2017 IEEE International Conference on Software Quality, Reliability and Security (Companion Volume)

Modelling and verification of user interactions using constraint programming

Mats Carlsson
SICS
Stockholm, Sweden
Mats.Carlsson@sics.se

Olga Grinchtein
Ericsson AB
Stockholm, Sweden
olga.grinchtein@ericsson.com

Justin Pearson
Department of Information Technology
Uppsala University, Sweden
justin.pearson@it.uu.se

***Abstract*—Graphical user interfaces are important components of today's software. User interfaces often require checking correctness of user interactions. In web applications such checks can be a part of the JavaScript code. User interfaces in web applications can evolve, some elements can be removed and new elements can be added. To check JavaScript code covers all possible incorrect scenarios in user interactions in web application, constraint programming is used. We use the MiniZinc constraint modelling language to model incorrect user behaviour and to convert JavaScript code into a constraint model. Then we perform an equivalence check to find deviations in JavaScript code. The approach was applied to design user interface of an industrial software product.**

I. Introduction

We introduce a counterexample-driven approach to support code design that alerts user in case of incorrect interaction with user interface. We start with JavaScript code that contains **if else** statements that cover different incorrect cases of user behaviour and generates alert messages. We convert **if else** statements into constraints by using logical negation. The constraints are expressed in the MiniZinc language [1]. On the other hand, we create a model of incorrect user behaviour as a constraint in MiniZinc. The advantage of MiniZinc is that it allows us to describe incorrect behaviour in a concise form. Then we use MiniZinc to check equivalence between our model and JavaScript code. If MiniZinc finds a counterexample, we modify model or JavaScript code. We can question if good user interface needs such approach. However, this technique was applied during integration of a module into existing software product, Ericsson system that generates reports, and the approach helped to design user interface for the module.

II. Constraint Programming

Constraint Programming [2] (CP) is a framework for modelling and solving combinatorial problems including verification and optimisation tasks. A constraint problem is specified as a set of *decision variables* that have to be assigned values so that the given constraints on these variables are satisfied, and optionally so that a given objective function is minimised or maximised. Constraint solving is based on the constructive search for such an assignment. Constraint propagation plays an important role: a constraint is not only a declarative modelling device, but has an associated propagator, which is an algorithm to prune the search space by removing values that cannot participate in a solution to that constraint. The removal can trigger other propagators, and this process continues until a fixpoint is reached, at which time the next assignment choice must be made.

MiniZinc [1] is a constraint modelling language, which has gained popularity recently due to its high expressivity and large number of available solvers that support it. It also contains many useful modelling abstractions such as quantifiers, sets, arrays, and a rich set of global constraints. MiniZinc is compiled into FlatZinc, a constraint solving language which specifies a set of built-in constraints that a constraint solver must support. The compilation process is based on flattening by introducing auxiliary variables, substituting them for nested subexpressions, and selecting the appropriate FlatZinc constraints. Common sub-expression elimination plays an important role as well.

III. Related work

Combinational equivalence checking is used in circuit design and checks whether specification and implementation translated into Boolean functions are the same. Then equivalence check can be performed by comparing implementation and specification transformed into BDDs or by using SAT solvers [3], [4].

[5] introduces adaptive model checking, where model checking is performed on some preliminary model. If a counterexample is found, it is compared with the actual system. This results in either the conclusion that the system does not satisfy its property, or an automatic refinement of the model. The approach uses learning algorithm to update the model and employs a testing algorithm to compare the model with the actual system.

In [6] linear equality and inequality constraints are used in specifying layout and other geometric relations in user interfaces. One of applications is processing potentially invalid user inputs, such as move a figure outside of its bounding window.

Constraint programming is used in configuration management that ensures that components of software system interact appropriately. Configuration information can be represented as a series of constraints expressed in first order predicate logic [7]. In [8] a constraint-based object-oriented configuration language is introduced.

A number of researches have worked on applying formal methods to design and verification of user interfaces [9], [10], [11], [12].

978-1-5386-2072-4/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS-C.2017.92

IV. Overview of the Approach

In our application we had to integrate new features into an existing user interface. The user interface had defined rules for user behaviour. Our extension required different rules. We decided to use message alerts and follow the product design to use features of the product that users are used to. The integration of a new module required adding several input control elements to user interface such as checkboxes, dropdown lists and textfields. Some of these input control elements could not be selected together and some of these input control elements should be selected together. We wanted to alert user if selected combination is incorrect.

