natMin	intMin-1	-->	Undefined Action
No.8	natMin	intMin	-->	arg1:even arg2:negative
No.9	natMin	intMax	-->	arg1:even arg2:positive
No.10	natMin	intMax+1	-->	Undefined Action
No.11				arg1:even arg2:positive
No.12				arg1:even arg2:negative
No.13				Undefined Action
No.14				arg1:odd arg2:negative
No.15				arg1:odd arg2:positive
No.16				Undefined Action
No.17				arg1:odd arg2:positive
No.18				arg1:odd arg2:negative
No.19				Undefined Action
No.20	natMax+1	intMin	-->	Undefined Action
No.21	natMax+1	intMax	-->	Undefined Action
No.22	natMax+1	intMax+1	-->	Undefined Action
No.23	natMax+1	0	-->	Undefined Action
No.24	natMax+1	-1	-->	Undefined Action
No.25	2	intMin-1	-->	Undefined Action
No.26	2	intMin	-->	arg1:even arg2:negative
No.27	2	intMax	-->	arg1:even arg2:positive
No.28	2	intMax+1	-->	Undefined Action
No.29	2	0	-->	arg1:even arg2:positive
No.30	2	-1	-->	arg1:even arg2:negative
No.31	1	intMin-1	-->	Undefined Action
No.32	1	intMin	-->	arg1:odd arg2:negative

overview of two tools



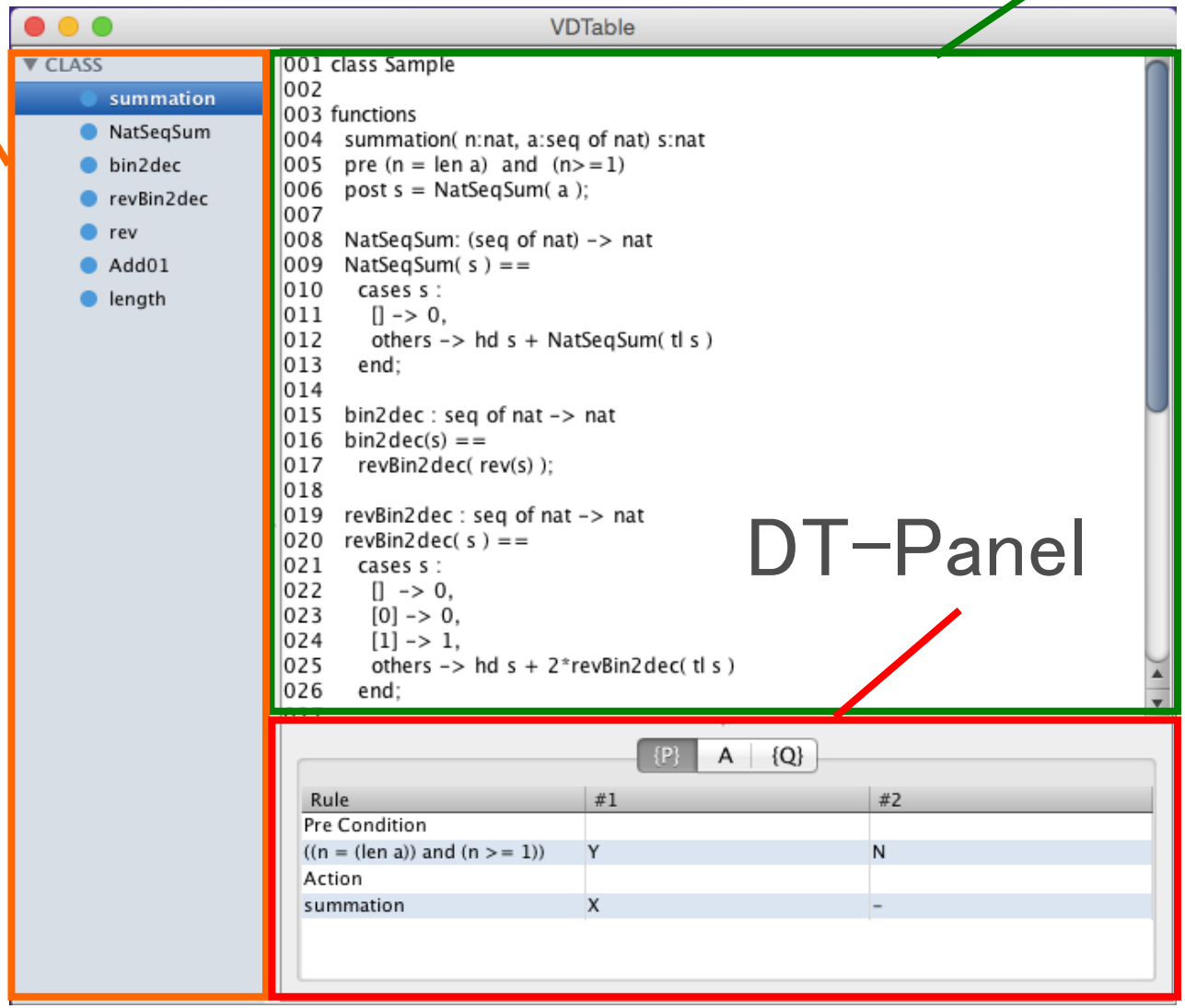

input
output
VDM++
Specification
VDTable (VDM / Decision Table)
Parser (VDMJ)
Converter
DT-Generator
Parsing
Data
Internal Representation Data
Pre
Condition
Post
Condition
Module
Condition
Action
Truth
Value
Decision
Table

