Using constraint solvers to support metamorphic testing

M.Carmen de Castro-Cabrera, Inmaculada Medina-Bulo, Antonio García-Dominguez

Department of Computer Science
Universidad de Cádiz,
SARI, EAS
Aston University

ICSE MET, May 2019

UCASE

Índex

UCASE

Index

Context

Context

- Metamorphic Testing(MT)
- Metamorphic Relations(MRs)
- Systematic MRs identification and validation

Target

- Represent MRs as a set of constraint
  - Constraint solvers
  - Select MiniZinc
  - Apply to a case study

Index

Fundamentals

Constraint Programs Systems (CPS)

- 90s: Constraint Logic Programming (CLP)
- Constraint Programs Systems CPS
- MiniZinc

Constraint solving and MT

- Prototype implementation using INKA test case generator (2003,Gotlieb and Botella)
- MT to validate the Minion CPS (2018, Akgün et al.)

Index

The MiniZinc language

Features

- Open source constraint language
- Uses: model constraint satisfaction and optimization
- Separate data and model files

The MiniZinc language (II)

```
% Colouring Andalusia using this many colours                    1
int: nc = 3;                                                     2
                                                                 3
var 1..nc: al; var 1..nc: ca; var 1..nc: co;                     4
var 1..nc: gr; var 1..nc: hu; var 1..nc: ja;                     5
var 1..nc: ma; var 1..nc: se;                                    6
                                                                 7
constraint al != gr; constraint gr != ja;                        8
constraint gr != co; constraint gr != ma;                        9
constraint ja != co; constraint co != ma;                        10
constraint co != se; constraint ma != ca;                        11
constraint ma != se; constraint se != ca;                        12
constraint se != hu;                                             13
                                                                 14
solve satisfy ;                                                  15
                                                                 16
output ["al=\(al)\t ca=\(ca)\n",                                 17
        "co=\(co)\t gr=\(gr)\t <hu=\(hu)\n",                     18
        "ja=\(ja)\t ma=\(ma)\n",                                 19
        "se ", show(se), "\n"];                                  20
```

Listing 1. Sample MiniZinc model

The MiniZinc language (III)

- Parameter definition
- Variable declarations
- Constraints
- *solve* (satisfaction or minimization/maximization)

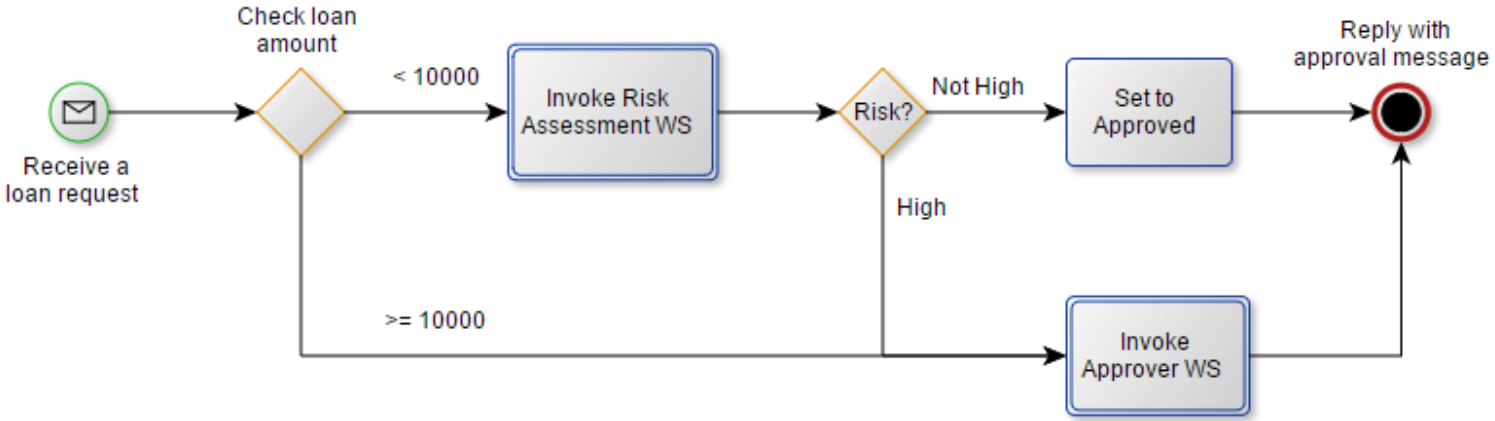
Index

Mapping MR to MiniZinc

- Decision variable and parameter declarations (initial and follow-up test cases)
- A set of constraints over previous decision variables and parameters

Index

Loan Approval Example



- Low amount ($< 10{,}000$)
  - Low Risk
  - High Risk
- High amount ($>= 10{,}000$)

Notation

We recall that each MR relates an initial test case (here it is represented by 1) to a subsequent test case (here it is represented by 2). The elements of each test case are:

- `req_amountN` is the amount requested by the customer, N can be 1 or 2 (depending on the initial case or the next case, generated by the MR)
- `ap_replyN` is the response from the approver service ("true" if approved, "false" if not)
- `as_replyN` is the assessor's response ("true" is high, "false" is low)
- `accepted` is the final answer given to the customer

Some MRs

- **MR1.2** (increasing low amount):

$$req\_amount2 = req\_amount1 * 10 \land req\_amount1 < 10000 \land req\_amount2 < 10000 \land ap\_reply2 = ap\_reply1 \land as\_reply2 = as\_reply1 \implies accepted2 = accepted1$$