We will now illustrate our methodology on a simple example. Assume that we have five input control elements such that first three must be selected together, 4th and 5th are optional, but if 4th or 5th is selected than first three should be also selected. Then the incorrect behaviour is selecting at least one of five input control elements, but not selecting first three input control elements. We introduce a boolean variable C_i for each input control element, $1 \leq i \leq 5$. C_i equals to 1 means that ith input control element is selected. Then incorrect user behaviour can be defined as the $Constraint_{spec}$

$$C_1 + C_2 + C_3 + C_4 + C_5 \geq 1 \wedge C_1 + C_2 + C_3 \leq 2 \tag{1}$$

The following code is used to alert the user.

Listing 1. Code example

```
if (C1 AND (!C2 OR !C3))
alert()
else if (C2 AND (!C1 OR !C3))
alert()
else if (C3 AND (!C1 OR !C2))
alert()
else if (C4 AND !C1)
alert()
else if (C5 AND !C1)
alert()
```

In the code $Constraint_{spec}$ is split into several conditions to simplify alerts for incorrect cases. The next step is to convert code into $Constraint_{impl}$ by replacing **else if** with logical negation

$$\begin{array}{l}
(C_1 = 1 \wedge (C_2 = 0 \vee C_3 = 0)) \\
\bigvee \\
(\neg(C_1 = 1 \wedge (C_2 = 0 \vee C_3 = 0))\wedge \\
(C_2 = 1 \wedge (C_1 = 0 \vee C_3 = 0)) \\
\bigvee \\
(\neg(C_1 = 1 \wedge (C_2 = 0 \vee C_3 = 0))\wedge \\
\neg(C_2 = 1 \wedge (C_1 = 0 \vee C_3 = 0))\wedge \\
(C_3 = 1 \wedge (C_1 = 0 \vee C_2 = 0))) \\
\bigvee \\
(\neg(C_1 = 1 \wedge (C_2 = 0 \vee C_3 = 0))\wedge \\
\neg(C_2 = 1 \wedge (C_1 = 0 \vee C_3 = 0))\wedge \\
\neg(C_3 = 1 \wedge (C_1 = 0 \vee C_2 = 0))\wedge \\
(C_4 = 1 \wedge C_1 = 0)) \\
\bigvee \\
(\neg(C_1 = 1 \wedge (C_2 = 0 \vee C_3 = 0))\wedge \\
\neg(C_2 = 1 \wedge (C_1 = 0 \vee C_3 = 0))\wedge \\
\neg(C_3 = 1 \wedge (C_1 = 0 \vee C_2 = 0))\wedge \\
\neg(C_4 = 1 \wedge C_1 = 0)\wedge \\
(C_5 = 1 \wedge C_1 = 0))
\end{array}$$

Then we can use MiniZinc to check if the constraint

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

is unsatisfiable. If it is unsatisfiable then it means that the program code is equivalent to the property we defined. In the case the constraint model is satisfiable we get a counterexample that is solution found by MiniZinc and we need to modify the constraint model.

If JavaScript code would be different

Listing 2. Code example

```
if (C1 AND (C2 OR !C3))
alert()
else if (C2 AND (!C1 OR !C3))
alert()
else if (C3 AND (!C1 OR !C2))
alert()
else if (C4 AND !C1)
alert()
else if (C5 AND !C1)
alert()
```

then the constraint

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

is satisfiable and we have a counterexample that is solution found by MiniZinc

$$C_1 = 1 \wedge C_2 = 1 \wedge C_3 = 1 \wedge C_4 = 0 \wedge C_5 = 0$$

The solution satisfies the constraint $Constraint_{impl}$ and does not satisfy the constraint $Constraint_{spec}$. The solution represents correct user behaviour and should be removed from $Constraint_{impl}$. Other cases of counterexample processing are possible, for example a solution that can represent incorrect behaviour that satisfies $Constraint_{spec}$ and should be added to $Constraint_{impl}$.

User interface can evolve and new input control element C_6 can be added or input control element C_4 can be removed.

In the next sections we exemplify these cases by applying the approach on real example.

V. Constraint Model

Suppose that the constraint

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl}) \tag{2}$$

is satisfiable with an assignment $solution$ to the decision variables.

