Catalog of Metamorphic Relations
ICST 2020 submission

This document presents all the metamorphic relations in our catalog, using the formatting of the SMRL editor. For each metamorphic relation, java-like documentation is provided in the heading.

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp {

/**
 * A login operation should not succeed if performed on the HTTP
channel.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with three clauses joined with logical conjunctions.
 * The 1st clause checks if the current action performs a log in.
 * The 2nd clause defines the follow-up input.
 * The 3rd clause changes the channel of the login action in the
follow-up input.
 *
 * The 2nd parameter of IMPLIES checks if the output generated by
the login operation is different in the two cases.
 */
MR OTG_AUTHN_001 {
 {
   for ( Action action : Input(1).actions() ) {
      var pos = action.getPosition();
      IMPLIES(
         isLogin(action)                    //1st par (1st clause)
         && EQUAL ( Input(2), Input(1) )    //1st par (2nd clause)
         &&Input(2).actions.get(action.position).setChannel("http")
                                            //1st par (3rd clause)
         ,
         different ( Output(Input(1), pos), Output(Input(2), pos) )
                                            //2nd par of IMPLIES
       );//end-IMPLIES
   }//end-for
 }
}//end-MR
}//end-package
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp {

/**
 * Without being authenticated, a user should not be able to access
a page that normally can be reached only through the user interface
of authenticated users.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with three clauses joined with logical conjunction.
 * The 1st clause checks if the current action has been performed
after a log in.
 * The 2nd clause checks if the current action has ever been
performed by a non-authenticated user.
 * The 3rd clause defines a follow-up input that performs only the
given action, without logging in before.
 *
 * The 2nd parameter of IMPLIES checks if the output generated by
the action is different in the two cases.
 */
MR OTG_AUTHN_004 {
 {
   for ( Action action : Input(1).actions() ) {
      IMPLIES(
         afterLogin( action )    //1st par of IMPLIES (1st clause)
         && notVisibleWithoutLoggingIn( action.getUrl() )
                                 //1st par of IMPLIES (2nd clause)
         && EQUAL( Input(2), action )
                                 //1st par of IMPLIES (3rd clause)
         ,
         different(
            Output(Input(1), action.position ),
            Output(Input(2), action.position ) )
                                 //2nd par of IMPLIES
      );//end-IMPLIES
   }//end-for
 }
}//end-MR
}//end-package
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * Without begin authenticated, a user should not be able to access
a page that normally can be reached only through the user interface
of authenticated users.
 * This should be true even if she tries on the http channel (i.e.,
the result of a same operation being performed on a different
channel should be different).
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with four clauses joined with logical conjunction.
 * The 1st clause checks if the current action has been performed
after a login.
 * The 2nd clause checks if the current action is not occurring
already on the http channel.
 * The 3rd clause defines a follow-up input.
 * The 4th clause set the channel of the action the follow-up input
to "http".
 *
 * The 2nd parameter of the operator IMPLIES checks if the output
generated by the action is different in the two cases.
 */
MR OTG_AUTHN_010 {
 {
    for ( Action action : Input(1).actions() ){
        var pos = action.position;
        IMPLIES(
          afterLogin( action )   //1st par of IMPLIES (1st clause)
          &&!Input(1).actions().get(pos).getChannel().equals("http")
                                 //1st par of IMPLIES (2st clause)
          && EQUAL ( Input(2), Input(1) )
                                 //1st par of IMPLIES (3nd clause)
          && Input(2).actions().get(pos).setChannel("http")
                                 //1st par of IMPLIES (4rd clause)
          ,
          different( Output(Input(1),pos), Output(Input(2),pos) )
                                 //2nd par of IMPLIES
      );//end-IMPLIES
    }//end-for
 }
}//end-MR
}//end-package
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * A file path passed in a parameter should never enable a user to
access data that is not provided by the user interface.
 *
 * This metamorphic relation contains two nested loops; the first
iterates over the actions in the input sequence, the second iterates
over the parameters of the action.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with two clauses joined with a logical conjunction.
 * The 1st clause defines a follow-up input that is a copy of the
source input.
 * The 2nd clause sets the value of a parameter to a random file
path.
 *
 * The 2nd parameter of IMPLIES verifies the result. It is
implemented as an OR operation where the 1st parameter verifies that
the follow-up input leads to an error page.
 * The 2nd parameter deals with the case in which the generated
request is valid, and verifies that the returned content is
something that the user has the right to access.
 */
MR OTG_AUTHZ_001a {
 {
   for ( Action action : Input(1).actions() ){
      for (var par=0; par < action.getParameters().size(); par++){
         var pos = action.getPosition();
         IMPLIES(
            EQUAL( Input(2), Input(1) )
                           //1st par of IMPLIES (1st clause)
            && Input(2).actions().get(pos)
                .setParameterValue(par, RandomFilePath())
                           //(2nd clause)
            ,
            //2nd par of IMPLIES, OR operator receiving 2 parameters
            OR(
               Output(Input(2),pos).isError()    //1st par of OR
               ,
               userCanRetrieveContent(
                      action.getUser(),
                      Output(Input(2),pos)) )    //2nd par of OR
         );//end-IMPLIES
```