appearance of VDTable

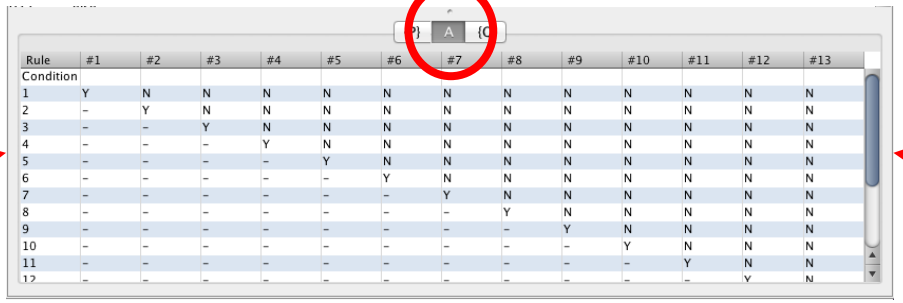
D−Tree

VS−Screen

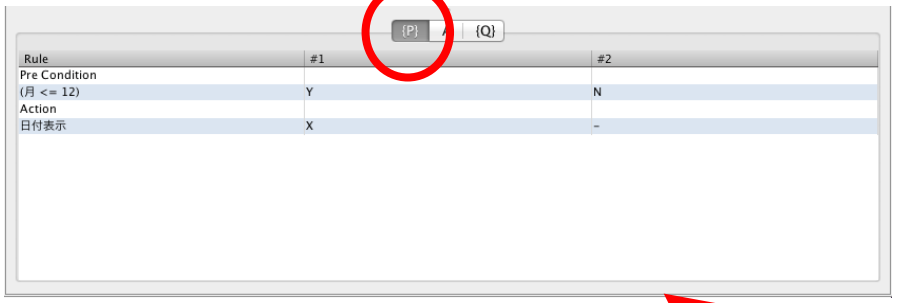
DT−Panel



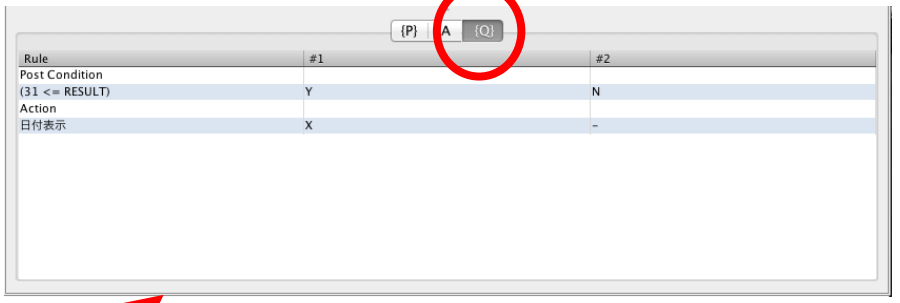
A：module tab

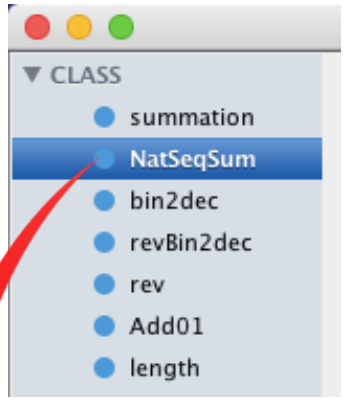


{P}：pre condition tab



{Q}：post condition tab



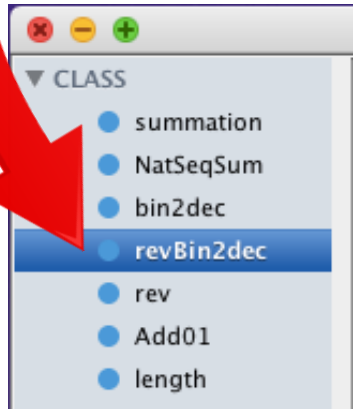


draw

{P} A {Q}

Rule	#1	#2
Condition		
[]	Y	N
Action		
0	X	-
((hd s) + NatSeqSum((tl s)))	-	X

click any definition



re-draw

{P} A {Q}

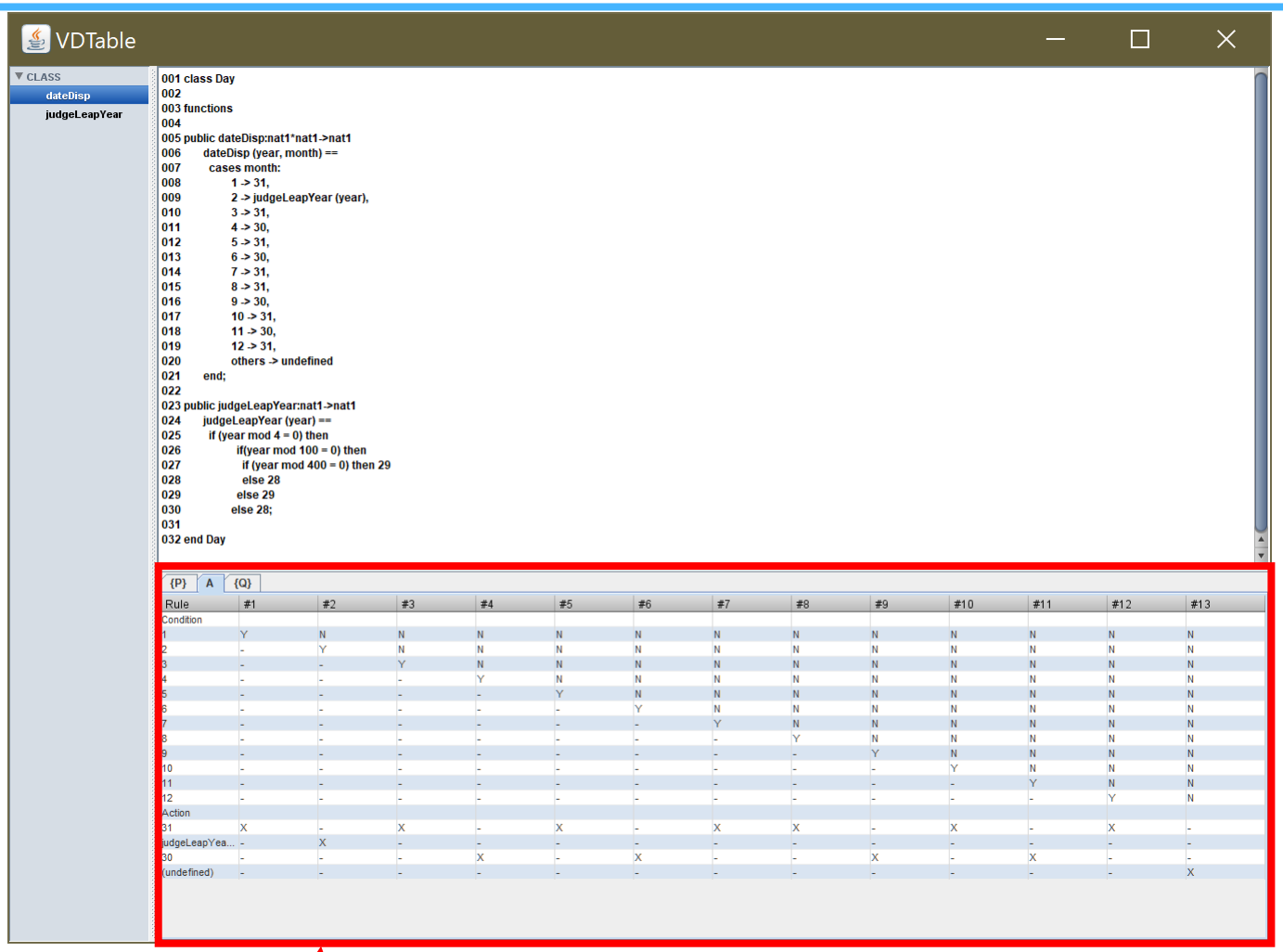
Rule	#1	#2	#3	#4
Condition				
[]	Y	N	N	N
[0]	-	Y	N	N
[1]	-	-	Y	N
Action				
0,	X	X	-	-
1	-	-	X	-
((hd s) + (2 * revBin2dec((tl s))))	-	-	-	X

```
class Day
functions
public dateDisp: nat1 * nat1 -> nat1
  dateDisp(year, month) ==
    cases month:
    1 -> 31,
    2 -> judgeLeapYear (year),
    3 -> 31,
    4 -> 30,
    5 -> 31,
    6 -> 30,
    7 -> 31,
    8 -> 31,
    9 -> 30,
    10 -> 31,
    11 -> 30,
    12 -> 31,
    others -> undefined
    end;
                    ...
```

Day specification

- ✓Inputs are year and month.
- ✓Return value is the date at the end of the month of the input.
  year / month.
- ✓If it was entered a month other than January to December, the return value is an undefined expression.
- ✓condition：12
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
- ✓action：4
  - 31 judgeLeapYear (year), 30, undefined

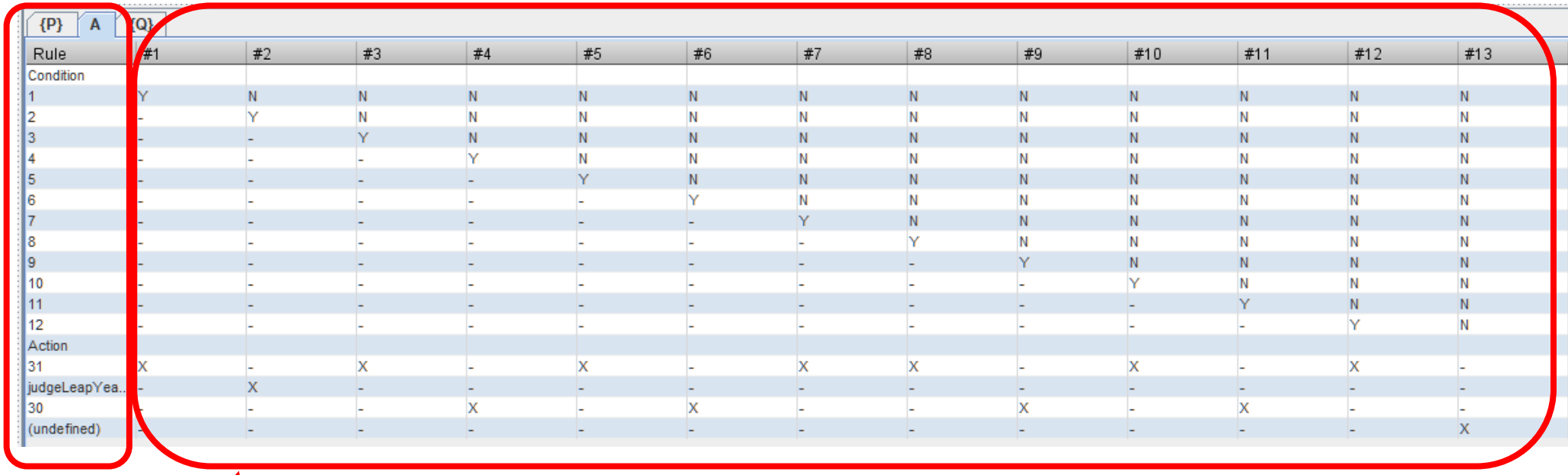
Application result



generated decision table (DT-Panel)

Application result

12 conditions and 4 actions (all conditions and actions) are extracted.



{P} A {Q}

Rule	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13
Condition													
1	Y	N	N	N	N	N	N	N	N	N	N	N	N
2	-	Y	N	N	N	N	N	N	N	N	N	N	N
3	-	-	Y	N	N	N	N	N	N	N	N	N	N
4	-	-	-	Y	N	N	N	N	N	N	N	N	N
5	-	-	-	-	Y	N	N	N	N	N	N	N	N
6	-	-	-	-	-	Y	N	N	N	N	N	N	N
7	-	-	-	-	-	-	Y	N	N	N	N	N	N
8	-	-	-	-	-	-	-	Y	N	N	N	N	N
9	-	-	-	-	-	-	-	-	Y	N	N	N	N
10	-	-	-	-	-	-	-	-	-	Y	N	N	N
11	-	-	-	-	-	-	-	-	-	-	Y	N	N
12	-	-	-	-	-	-	-	-	-	-	-	Y	N
Action													
31	X	-	X	-	X	-	X	X	-	X	-	X	-
judgeLeapYea...	-	X	-	-	-	-	-	-	-	-	-	-	-
30	-	-	-	X	-	X	-	-	X	-	X	-	-
(undefined)	-	-	-	-	-	-	-	-	-	-	-	-	X

Truth values are properly generated.

We confirmed that the VDTable operates correctly.

Subject Experiment

VDTable : decision table

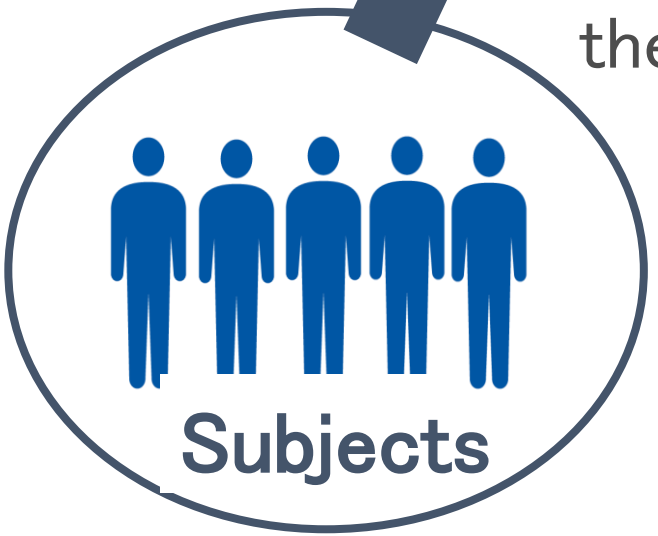
The number of combination of truth values is different

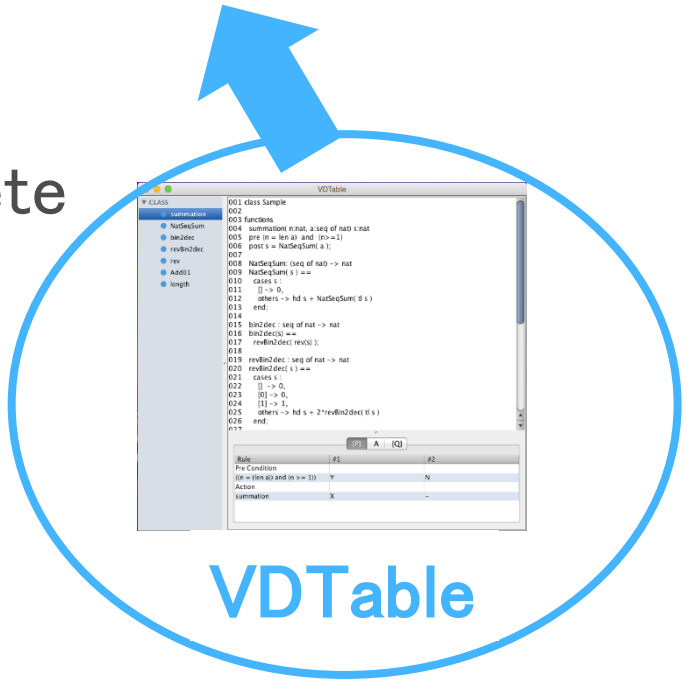
Specification A (4)

Specification B (16)

Specification C (256)

Measure and compare the time taken to complete the decision table for each specification





measurement result (sec)

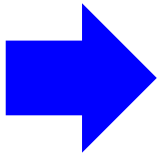
subject	VDM++ specification (The number of combination of truth values)		
	specification A (4)	specification B (16)	specification C (256)
A	252	405	1131
B	213	559	1292
C	169	499	1194
D	240	435	1859
E	134	557	1397
average	216	498	1629
VDTable	**0.012**	**0.016**	**0.02**

- VDTable reduced labor and time for manual work.
- VDTable is useful for improving the work of designing a decision table using the VDM++ specification.