We check if the constraint

$$solution \wedge Constraint_{spec} \tag{3}$$

is satisfiable. If yes, then we can analyze a counterexample, which could mean that some combinations are removed from $Constraint_{spec}$ or added to $Constraint_{impl}$. If constraint (3) is unsatisfiable, then the constraint

$$solution \wedge Constraint_{impl} \tag{4}$$

should be satisfiable. Then some combinations are removed from $Constraint_{impl}$ or added to $Constraint_{spec}$. In next sections we show some examples how we process counterexamples.

TABLE I. VARIABLES IN THE CONSTRAINT MODEL

$timeper.dl.ac$	dropdown list
$timeper.dl.up$	dropdown list
$repA.dl.x$	dropdown list
$repA.dl.y$	dropdown list
$repA.dl.nc$	dropdown list
$repA.dl.sc$	dropdown list
$param.tf.ac$	textfield
$param.tf.nu$	textfield
$param.chk.nuf$	checkbox
$param.dl.sr$	dropdown list
$param.dl.su$	dropdown list
$param.tf.tn$	textfield
$param.chk.ch$	checkbox
$repB.dl.stt$	dropdown list
$repB.tf.stn$	textfield
$repB.dl.stx$	dropdown list
$repB.dl.sty$	dropdown list
$repB.dl.cht$	dropdown list

VI. INITIAL SETUP OF EXAMPLE

In this and following sections we show steps of applying the approach on our user interface. The first step that is described in this section is to define specification model and to transform JavaScript code into a constraint model. We started by creating specification of user interactions in already integrated user interface, which in the beginning consisted of 11 input controls elements such as dropdown lists, checkboxes and text fields. We emphasis that we started to define the specification as constraint model after JavaScript code with message alerts was written. We also used the approach during evolution of user interface and JavaScript code.

In Section VII we show how we processed first counterexample and weaken some condition in $Constraint_{impl}$. In Section VIII we describe three consecutive steps of adding new conditions to $Constraint_{impl}$. We modified $Constraint_{impl}$ after processing three counterexamples. During analysis of counterexamples we found that some case was missing in specification. We added this case and modified $Constraint_{spec}$ and $Constraint_{impl}$ in Section IX. Two next steps are described in Section X and Section XI where constraint model required modification due to adding new input control elements and removing one input control element. In Section XII we show that it can be several ways to add condition to the model and selecting condition can have impact on how fast equivalence check converges.

The input control elements are used to customize reports that the tool generates. The input control elements $param.tf.ac$, $param.tf.nu$, $param.chk.nuf$, $timeper.dl.up$, $param.dl.su$, $param.tf.nu$, $param.dl.sr$ are used to customize the basic report. The input control elements $repA.dl.x$, $repA.dl.y$, $repA.dl.nc$, $repA.dl.sc$ are only used to customize more advanced report A. Parameters of basic report are also used in customizing advanced report.

Several incorrect scenarios are possible. At least one of input control elements $param.tf.ac$, $param.tf.nu$, $param.chk.nuf$ should be selected. If $param.tf.nu$ or $param.chk.nuf$ are not selected, then elements $timeper.dl.up$ and $param.dl.su$ should not be selected. The element $repA.dl.sc$ is optional. If one of elements $repA.dl.x$, $repA.dl.y$, $repA.dl.nc$, $repA.dl.sc$ is selected then elements $repA.dl.x$, $repA.dl.y$, $repA.dl.nc$ should all be selected. The elements $param.dl.sr$ and $repA.dl.sc$ should not be selected together.

We formalize possible incorrect user behaviour as $Constraint_{spec}$

$$\begin{aligned}
&(param.tf.ac = 0 \land param.tf.nu = 0 \land \\
&\quad param.chk.nuf = 0) \\
&\lor \\
&((timeper.dl.up = 1 \lor param.dl.su = 1) \land \\
&\quad (param.tf.nu = 0 \land param.chk.nuf = 0)) \\
&\lor \\
&(repA.dl.x + repA.dl.y + repA.dl.nc + repA.dl.sc \geq 1 \land \\
&\quad repA.dl.x + repA.dl.y + repA.dl.nc \leq 2) \\
&\lor \\
&(param.dl.sr = 1 \land repA.dl.sc = 1)
\end{aligned}$$

In initial setup JavaScript code contained 7 conditions

Listing 3. Code in initial setup

```
if (!timeper.dl.ac AND !param.tf.ac AND
    !param.dl.sr AND !param.tf.nu AND
    !param.chk.nuf AND !timeper.dl.up AND
    !param.dl.su)
alert()
else if (repA.dl.x AND !repA.dl.y)
alert()
else if (repA.dl.y AND !repA.dl.x)
alert()
else if (repA.dl.x AND !repA.dl.nc)
alert()
else if (repA.dl.sc AND param.dl.sr)
alert()
else if (repA.dl.nc AND !repA.dl.x)
alert()
else if (repA.dl.nc AND !repA.dl.y)
alert()
```

To simplify presentation we introduce a label to each condition in the code shown in Table II.