- **MR2.1** (reducing high amount):

$$req\_amount2 = req\_amount1/2 \land req\_amount1 >= 10000 \land req\_amount2 >= 10000 \land ap\_reply2 = ap\_reply1 \land as\_reply2 = as\_reply1 \implies accepted2 = accepted1$$

- **MR3** (negating the approver on a high amount):

$$req\_amount2 = req\_amount1 \land req\_amount1 >= 10000 \land ap\_reply2 = not(ap\_reply1) \land as\_reply2 = as\_reply1 \implies accepted2 = not(accepted1)$$

- **MR4** (negating the assessor on a high amount):

$$req\_amount2 = req\_amount1 \land req\_amount1 >= 10000 \land ap\_reply2 = ap\_reply1 \land as\_reply2 = not(as\_reply1) \implies accepted2 = accepted1$$

New test cases generation

Once the satisfiability of the proposed MRs has been checked, this section tries to use these MRs to generate new test cases (the follow-up test cases). This is the traditional use of MRs in the MT technique

- For high amounts, `(amount = 15000, ap_reply = true, as_reply = false, accepted = true)` was used. While the common model file requires a value for `as_reply`, this value did not impact executions: the assessor is not used for large amounts
- For low amounts, there are two options: low risk or high risk. For low risk, `(amount = 150, ap_reply = false, as_reply = false, accepted = true)` could be used with MR1.2 and MR2.2. For high risk, `(amount = 15000, ap_reply = false, as_reply = true, accepted = false)` was used with MR1.3

```
1  % PARAMETERS (OLD TEST CASE)
2  int: req_amount1;
3  bool: ap_reply1;
4  bool: as_reply1;
5  bool: accepted1;
6
7  % VARIABLES (NEW TEST CASE)
8  var int: req_amount2;
9  var bool: ap_reply2;
10 var bool: as_reply2;
11 var bool: accepted2;
12
13 %% COMPOSITION-SPECIFIC CONSTRAINTS
14 constraint req_amount1 >= 0;
15 constraint req_amount2 >= 0;
```

Listing 1: Common MiniZinc model for LoanApproval

```
include "loanAp_common.mzn";

%% MR-SPECIFIC CONSTRAINTS
constraint req amount1 < 10000 ;
constraint req amount2 < 10000 ;
constraint req_amount2 = req_amount1 * 10;
constraint ap_reply2 = ap_reply1;
constraint as_reply2 = as_reply1;
constraint accepted2 = accepted1;
```

Listing 2: MiniZinc model of MR1.2 (increasing low amount) of LoanApproval

```
include "loanAp_common.mzn";

%% MR-SPECIFIC CONSTRAINTS
constraint req_amount1 >= 10000 ;
constraint req_amount2 = req_amount1;
constraint ap_reply2 = ap_reply1;
constraint as_reply2 = not as_reply1;
constraint accepted2 = accepted1;
```

Listing 3: MiniZinc model of MR4 (negating the assessor on a high amount) of LoanApproval

```
% VARIABLES (OLD TEST CASE)                                    1
var int: req_amount1;                                          2
var bool: ap_reply1;                                           3
var bool: as_reply1;                                           4
var bool: accepted1;                                           5
                                                               6
% VARIABLES (NEW TEST CASE)                                    7
var int: req_amount2;                                          8
var bool: ap_reply2;                                           9
var bool: as_reply2;                                           10
var bool: accepted2;                                           11
                                                               12
%% COMPOSITION-SPECIFIC CONSTRAINTS                            13
constraint req_amount1 >= 0;                                   14
constraint req_amount2 >= 0;                                   15
```

Listing 4: Common MiniZinc model of LoanApproval with variables

Generating new cases

Test cases generation tables

TABLE II
GENERATED TESTS, WITH SOURCE MRS (HIGH AMOUNT)

MR	Initial test case	Following test case
MR1.1	(15000, true, true, true)	(150000, true,true, true)
MR2.1	(15000, true, true, true)	(7500, true,true, true)
MR3	(15000, true, true, true)	(15000, false,true, false)
MR4	(15000, true, true, true)	(15000, true, false, true)

TABLE III
GENERATED TESTS, WITH SOURCE MRS (LOW AMOUNT)

MR	Initial test case	Following test case
MR1.2	(150, false, false, true)	(1500, false,false, true)
MR2.2	(150, false, false, true)	(75, false, false, true)
MR1.3	(1500, false, true, false)	(15000, false, false, false)

Index

MuBpel

Steps

1. The composition was analyzed
2. All possible mutants were generated
3. The original composition, initial test case, collecting its outputs
4. The mutants were executed
5. The original composition, following test cases, collecting its outputs
6. The remaining mutants were executed. The following test cases, and their outputs were compared against those of the original composition

Results

TABLE IV
MUBPEL VALIDATION RESULTS

Suite	Total	Killed	Alive	# tests
Initial	96	50	46	3
Follow-up	96	84	12	3 + 7

Index

Conclusions

Conclusions

- Challenges in MT
- Proposal of representing MRs through MiniZinc
- A case of study has been presented
- It has been checked through mutation testing
- The following tests cases were be able to detect 34 defect

Future work

- Further improve defect detection
- Larger and more complex programs

Thanks

maricarmen.decastro@uca.es