Boundary Value + Vienna Development Method = BWDM

VDM++ Specification

Decision Table

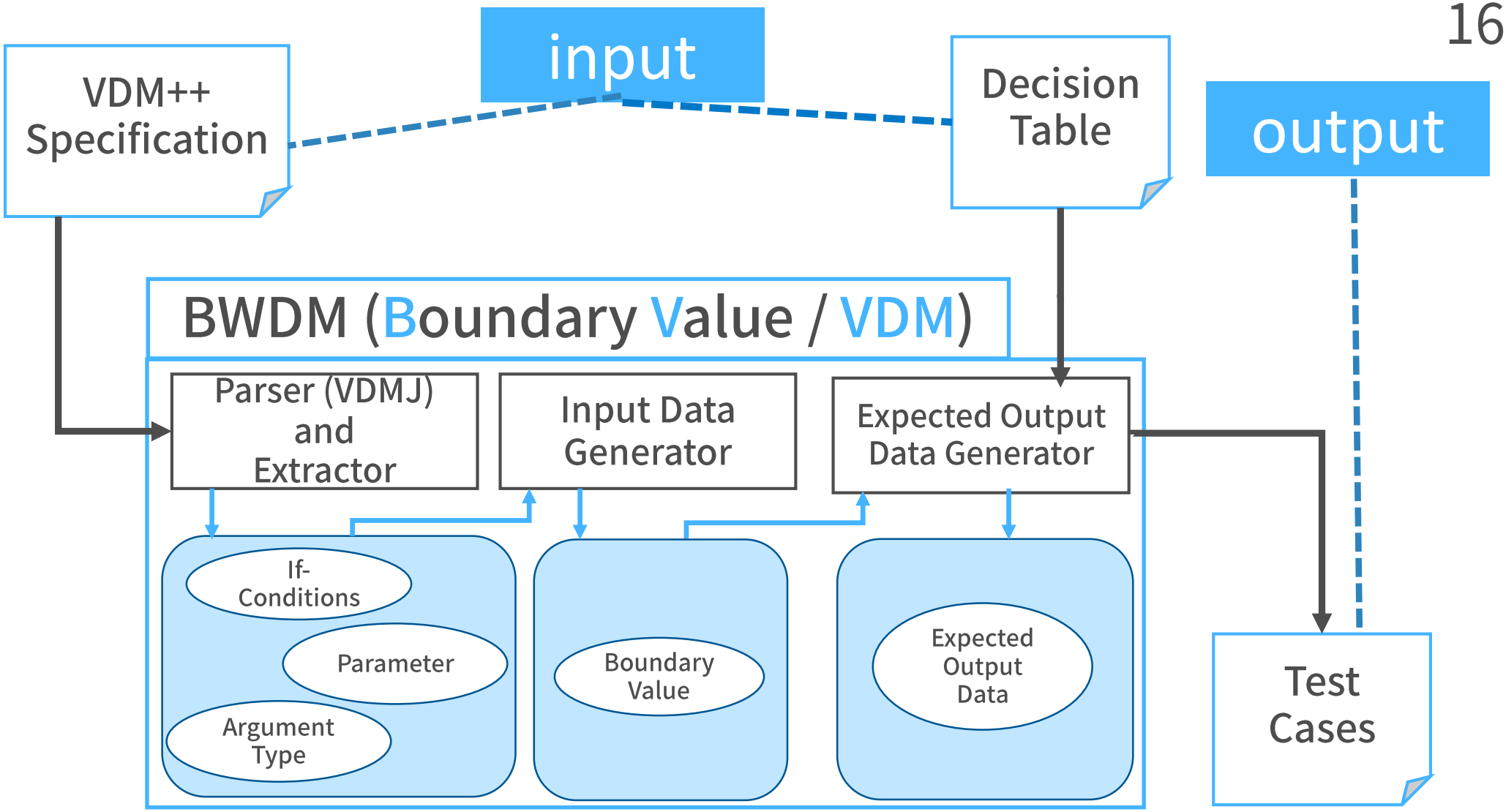


Test Cases

No.5	natMin-1	0	-->	Undefined Action
No.6	natMin-1	-1	-->	Undefined Action
No.7	natMin	intMin-1	-->	Undefined Action
No.8	natMin	intMin	-->	arg1:even arg2:negative
No.9	natMin	intMax	-->	arg1:even arg2:positive
No.10	natMin	intMax+1	-->	Undefined Action
No.11	natMin	0	-->	arg1:even arg2:positive
No.12				arg1:even arg2:negative
No.13				Undefined Action
No.14				arg1:odd arg2:negative
No.15				arg1:odd arg2:positive
No.16				Undefined Action
No.17				arg1:odd arg2:positive
No.18				arg1:odd arg2:negative
No.19				Undefined Action
No.20				Undefined Action
No.21				Undefined Action
No.22	natMax+1	intMax+1	-->	Undefined Action
No.23	natMax+1	0	-->	Undefined Action
No.24	natMax+1	-1	-->	Undefined Action
No.25	2	intMin-1	-->	Undefined Action
No.26	2	intMin	-->	arg1:even arg2:negative
No.27	2	intMax	-->	arg1:even arg2:positive
No.28	2	intMax+1	-->	Undefined Action
No.29	2	0	-->	arg1:even arg2:positive
No.30	2	-1	-->	arg1:even arg2:negative
No.31	1	intMin-1	-->	Undefined Action
No.32	1	intMin	-->	arg1:odd arg2:negative

BWDM outputs test cases that based on boundary value analysis.

overview of BWDM



test cases example

SampleSpecICAROB...

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
class Sample

functions

SampleFunction : int -> seq of char
    SampleFunction(a)==
       if(a < 5) then
         "a < 5"
       else if(12 <= a) then
         "12 <= a"
       else
         "5 <= a < 12";

end Sample
```

VDM++ specification

- a function definition in VDM++ specification
- an argument of int
- return value is Sequence of char
- two if-conditions

test cases example

SampleSpecICAROB...

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
class Sample

functions

SampleFunction : int -> seq of char
    SampleFunction(a)==
        if(a < 5) then
          "a < 5"
        else if(12 <= a) then
          "12 <= a"
        else
          "5 <= a < 12";

end Sample
```

Type Boundary Value

if-condition boundary value

VDM++ Specification

Boundary Value Analysis

Expected Output Data of outside input values of type range

Test Case No.	Input Data	Expected Output Data
No.1	intMin-1	“Undefined Action”
No.2	intMin	“a < 5”
No.3	intMax	“12 <= a”
No.4	intMax+1	“Undefined Action”
No.5	4	“a < 5”
No.6	5	“5 < a <= 12”
No.7	11	“5 < a <= 12”
No.8	12	“12 <= a”

test cases

Input Data

- Generated from function definition in VDM++ specification.
- Consists of 2 kind of boundary value; type boundary value that is generated from arguments type, and if-conditions boundary value that is generated from if conditions.

type boundary value

Ex.) When the argument of the function is an int

↓

intMin, intMax, intMin-1, intMax+1

Minimum, maximum, and outside values of Integer range

In testing phase, tester corrects it to actual numbers, such as intMin:-2147483648, according to the development environment (language, bit number, etc.), and performs test.

if-conditions boundary value

Ex.) When the if-condition is “a<4”

↓

3, 4 (3:true, 4:false)

two integer values at the boundary of the inequality (three values when if-condition is modulo)

Expected Output Data

- Output when input data is input to program
- When receiving outside values of type range (*Min-1, *Max+1), the program overflows and the operation can not be predicted so the expected output data is assumed to be “Undefined Action”.

test cases example

SampleSpecICAROB...

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
class Sample

functions

SampleFunction : int -> seq of char
    SampleFunction(a)==
        if(a < 5) then
          "a < 5"
        else if(12 <= a) then
          "12 <= a"
        else
          "5 <= a < 12";

end Sample
```

Type Boundary Value

if-condition boundary value

VDM++ Specification

Boundary Value Analysis

Expected Output Data of outside input values of type range

Test Case No.	Input Data	Expected Output Data
No.1	intMin-1	“Undefined Action”
No.2	intMin	“a < 5”
No.3	intMax	“12 <= a”
No.4	intMax+1	“Undefined Action”
No.5	4	“a < 5”
No.6	5	“5 < a <= 12”
No.7	11	“5 < a <= 12”
No.8	12	“12 <= a”

test cases

Mixed inequality and modulo

```
class MixSpecification

functions

  mix: nat * int -> seq of char
    mix( arg1 , arg2 ) ==
      if( arg1 mod 2 = 0) then
        if (0 <= arg2 ) then
          " arg1:even arg2:positive"
        else
          " arg1:even arg2:negative"
      else
        if( arg2 < 0) then
          " arg1:odd arg2:negative"
        else
          " arg1:odd arg2:positive";

end MixSpecification
```

- two arguments
- determine whether arg1 is an even number in the first remainder expression
- In two inequalities, it judges the sign of arg2

Application result


```
The number of arguments:2

argument type:  argument1:nat   argument2:int

No.     input data      -->     expected output data
No.1    natMin-1        intMin-1        -->     Undefined Action
No.2    natMin-1        intMin          -->     Undefined Action
No.3    natMin-1        intMax          -->     Undefined Action
No.4    natMin-1        intMax+1        -->     Undefined Action
No.5    natMin-1        0               -->     Undefined Action
No.6    natMin-1        -1              -->     Undefined Action
No.7    natMin          intMin-1        -->     Undefined Action
No.8    natMin          intMin          -->      arg1:even arg2:negative
No.9    natMin          intMax          -->      arg1:even arg2:positive
No.10   natMin          intMax+1        -->     Undefined Action
No.11   natMin          0               -->      arg1:even arg2:positive
No.12   natMin          -1              -->      arg1:even arg2:negative
No.13   natMax          intMin-1        -->     Undefined Action
No.14   natMax          intMin          -->      arg1:odd arg2:negative
No.15   natMax          intMax          -->      arg1:odd arg2:positive
No.16   natMax          intMax+1        -->     Undefined Action
No.17   natMax          0               -->      arg1:odd arg2:positive
No.18   natMax          -1              -->      arg1:odd arg2:negative
No.19   natMax+1        intMin-1        -->     Undefined Action
No.20   natMax+1        intMin          -->     Undefined Action
No.21   natMax+1        intMax          -->     Undefined Action
No.22   natMax+1        intMax+1        -->     Undefined Action
No.23   natMax+1        0               -->     Undefined Action
No.24   natMax+1        -1              -->     Undefined Action
No.25   2               intMin-1        -->     Undefined Action
No.26   2               intMin          -->      arg1:even arg2:negative
No.27   2               intMax          -->      arg1:even arg2:positive
No.28   2               intMax+1        -->     Undefined Action
No.29   2               0               -->      arg1:even arg2:positive
No.30   2               -1              -->      arg1:even arg2:negative
No.31   1               intMin-1        -->     Undefined Action
No.32   1               intMin          -->      arg1:odd arg2:negative
No.33   1               intMax          -->      arg1:odd arg2:positive
No.34   1               intMax+1        -->     Undefined Action
No.35   1               0               -->      arg1:odd arg2:positive
No.36   1               -1              -->      arg1:odd arg2:negative
No.37   3               intMin-1        -->     Undefined Action
No.38   3               intMin          -->      arg1:odd arg2:negative
No.39   3               intMax          -->      arg1:odd arg2:positive
No.40   3               intMax+1        -->     Undefined Action
No.41   3               0               -->      arg1:odd arg2:positive
No.42   3               -1              -->      arg1:odd arg2:negative
```

According to the specification,

- arg1:7, arg2:6 boundary values are output correctly
- Output the input data and the expected output data correctly as the boundary value test case

Evaluation of usefulness

We input three specifications with different number of if-conditional expressions to BWDM,
and measured the time until test cases are generated.

Times	specification1	specification2	specification3
1st time	325	316	6277
2nd time	283	389	5321
3rd time	500	371	6236
4th time	291	334	5070
5th time	269	371	5700
Average	334	356	5720

We confirmed that boundary value test cases are automatically generated within a few seconds is the number of conditions is 15

Combinatorial Testing

Overture* supports for Combinatorial Testing.

→ By conducting combinatorial test with test cases generated by BWDM, its possible to conduct more effective testing to find bugs.

*http://overturetool.org/

Conclusion of this research

- In this research, we developed two tools to improve testing phase of software development with VDM++.
  (1) VDTable : generates a decision table
  (2) BWDM : generates boundary values

- We think that our tools have the potential of improving software testing process with VDM++.

Common issues of both tools

- Evaluation by using huge specification that used in real development
- Correspondence to exponential explosion
- Expanding coverage of the tool
- Correspondence to other test design techniques

The issue of VDTable

- Correspondence to compound conditional expression

The issue of BWDM

- Implementation of Symbolic Execution

parsing data

VDTable : decision table