TABLE II. SPECIFIED CONDITIONS IN IF STATEMENTS. STEP 1.

ifcond1	$timeper.dl.ac = 0 \land param.tf.ac = 0 \land$ $param.dl.sr = 0 \land param.tf.nu = 0 \land$ $param.chk.nuf = 0 \land timeper.dl.up = 0 \land$ $param.dl.su = 0$
ifcond2	$repA.dl.x = 1 \land repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \land repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \land repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \land param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \land repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \land repA.dl.y = 0$

Based on Table II we create $Constraint_{impl}$

$$\begin{aligned}
&\text{ifcond1} \bigvee \\
&(\neg\text{ifcond1} \land \text{ifcond2}) \bigvee \\
&(\neg\text{ifcond1} \land \neg\text{ifcond2} \land \text{ifcond3}) \bigvee \\
&(\neg\text{ifcond1} \land \neg\text{ifcond2} \land \neg\text{ifcond3} \land \\
&\quad \text{ifcond4}) \bigvee \\
&(\neg\text{ifcond1} \land \neg\text{ifcond2} \land \neg\text{ifcond3} \\
&\quad \land\neg\text{ifcond4} \land \text{ifcond5}) \bigvee \\
&(\neg\text{ifcond1} \land \neg\text{ifcond2} \land \neg\text{ifcond3} \land \\
&\quad \neg\text{ifcond4} \land \neg\text{ifcond5} \land \text{ifcond6}) \bigvee \\
&(\neg\text{ifcond1} \land \neg\text{ifcond2} \land \neg\text{ifcond3} \land \\
&\quad \neg\text{ifcond4} \land \neg\text{ifcond5} \land \neg\text{ifcond6} \land \text{ifcond7})
\end{aligned}$$

The ordering of conditions in Table II is important. The different ordering would generate different $Constraint_{impl}$.

VII. Weaken condition in $Constraint_{impl}$

The constraint model defined in previous section

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

has a solution and the constraint

$$solution \wedge Constraint_{spec}$$

is satisfiable. The variable $timeper.dl.ac$ is equal to 1 in the solution and all other variables are equal to 0. The variable $timeper.dl.ac$ only appears in ifcond1 and it does not appear in $Constraint_{spec}$. We can also see that $Constraint_{spec}$ contains the disjunct

$$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0 \tag{5}$$

that is weak form of ifcond1. While ifcond1 defines values of 7 variables, (5) defines values of only three of them. We do not need to create alert for each combination of these 7 variables. It is enough to use (5). We change ifcond1 to (5), update the table and generate $Constraint_{impl}$ from Table III.

TABLE III. Step 2.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$

VIII. Adding conditions to $Constraint_{impl}$

The new constraint model

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

has a solution and the constraint

$$solution \wedge Constraint_{spec}$$

is satisfiable. The variable $param.tf.ac$ and $timeper.dl.up$ are equal to 1 in the solution and all other variables are equal to 0. The variables $param.tf.ac$ and $timeper.dl.up$ do not appear together in any disjunct of $Constraint_{spec}$. However, $Constraint_{spec}$ contains combination

$$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$$

and this combination does not exist in Table III. We need to add new condition to the table III and decide where in the table to insert it. If the new condition contains only variables of the same report type we add the condition to the group of conditions of this report type. If the condition contains variable of report A and variable from other report type we add it to the group of conditions containing report A. In this step we add condition ifcond8 after ifcond1 and before conditions containing variables of report A, update the table, and generate $Constraint_{impl}$ from Table IV.