```
#access
private
#location
in '/Users/katlab/Documents/workspace/VDMDT/data/Sample.vdmpp' at line 4:3
#name
revBin2dec
#type
(seq of (nat) -> nat)
#kind
explicit function
#body
private 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)))))
```

#access

- types of access modifiers

#location

- location of VDM++ file
- start position of definition

#name

- name of definition

#type

- argument type

#kind

- types of definition

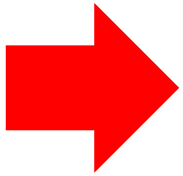
#body

- inside definition

internal representation data

VDTable : decision table


```
...
#kind
explicit function
#body
private 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)))))
```



```
if1
(s = [])
then
0
elseif
(s = [0])
then
0
elseif
(s = [1])
then
1
else1
((hd s) + (2 * revBin2dec((tl s))))
```

condition extraction pattern	action extraction pattern
if “condition” ’ then elseif “condition” then	then “action” ’ if then “action” ’ elseif then “action” ’ else else “action” ’ if else “action” ’ elseif else “action” ’ else else “action” ’ EOF
cases “condition” ’ –>	–> “action” ’ cases others “Action” ’ EOF
pre “condition” ’ post pre “condition” ’ EOF post “condition” ’ EOF	

※EOF(End Of File)

condition array

internal representation data

```
if1
(s = [])
then
0
elseif
(s = [0])
then
0
elseif
(s = [1])
then
1
else1
((hd s) + (2 * revBin2dec((tl s))))
```

condition array

index	condition
0	(s = [])
1	(s = [0])
2	(s = [1])

extraction rule ※EOF(End Of File)

condition extraction pattern	action extraction pattern
if “condition’’ then elseif “condition” then	then “action’’ if then “action’’ elseif then “action’’ else else “action’’ else else “action’’ elseif else “action’’ else else “action’’ EOF
cases “condition’’ ->	-> “action’’ cases others “action’’ EOF
pre “condition’’ post pre “condition’’ EOF post “condition’’ EOF	

internal representation data

```
if1
(s = [])
then
0
elseif
(s = [0])
then
0
elseif
(s = [1])
then
1
else1
((hd s) + (2 * revBin2dec((tl s))))
```

action array

index	action
0	0
1	1
2	((hd s) + (2 * revBin2dec((tl s))))

extraction rule ※EOF(End Of File)

condition extraction pattern	action extraction pattern
if “condition’’ then elseif “condition” then	then “action’’ if then “action’’ elseif then “action’’ else else “action’’ else else “action’’ elseif else “action’’ else else “action’’ EOF
cases “condition’’ ->	-> “action’’ cases others “action’’ EOF
pre “condition’’ post pre “condition’’ EOF post “condition’’ EOF	

condition-action table

VDTable : decision table

internal representation data

```
if1
(s = [])
then
0
elseif
(s = [0])
then
0
elseif
(s = [1])
then
1
else1
((hd s) + (2 * revBin2dec((tl s))))
```

index	condition
0	(s = [])
1	(s = [0])
2	(s = [1])

condition array

CA-Table

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2

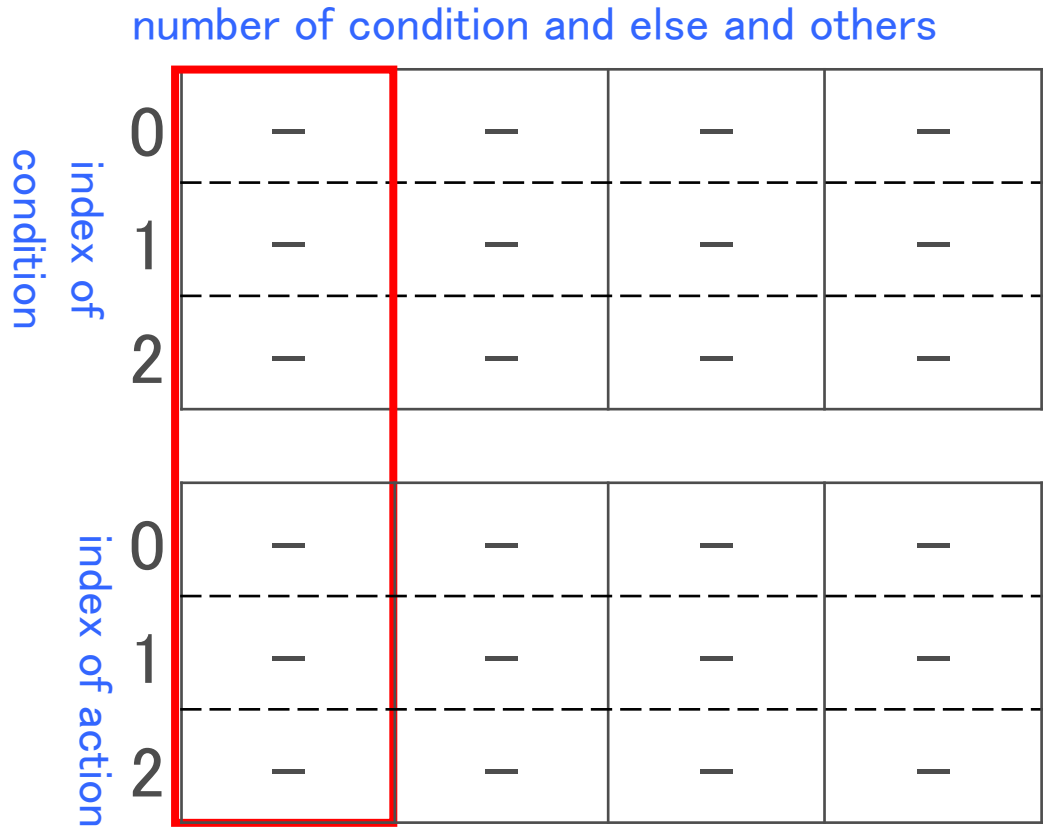
action array

index	action
0	0
1	1
2	((hd s) + (2 * revBin2dec((tl s))))

1. Create an array that is two dimensional and stores truth value.
2. Select the first column of the array.
3. Select a row of CA-Table from the first row.
4. Compare tokens and store truth value in the array
   A) When “if”, “elseif”, “cases” are matched,
      I. Store "Y" to the column selected for the conditional index row and "N" from the next column to the end of the column.
      II. Store “X” in action index of selected column.
   B) When “else”, “others” are matched,
      I. Store "N" to the column selected for the conditional index row and "-" from the next column to the end of the column.
      II. Store “X” in action index of selected column.
5. If there are unselected rows, select the next column of the array and return to 3. When there is no, truth value is completed.

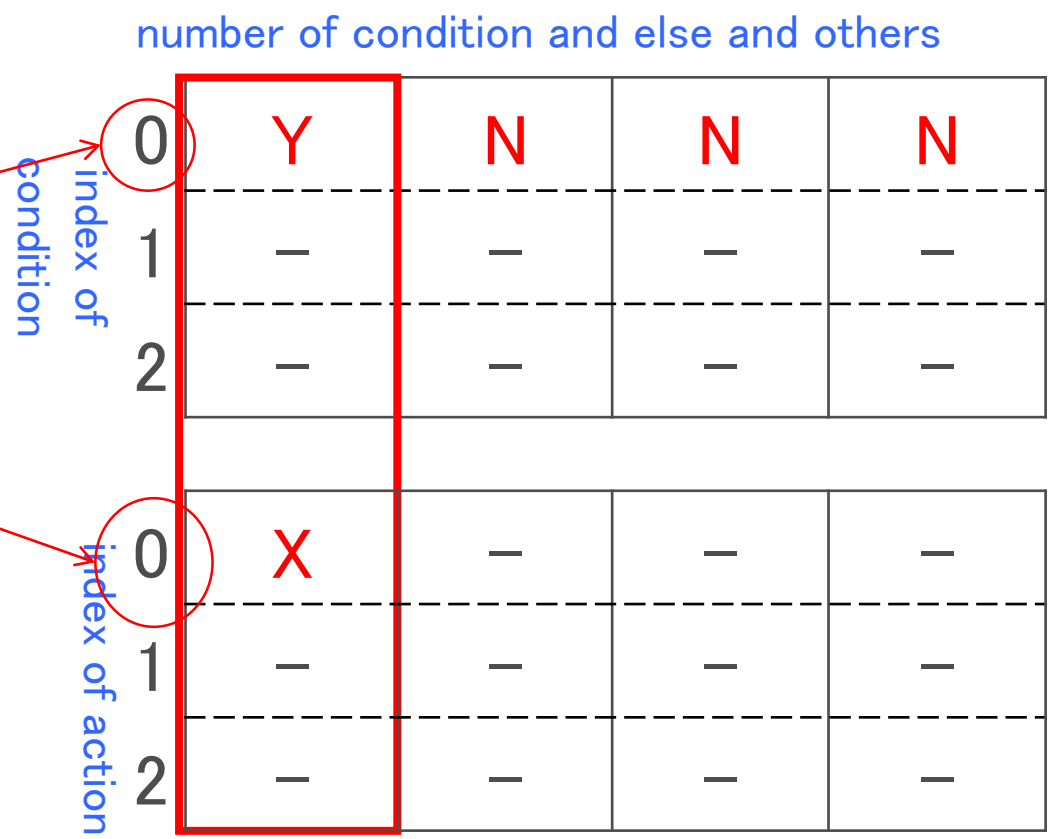
1. Create an array that is two dimensional and stores truth value.
2. Select the first column of the array.
3. Select one row of the CA-Table.

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2



4. Compare tokens and store truth value in the array
   A) When “if”, “elseif”, “cases” are matched,
      I. Store "Y" to the column selected for the conditional index row and "N" from the next column to the end of the column.
      II. Store “X” in action index of selected column.

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2



5. If there are unselected rows, select the next column of the array and return to 3.

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2

number of condition and else and others

index of condition				
0	Y	N	N	N
1	–	–	–	–
2	–	–	–	–

index of action				
0	X	–	–	–
1	–	–	–	–
2	–	–	–	–

4. Compare tokens and store truth value in the array
   A) When “if”, “elseif”, “cases” are matched,
      I. Store "Y" to the column selected for the conditional index row and "N" from the next column to the end of the column.
      II. Store “X” in action index of selected column.

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2

number of condition and else and others

index of condition

0	Y	N	N	N
1	–	Y	N	N
2	–	–	–	–

index of action

0	X	X	–	–
1	–	–	–	–
2	–	–	–	–

4. Compare tokens and store truth value in the array
   A) When “if”, “elseif”, “cases” are matched,