The new constraint model

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

TABLE IV. Step 3.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$

has a solution and the constraint

$$solution \wedge Constraint_{spec}$$

is satisfiable. The variables $param.tf.ac$ and $param.dl.su$ are equal to 1 in the solution and all other variables are equal to 0. The parameters $param.tf.ac$ and $param.dl.su$ do not appear together in any disjunct of $Constraint_{spec}$. However, $Constraint_{spec}$ contains disjunct

$$param.dl.su = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$$

and this combination does not exist in Table IV. We add condition ifcond9 after ifcond8 and before conditions containing variables of report A, update the table, and generate $Constraint_{impl}$ from Table V.

TABLE V. Step 4.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

has a solution and the constraint

$$solution \wedge Constraint_{spec}$$

is satisfiable. The variable $param.tf.ac$ and $repA.dl.sc$ are equal to 1 in the solution and all other variables are equal to 0. The variables $param.tf.ac$ and $repA.dl.sc$ do not appear together in any disjunct of $Constraint_{spec}$, but the variable $repA.dl.sc$ appears in two disjuncts of $Constraint_{spec}$. We do not need to consider the disjunct

$$param.dl.sr = 1 \wedge repA.dl.sc = 1, \tag{6}$$

since $param.dl.sr$ is equal to 0 in the solution. Thus, we only need to analyze the disjunct

$$repA.dl.x + repA.dl.y + repA.dl.nc + repA.dl.sc \geq 1 \wedge repA.dl.x + repA.dl.y + repA.dl.nc \leq 2 \tag{7}$$

First conjunct in (7) holds, since $reportA.dl.sc$ is equal to 1. In order to make hold second conjunct we assign 0 to $reportA.dl.x$. We add condition ifcond10

$$repA.dl.x = 0 \wedge repA.dl.sc = 1 \tag{8}$$

after condition ifcond7, update the table, and generate $Constraint_{impl}$ from Table VI.

TABLE VI. STEP 5.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$
ifcond10	$repA.dl.x = 0 \wedge repA.dl.sc = 1$

The new constraint model

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

is unsatisfiable.

IX. MISSING CASE IN THE CONSTRAINT MODEL

Analysis of counterexamples requires extracting combination that should be added or removed from constraint model. Since we do this analysis manually, it can be that by exploring different combinations we find invalid combination that is missing in $Constraint_{spec}$. We found one such case

$$param.tf.ac = 0 \wedge param.dl.sr = 1 \tag{9}$$

We added this combination to $Constraint_{spec}$

$$\begin{aligned}
&(param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge\\
&\quad param.chk.nuf = 0)\\
&\vee\\
&((timeper.dl.up = 1 \vee param.dl.su = 1) \wedge\\
&\quad (param.tf.nu = 0 \wedge param.chk.nuf = 0))\\
&\vee\\
&(repA.dl.x + repA.dl.y + repA.dl.nc + repA.dl.sc \geq 1 \wedge\\
&\quad repA.dl.x + repA.dl.y + repA.dl.nc \leq 2)\\
&\vee\\
&(param.dl.sr = 1 \wedge repA.dl.sc = 1)\\
&\vee\\
&(param.tf.ac = 0 \wedge param.dl.sr = 1)
\end{aligned}$$

and added the combination as ifcond11 after ifcond9 and before conditions containing variables of report A, update the table, and generate $Constraint_{impl}$ from Table VII.

TABLE VII. STEP 6.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond11	$param.tf.ac = 0 \wedge param.dl.sr = 1$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$
ifcond10	$repA.dl.x = 0 \wedge repA.dl.sc = 1$

The new constraint model

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

is unsatisfiable.

X. ADDING NEW ELEMENTS TO USER INTERFACE

We added two new input control elements to user interface and two new variables $param.tf.tn$ and $param.chk.ch$ was added to constraint model. We have a new requirement that input control elements $param.tf.tn$ and $param.chk.ch$ can be only selected if $param.tf.ac$ is selected. We added disjunct

$$\begin{aligned}
&(param.tf.tn = 1 \vee param.chk.ch = 1) \wedge\\
&\quad parametertfac = 0
\end{aligned}$$

to $Constraint_{spec}$

$$\begin{aligned}
&(param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge\\
&\quad param.chk.nuf = 0)\\
&\vee\\
&((timeper.dl.up = 1 \vee param.dl.su = 1) \wedge\\
&\quad (param.tf.nu = 0 \wedge param.chk.nuf = 0))\\
&\vee\\
&(repA.dl.x + repA.dl.y + repA.dl.nc + repA.dl.sc \geq 1 \wedge\\
&\quad repA.dl.x + repA.dl.y + repA.dl.nc \leq 2)\\
&\vee\\
&(param.dl.sr = 1 \wedge repA.dl.sc = 1)\\
&\vee\\
&(param.tf.ac = 0 \wedge param.dl.sr = 1)\\
&\vee\\
&((param.tf.tn = 1 \vee param.chk.ch = 1) \wedge\\
&\quad param.tf.ac = 0)
\end{aligned}$$

We added to the table conditions ifcond12 and ifcond13 after ifcond1 shown in Table VIII and generate $Constraint_{impl}$ from Table VIII.