      I. Store "Y" to the column selected for the conditional index row and "N" from the next column to the end of the column.
      II. Store “X” in action index of selected column.

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2

number of condition and else and others

index of condition

0	Y	N	N	N
1	–	Y	N	N
2	–	–	Y	N

index of action

0	X	X	–	–
1	–	–	X	–
2	–	–	–	–

4. Compare tokens and store truth value in the array
   B) When “else”, “others” are matched,
      I. Store "N" to the column selected for the conditional index row and "-" from the next column to the end of the column.
      II. Store “X” in action index of selected column.

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2

number of condition and else and others

index of condition

0	Y	N	N	N
1	–	Y	N	N
2	–	–	Y	N

index of action

0	X	X	–	–
1	–	–	X	–
2	–	–	–	X

5. If there are unselected rows, select the next column of the array and return to 3.
   When there is no, truth value is completed.

index of condition	token	index of action
0	if1	0
1	elseif	0
2	elseif	1
0	else1	2

number of condition and else and others

index of condition

0	Y	N	N	N
1	–	Y	N	N
2	–	–	Y	N

index of action

0	X	X	–	–
1	–	–	X	–
2	–	–	–	X

completion of the decision table

index	condition
0	(s = [])
1	(s = [0])
2	(s = [1])

index	action
0	0
1	1
2	((hd s) + (2 * revBin2dec((tl s))))

0	Y	N	N	N
1	–	Y	N	N
2	–	–	Y	N

0	X	X	–	–
1	–	–	X	–
2	–	–	–	X

Rule	#1	#2	#3	#4
Condition				
(s = [])	Y	N	N	N
(s = [0])	–	Y	N	N
(s = [1])	–	–	Y	N
Action				
0	X	X	–	–
1	–	–	X	–
((hd s) + (2 * revBin2dec((tl s))))	–	–	–	X

CEGTest*

The tool for automatically generating a decision table.

The cause result graph as the input of CEGTest has to be created manually by the user from the VDM++ specification.

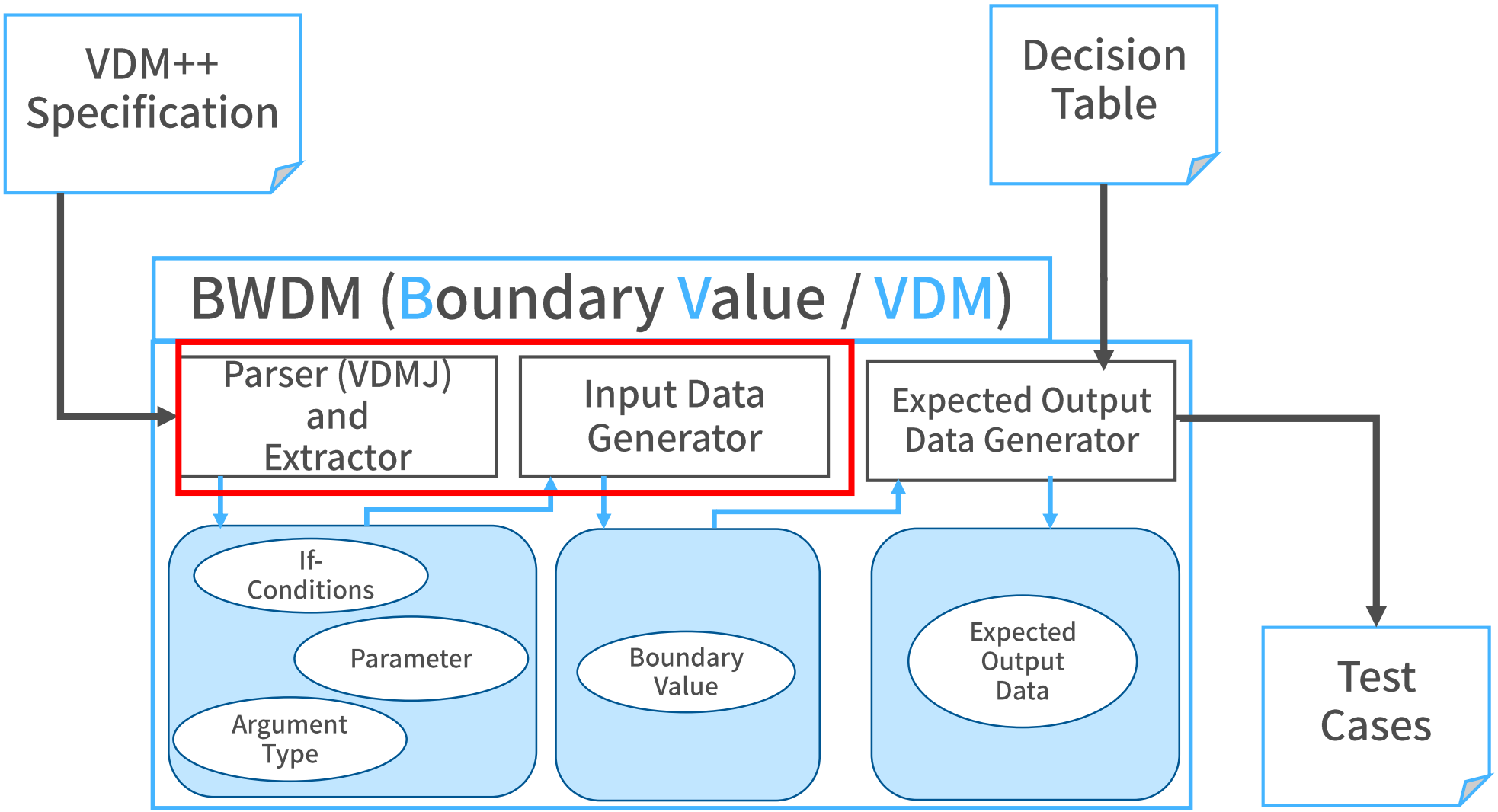
→ VDTable only needs a VDM++ specification for input.

*http://softest.jp/tools/CEGTest/

limitation of BWDM

- Function definition is only supported definition of VDM++ .
- The number of arguments is limited to one or two.
- The type of arguments is limited to Integer. (int, nat, nat1)
- If-conditions are limited to inequalities and modulo.
- If-conditions are limited to those not connected by “and”, “or”.
- If-conditions are limited to ones whose sides are not variables.

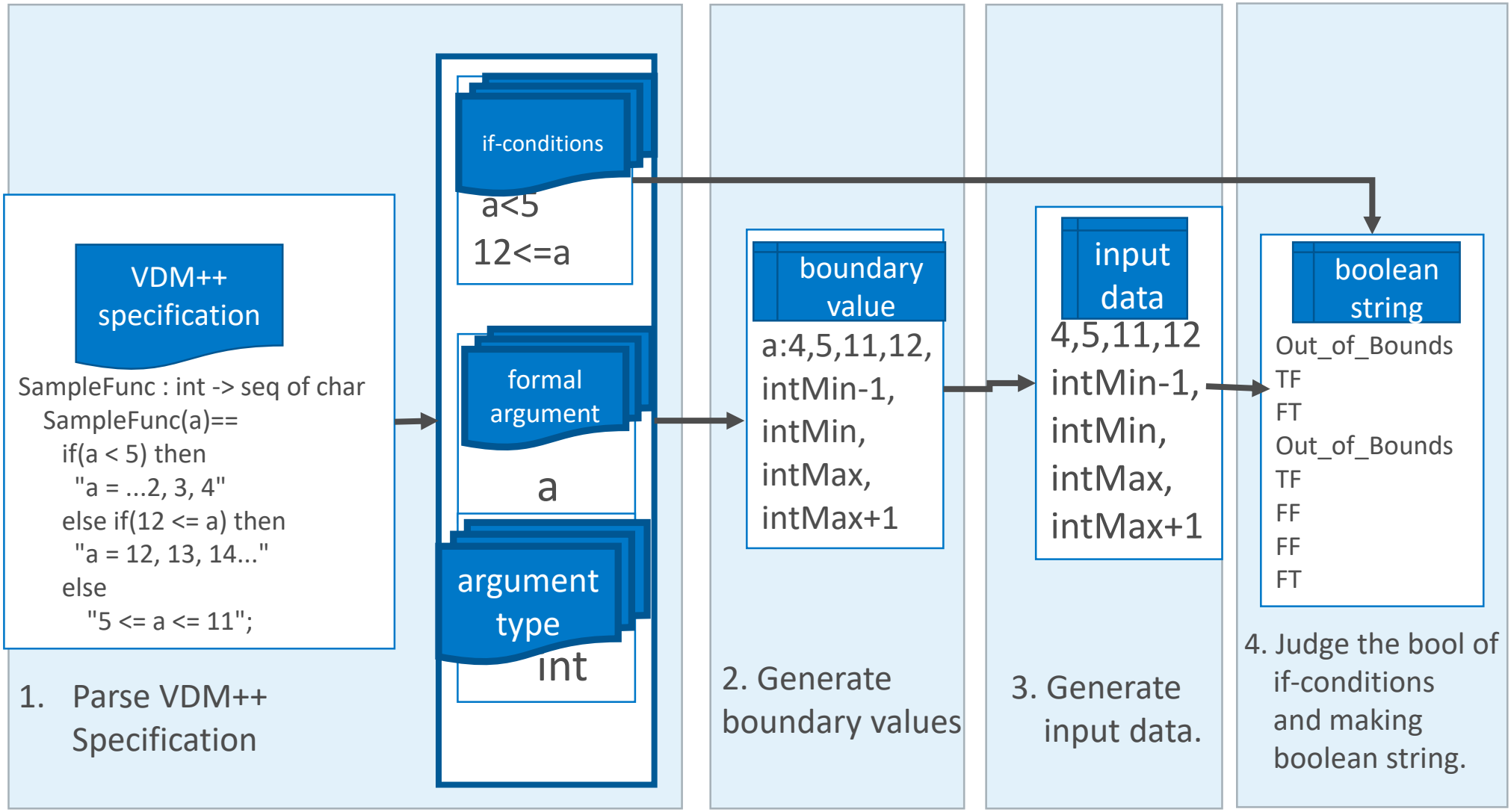
Explain process of making input data.



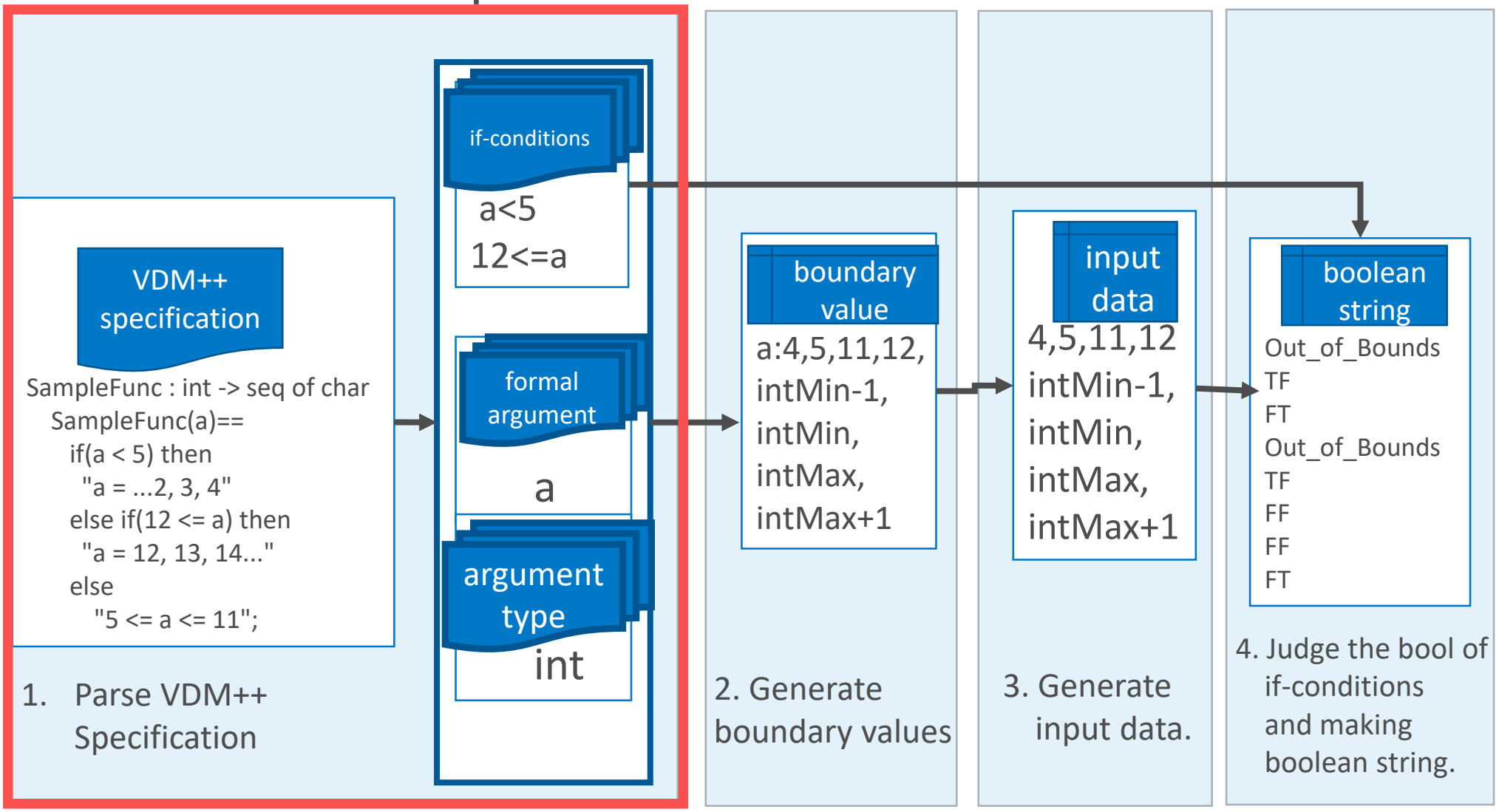
Following four processes are performed for making input data

1. Parse VDM++ specification
2. Generate boundary values
3. Generate input data
4. Generate boolean strings

Overview of the process flow

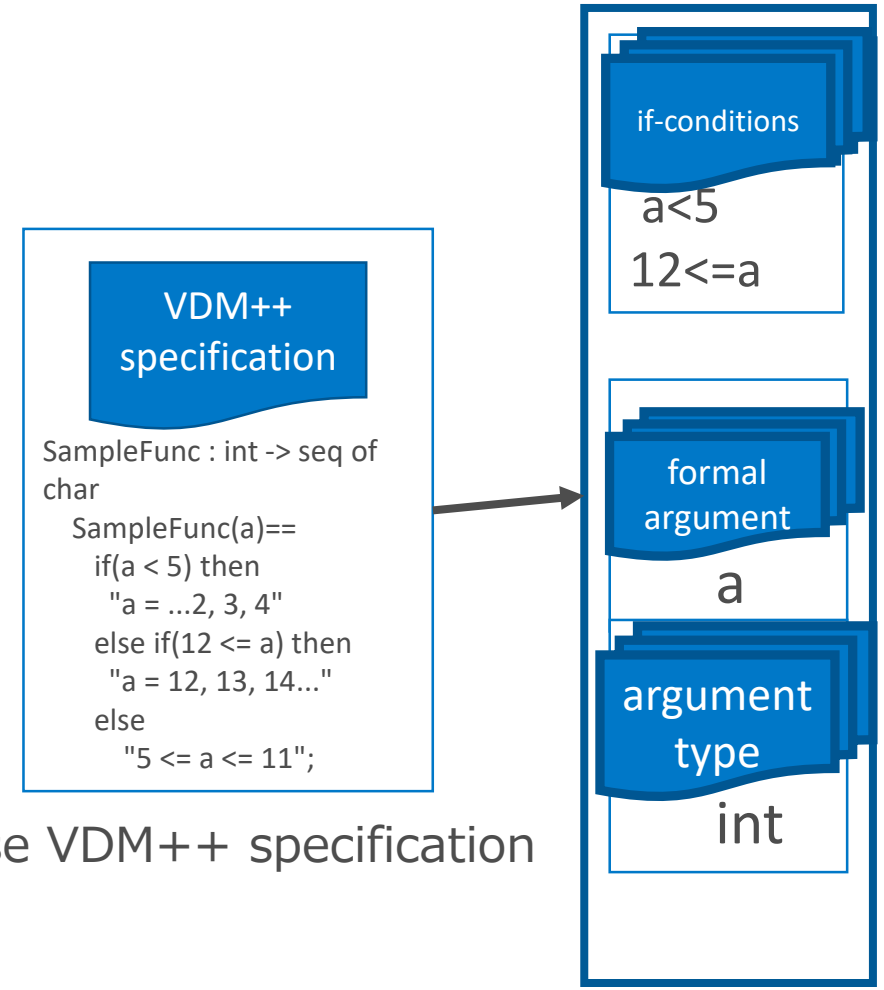


1. Parse VDM++ specification



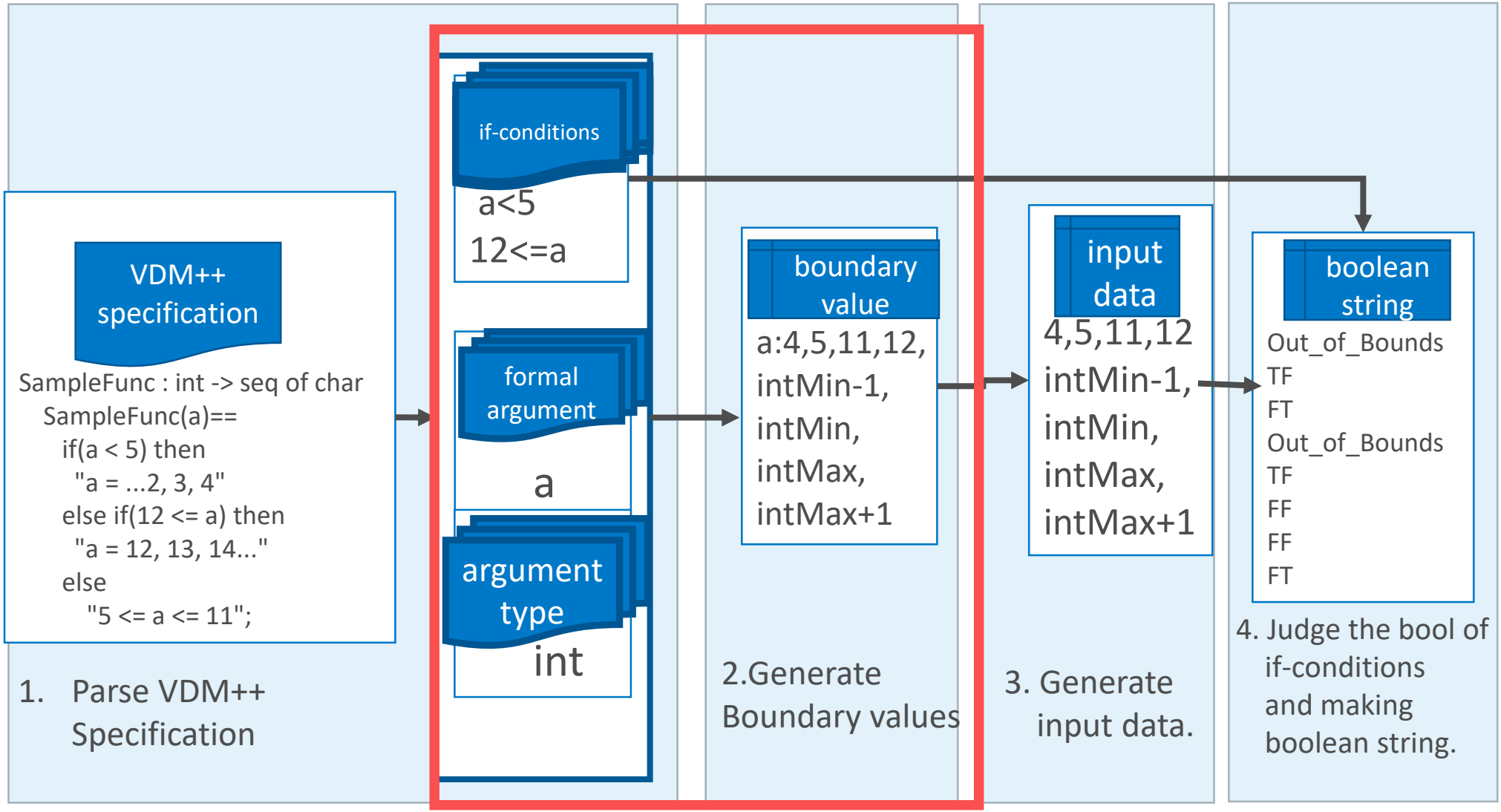
1.Parse VDM++ specification

Parse VDM++ specification
by using VDMJ
↓
Extract argument types,
formal arguments, and
if-conditions



1.Parse VDM++ specification

2.Generate boundary Values



2.Generate boundary Values

Based on the extracted information, generate boundary values.

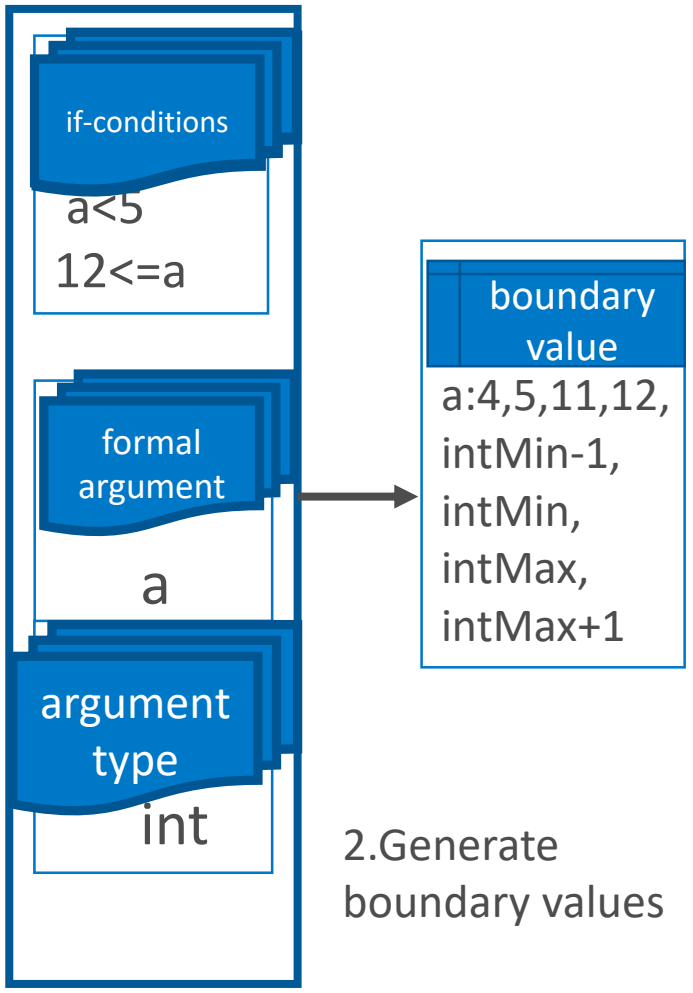
type boundary value

- Generate minimum value, maximum value, minimum value-1, maximum value+1 of argument type

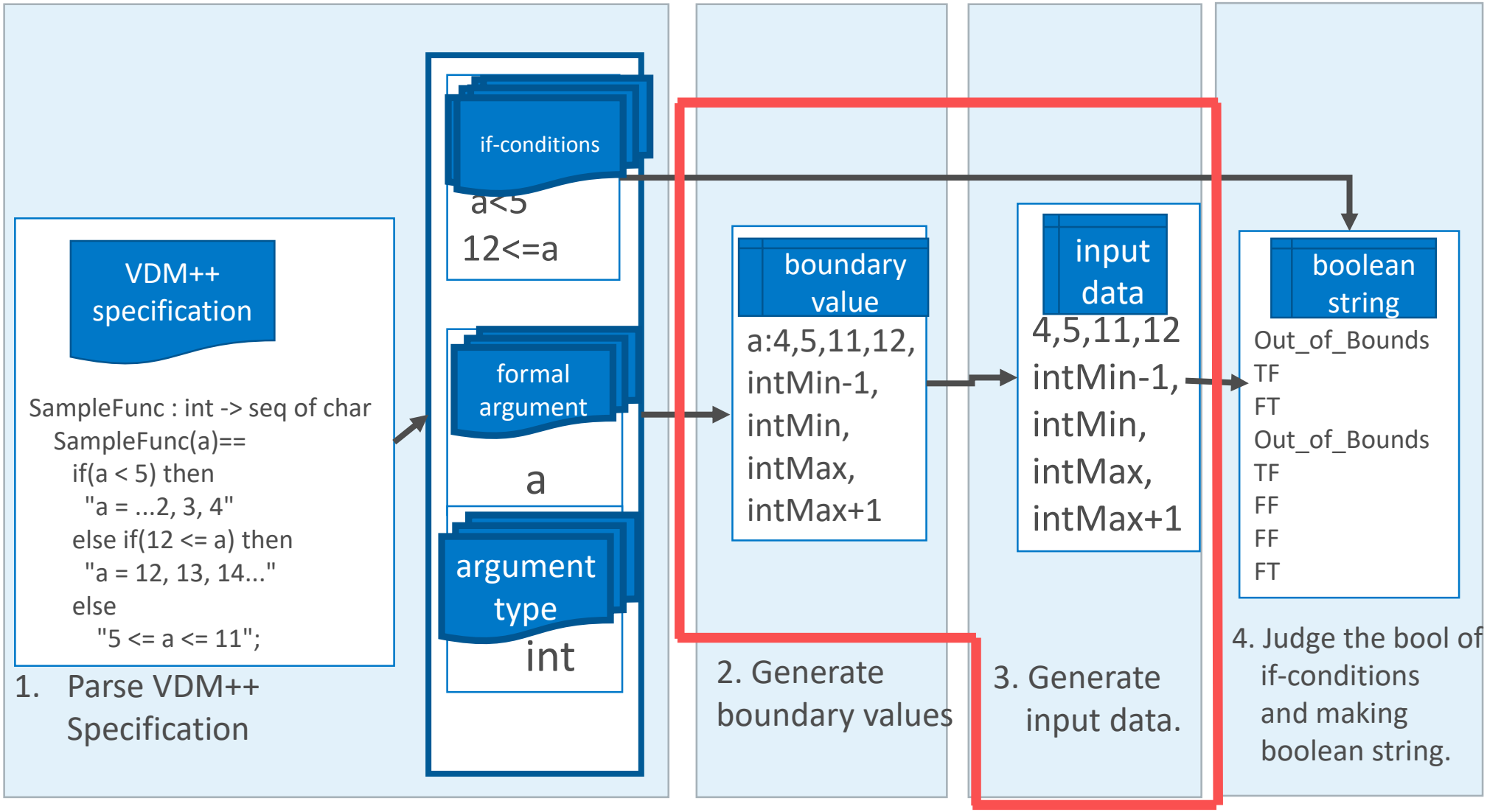
if-condition boundary value

- Generate two boundary values in case of inequality
- three values in case of modulo;

  a value that satisfies “=” and two values of ±1 of it



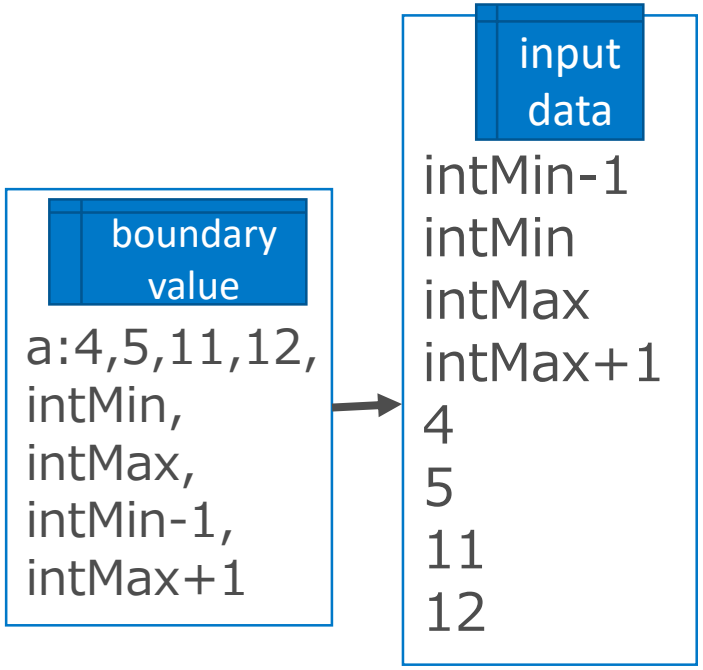
- 3.Generate Input Data



3.Generate Input Data

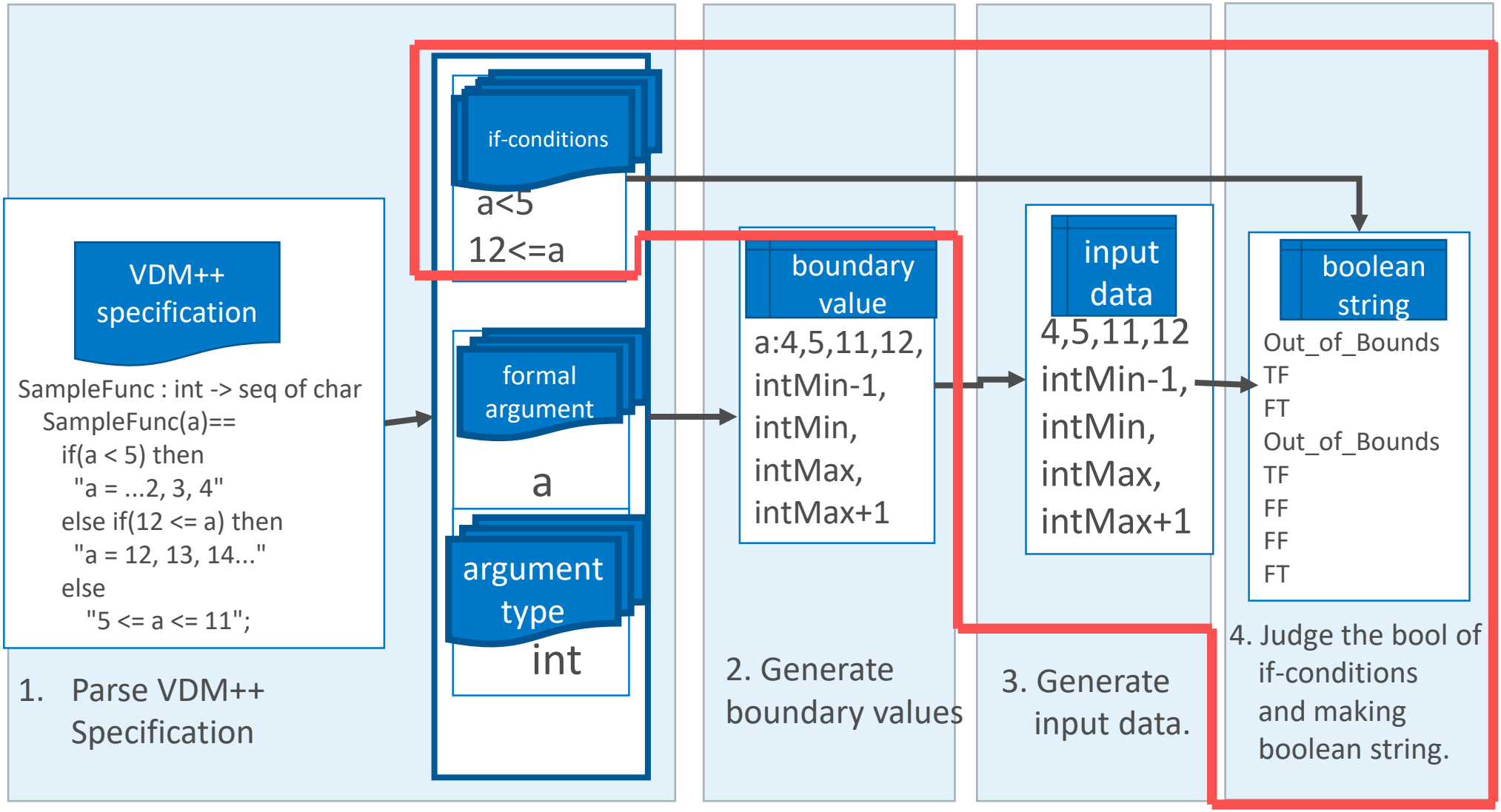
Generate input data from generated boundary value

In the case of two arguments, all combinations of boundary values of each arguments are generated for input data.



3.Generate input data

- 4.Genarate Boolean Strings



4.Genarate Boolean Strings

Generate Boolean strings from
the generated input data
and if-conditional expressions

↓

used in the expected output data generator

Boolean String
- Assigning input data to the if-conditional expressions to determine a Boolean value and characterizing the result of judgement as T of F

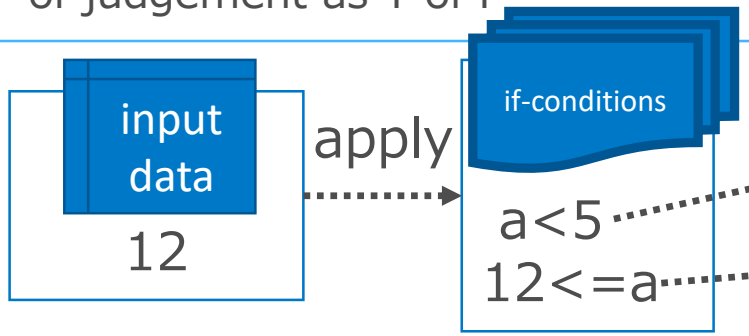
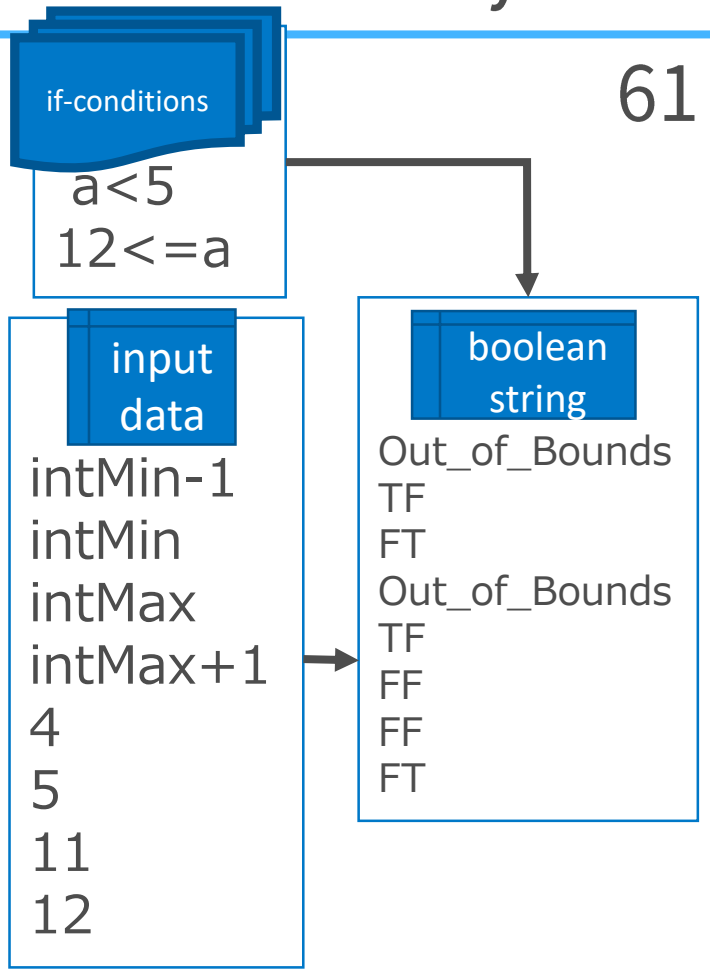
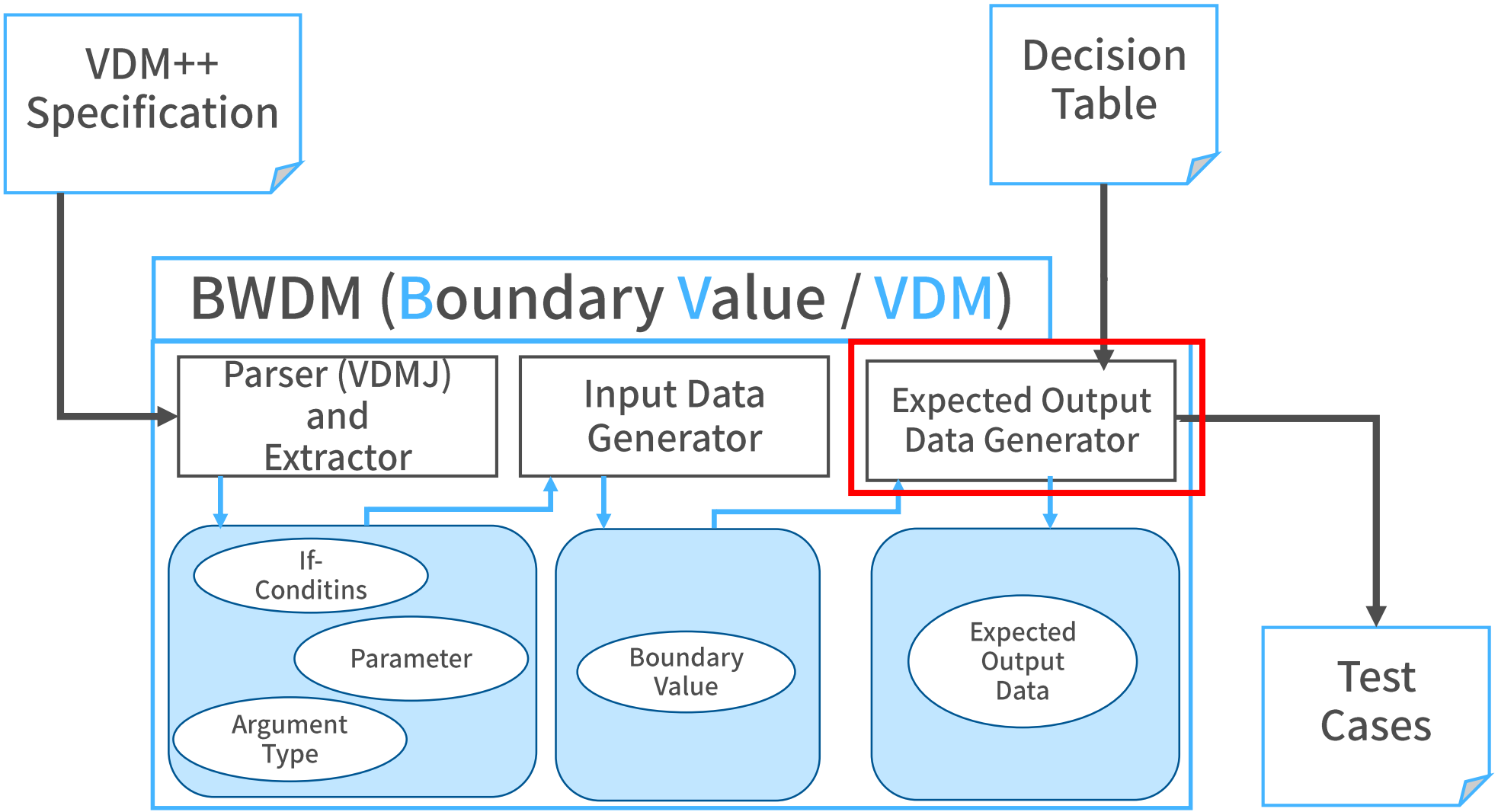


figure: Example of creating a Boolean string "FT" from input data "1, 2" and if-conditional expressions "a<5, 12<=a"



4. Judge the bool of if-conditions and making boolean string.

Explain process of making expected output data.



making expected output data

Following two steps are performed
to make expected output data.

1. Load decision table and generate
   corresponding table of Boolean string and action
2. Generate expected output data and test cases

making expected output data

- Overview of the Process Flow

Decision Table

		#1	#2	#3	#4
Condition	a < 5	T	T	F	F
Condition	12 <= a	T	F	T	F
Action	“a<5”	T	T	F	F
Action	“12 <= a”	F	F	T	F
Action	“5<=a<=11”	F	F	F	T

corresponding table of boolean strings and actions

boolean string	action
TT	“a<5”
TF	“a<5”
FT	“12<=a”
FF	“5<=a<=11”

1.Load decision table and generating table of Boolean string and action.

input data and Boolean string

input data
intMin-1
intMin
intMax
intMax+1
4
5
11
12

boolean string
Out_of_Bounds
TF
FT
Out_of_Bounds
TF
FF
FF
FT

test cases

input	expected output
intMin-1	-> “Undefined”
intMin	-> ”a<5”
intMax	-> “12<=a”
intMax+1	-> “Undefined”
4	-> “a<5”
5	-> “5<=a<=11”
11	-> “5<=a<=11”
12	-> “12<=a”