TABLE VIII. STEP 7.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond12	$param.tf.tn = 1 \wedge param.tf.ac = 0$
ifcond13	$param.chk.ch = 1 \wedge param.tf.ac = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond11	$param.tf.ac = 0 \wedge param.dl.sr = 1$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$
ifcond10	$repA.dl.x = 0 \wedge repA.dl.sc = 1$

The new constraint model

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

is unsatisfiable.

In the next step we add new report B to the tool and we add five new input control elements to user interface. The new variables in the constraint model are $repB.dl.stt$, $repB.tf.stn$, $repB.dl.stx$, $repB.dl.sty$ and $repB.dl.cht$.

We have a new requirement and we modify $Constraint_{spec}$ by adding disjunct

$$\begin{aligned}
&(repB.dl.stt + repB.tf.stn + repB.dl.stx +\\
&\quad repB.dl.sty + repB.dl.cht \geq 1 \wedge\\
&\quad repB.dl.stt + repB.dl.stx + repB.dl.sty \leq 2)
\end{aligned}$$

We added to the table conditions ifcond14, ifcond15, ifcond16, ifcond17 and ifcond18 after ifcond10 shown in Table IX and generate $Constraint_{impl}$ from Table IX.

TABLE IX. STEP 8.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond12	$param.tf.tn = 1 \wedge param.tf.ac = 0$
ifcond13	$param.chk.ch = 1 \wedge param.tf.ac = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond11	$param.tf.ac = 0 \wedge param.dl.sr = 1$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$
ifcond10	$repA.dl.x = 0 \wedge repA.dl.sc = 1$
ifcond14	$repB.dl.stt = 1 \wedge repB.dl.stx = 0$
ifcond15	$repB.dl.stt = 1 \wedge repB.dl.sty = 0$
ifcond16	$repB.dl.stx = 1 \wedge repB.dl.stt = 0$
ifcond17	$repB.tf.stn = 1 \wedge repB.dl.stt = 0$
ifcond18	$repB.dl.cht = 1 \wedge repB.dl.stt = 0$

The new constraint model

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

is satisfiable. We process counterexample and add ifcond19 after ifcond16, shown in Table X and generate $Constraint_{impl}$ from Table X.

TABLE X. STEP 9.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond12	$param.tf.tn = 1 \wedge param.tf.ac = 0$
ifcond13	$param.chk.ch = 1 \wedge param.tf.ac = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0 \wedge param.chk.nuf = 0$
ifcond11	$param.tf.ac = 0 \wedge param.dl.sr = 1$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$
ifcond10	$repA.dl.x = 0 \wedge repA.dl.sc = 1$
ifcond14	$repB.dl.stt = 1 \wedge repB.dl.stx = 0$
ifcond15	$repB.dl.stt = 1 \wedge repB.dl.sty = 0$
ifcond16	$repB.dl.stx = 1 \wedge repB.dl.stt = 0$
ifcond19	$repB.dl.sty = 1 \wedge repB.dl.stt = 0$
ifcond17	$repB.tf.stn = 1 \wedge repB.dl.stt = 0$
ifcond18	$repB.dl.cht = 1 \wedge repB.dl.stt = 0$

The new constraint model

$$\neg(Constraint_{impl} \leftrightarrow Constraint_{spec})$$

is unsatisfiable.