2.Generate expected output data from input data, boolean string, and corresponding table of boolean string and action. And generate test cases.

making expected output data

- 1. Load decision table and generate table of Boolean strings and actions

Decision Table

		#1	#2	#3	#4
Condition	a < 5	T	T	F	F
Condition	12 <= a	T	F	T	F
Action	“a<5”	T	T	F	F
Action	“12 <= a”	F	F	T	F
Action	“5<=a<=11”	F	F	F	T

corresponding table of boolean strings and actions

boolean string	action
TT	“a<5”
TF	“a<5”
FT	“12<=a”
FF	“5<=a<=11”

1.Load decision table and generating table of Boolean string and action.

input data and Boolean string

input data

intMin-1
intMin
intMax
intMax+1
4
5
11
12

boolean string

Out_of_Bounds
TF
FT
Out_of_Bounds
TF
FF
FF
FT

test cases

input	expected output
intMin-1	-> “Undefined”
intMin	-> ”a<5”
intMax	-> “12<=a”
intMax+1	-> “Undefined”
4	-> “a<5”
5	-> “5<=a<=11”
11	-> “5<=a<=11”
12	-> “12<=a”

2.Generate expected output data from input data, boolean string, and corresponding table of boolean string and action. And generate test cases.

making expected output data

- 1.Load decision table and generate table of Boolean strings and actions

Load the decision table that generated by VDTable.

↓

Generate correspondence table of Boolean strings and actions.

Decision Table

		#1	#2	#3	#4
Condition	a < 5	T	T	F	F
Condition	12 <= a	T	F	T	F
Action	“a<5”	T	T	F	F
Action	“12 <= a”	F	F	T	F
Action	“5<=a<=11”	F	F	F	T

corresponding table of boolean strings and actions

boolean string	action
TT	“a<5”
TF	“a<5”
FT	“12<=a”
FF	“5<=a<=11”

1.Load decision table and generating table of Boolean string and action.

2.Generate expected output data and test cases

Decision Table

		#1	#2	#3	#4
Condition	a < 5	T	T	F	F
Condition	12 <= a	T	F	T	F
Action	“a<5”	T	T	F	F
Action	“12 <= a”	F	F	T	F
Action	“5<=a<=11”	F	F	F	T

corresponding table of Boolean string and action

boolean string	action
TT	“a<5”
TF	“a<5”
FT	“12<=a”
FF	“5<=a<=11”

1.Load decision table and generating table of Boolean string and action.

input data and Boolean string

input data
intMin-1
intMin
intMax
intMax+1
4
5
11
12

boolean string
Out_of_Bounds
TF
FT
Out_of_Bounds
TF
FF
FF
FT

test cases

input	expected output
intMin-1	-> “Undefined”
intMin	-> ”a<5”
intMax	-> “12<=a”
intMax+1	-> “Undefined”
4	-> “a<5”
5	-> “5<=a<=11”
11	-> “5<=a<=11”
12	-> “12<=a”

2.Generate expected output data from input data, boolean string, and corresponding table of boolean string and action.
And generate test cases.

2.Generate Expected Output Data and Test Cases

Generate expected output data from input data, boolean strings, and corresponding table of boolean string and action.

↓

Finally, BWDM outputs test cases that consist input data and expected output data.