XI. REMOVING ELEMENTS FROM USER INTERFACE

At this point we decided to remove input control element representing by variable $param.chk.nuf$. We removed $param.chk.nuf$ from $Constraint_{impl}$ and $Constraint_{spec}$. The new $Constraint_{spec}$ is

$$
\begin{aligned}
&(param.tf.ac = 0 \wedge param.tf.nu = 0)\\
&\vee\\
&((timeper.dl.up = 1 \vee param.dl.su = 1) \wedge param.tf.nu = 0)\\
&\vee\\
&(repA.dl.x + repA.dl.y + repA.dl.nc + repA.dl.sc \geq 1 \wedge\\
&\quad repA.dl.x + repA.dl.y + repA.dl.nc \leq 2)\\
&\vee\\
&(param.dl.sr = 1 \wedge repA.dl.sc = 1)\\
&\vee\\
&(param.tf.ac = 0 \wedge param.dl.sr = 1)\\
&\vee\\
&((param.tf.tn = 1 \vee param.chk.ch = 1) \wedge\\
&\quad param.tf.ac = 0)\\
&\vee\\
&(repB.dl.stt + repB.tf.stn + repB.dl.stx +\\
&\quad repB.dl.sty + repB.dl.cht \geq 1 \wedge\\
&\quad repB.dl.stt + repB.dl.stx + repB.dl.sty \leq 2)
\end{aligned}
$$

The new $Constraint_{impl}$ is generated from Table XI

TABLE XI. STEP 10.

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0$
ifcond12	$param.tf.tn = 1 \wedge param.tf.ac = 0$
ifcond13	$param.chk.ch = 1 \wedge param.tf.ac = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0$
ifcond11	$param.tf.ac = 0 \wedge param.dl.sr = 1$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$
ifcond10	$repA.dl.x = 0 \wedge repA.dl.sc = 1$
ifcond14	$repB.dl.stt = 1 \wedge repB.dl.stx = 0$
ifcond15	$repB.dl.stt = 1 \wedge repB.dl.sty = 0$
ifcond16	$repB.dl.stx = 1 \wedge repB.dl.stt = 0$
ifcond19	$repB.dl.sty = 1 \wedge repB.dl.stt = 0$
ifcond17	$repB.tf.stn = 1 \wedge repB.dl.stt = 0$
ifcond18	$repB.dl.cht = 1 \wedge repB.dl.stt = 0$

XII. IMPACT OF SELECTING CONDITION

Input control elements for report A and report B cannot be selected together. We modify $Constraint_{spec}$ by adding disjunct

$$
\begin{aligned}
&(repB.dl.stt + repB.tf.stn + repB.dl.stx +\\
&\quad repB.dl.sty + repB.dl.cht \geq 1 \wedge\\
&\quad repA.dl.x + repA.dl.y + repA.dl.nc + repA.dl.sc \geq 1)
\end{aligned}
\tag{10}
$$

The constraint (10) can be simplified as

$$
\begin{aligned}
&(repB.dl.stt + repB.dl.stx +\\
&\quad repB.dl.sty \geq 1 \wedge\\
&\quad repA.dl.x + repA.dl.y + repA.dl.nc \geq 1)
\end{aligned}
\tag{11}
$$

since $repB.dl.cht$, $repB.dl.cht$, $repA.dl.sc$ are optional in reports. If we proceed with constraint (10) we can choose to add ifcond20 to the table. We add ifcond20 after conditions containing variables from report A and before all conditions containing variables from report B. In this case we get

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

satisfiable and we need to process the counterexample.

TABLE XII. STEP 11

ifcond1	$param.tf.ac = 0 \wedge param.tf.nu = 0$
ifcond12	$param.tf.tn = 1 \wedge param.tf.ac = 0$
ifcond13	$param.chk.ch = 1 \wedge param.tf.ac = 0$
ifcond8	$timeper.dl.up = 1 \wedge param.tf.nu = 0$
ifcond9	$param.dl.su = 1 \wedge param.tf.nu = 0$
ifcond11	$param.tf.ac = 0 \wedge param.dl.sr = 1$
ifcond2	$repA.dl.x = 1 \wedge repA.dl.y = 0$
ifcond3	$repA.dl.y = 1 \wedge repA.dl.x = 0$
ifcond4	$repA.dl.x = 1 \wedge repA.dl.nc = 0$
ifcond5	$repA.dl.sc = 1 \wedge param.dl.sr = 1$
ifcond6	$repA.dl.nc = 1 \wedge repA.dl.x = 0$
ifcond7	$repA.dl.nc = 1 \wedge repA.dl.y = 0$
ifcond10	$repA.dl.x = 0 \wedge repA.dl.sc = 1$
ifcond20	$repA.dl.nc = 1 \wedge repB.dl.cht = 1$
ifcond14	$repB.dl.stt = 1 \wedge repB.dl.stx = 0$
ifcond15	$repB.dl.stt = 1 \wedge repB.dl.sty = 0$
ifcond16	$repB.dl.stx = 1 \wedge repB.dl.stt = 0$
ifcond19	$repB.dl.sty = 1 \wedge repB.dl.stt = 0$
ifcond17	$repB.tf.stn = 1 \wedge repB.dl.stt = 0$
ifcond18	$repB.dl.cht = 1 \wedge repB.dl.stt = 0$

However, if we proceed with constraint (11), and replace ifcond20 by

$$repA.dl.nc = 1 \wedge repB.dl.stt = 1 \tag{12}$$

in Table XII, we get

$$\neg(Constraint_{spec} \leftrightarrow Constraint_{impl})$$

is unsatisfiable. In fact we can replace $repA.dl.nc$ by $repA.dl.x$ or $repA.dl.y$ in (12) and replace $repB.dl.stt$ by $repB.dl.stx$ or $repB.dl.sty$ in (12) and the constraint model is still unsatisfiable. This example shows that it can be several ways to add new disjunct to $Constraint_{spec}$ and to add new conditions to JavaScript code. However, the selection of conditions for $Constraint_{impl}$ can have impact on how fast the equivalence check converges.

XIII. CONCLUSION

We presented an approach, where part of user interface code is designed and verified by performing equivalence check and analyzing counterexamples generated by $MiniZinc$. The approach can also be extended to non-boolean variables. For example, droplist contains several values and variable representing droplist can have range depending on number of values. In the future we plan to partially automate generation of $Constraint_{spec}$ and $Constraint_{impl}$ by processing counterexamples. Automation requires solving several problems

- We need to automate processing of a counterexample and finding a combination to be removed or added to a constraint model.
- New condition to $Constraint_{impl}$ should be added by using some ordering rules.
- A constraint model should be modified automatically if user interface evolves.
- Simplification of $Constraint_{impl}$ by merging several cases should be automated.

However, manual analysis of counterexamples is also important, since it can discover missing cases in $Constraint_{spec}$.

ACKNOWLEDGMENT

The second author was supported by Swedish Foundation for Strategic Research.

REFERENCES

[1] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "Minizinc: Towards a standard CP modelling language," in *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, ser. CP'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 529–543.

[2] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[3] S. Disch and C. Scholl, "Combinational equivalence checking using incremental SAT solving, output ordering, and resets," in *Proceedings of the 12th Conference on Asia South Pacific Design Automation, ASP-DAC*, 2007, pp. 938–943.

[4] P. Tafertshofer, A. Ganz, and M. Henftling, "A SAT-based implication engine for efficient ATPG, equivalence checking, and optimization of netlists," in *Proceedings of the 1997 IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, 1997, pp. 648–655.

[5] A. Groce, D. A. Peled, and M. Yannakakis, "Adaptive model checking," in *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS*, 2002, pp. 357–370.

[6] A. Borning, K. Marriott, P. J. Stuckey, and Y. Xiao, "Solving linear arithmetic constraints for user interface applications," in *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST*, 1997, pp. 87–96.

[7] T. Coatta and G. Neufeld, "Configuration management via constraint programming," in *International Workshop on Configurable Distributed Systems*, 1992.

[8] J. A. Hewson, P. Anderson, and A. D. Gordon, "A declarative approach to automated configuration," in *Strategies, Tools , and Techniques: Proceedings of the 26th Large Installation System Administration Conference, LISA*, 2012, pp. 51–66.

[9] J. C. Campos, M. D. Harrison, and K. Loer, "Verifying user interface behaviour with model checking," in *Verification and Validation of Enterprise Information Systems, Proceedings of the 2nd International Workshop on Verification and Validation of Enterprise Information Systems, VVEIS*, 2004, pp. 87–96.

[10] M. B. Dwyer, V. Carr, and L. Hines, "Model checking graphical user interfaces using abstractions," in *Software Engineering - ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering,*, 1997, pp. 244–261.

[11] A. C. R. Paiva, J. C. P. Faria, and R. F. A. M. Vidal, "Specification-based testing of user interfaces," in *Interactive Systems. Design, Specification, and Verification, 10th International Workshop, DSV-IS*, 2003, pp. 139–153.

[12] A. M. Memon and M. B. Cohen, "Automated testing of GUI applications: models, tools, and controlling flakiness," in *35th International Conference on Software Engineering, ICSE '13*, 2013, pp. 1479–1480.