```
      }//end-for
    }//end-for
  }
}//end-MR
}//end-package
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * A file path passed in the URL of a request should never enable a
 * user to access data that is not provided by the user interface.
 * This metamorphic relation contains two nested loops; the first is
used to introduce relative paths in the query (jumps to parent
folders), the second iterates over the actions in the input
sequence.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with three clauses joined with a logical conjunction.
 * The 1st clause verifies whether the current action has been not
performed by an administrator.
 * The 2nd clause checks if the current action has been performed
after a login.
 * The 3rd clause defines a follow-up input that is a copy of the
source input.
 * The 4th clause adds to the end of the current URL a relative path
to a file.
 * The 5th clause verifies that the given path was not tried in a
previous execution of the loop (to speed up).
 *
 * The 2nd parameter of IMPLIES verifies the result. It is
implemented as an OR operation where
 * The 1st parameter verifies that the follow-up input does not lead
to a file.
 * The 2nd parameter deals with the case in which the generated
request is valid, and verifies that the returned file is something
that the user has the right to access.
 * The 3rd parameter verifies that the follow-up input leads to an
error page.
*/
MR OTG_AUTHZ_001b {
 {
   var sep="/";
      for ( var par=0; par < 4; par++ ){
         for ( Action action : Input(1).actions() ){
            var pos = action.getPosition();
            var newUrl = action.urlPath+sep+RandomFilePath();
            IMPLIES(
               !isAdmin(action.user) &&
                     //1st clause of IMPLIES (1st par)
               afterLogin(action) &&
```

```
                    //2nd clause of IMPLIES (1st par)
                EQUAL( Input(2), Input(1) ) &&
                    //3rd clause of IMPLIES (1st par)
                Input(2).actions().get(pos).setUrl( newUrl ) &&
                    //4th clause of IMPLIES (1st par)
                notTried( action.getUser(), newUrl )
                    //5th clause of IMPLIES (1st par)
                ,
                TRUE ( //2nd par of IMPLIES
                   //TRUE operator receiving 3 clauses
                   Output(Input(2),pos).noFile() || //1st par of TRUE
                   userCanRetrieveContent(          //2nd par of TRUE
                      action.getUser(),
                      Output(Input(2),pos).file()) ||
                   Output(Input(2),pos).isError()   //3rd par of TRUE
               );//end-IMPLIES
            }//end-for
            sep=sep+"../";
         }//end-for
      }
   }//end-MR
}
```

```
import static smrl.mr.language.Operations.*
import smrl.mr.language.Action;

package smrl.mr.owasp {

  /**
   * A URL that cannot be reached by a user while navigating the
user interface should not be available to that same user even when
she directly requests the URL to the server.
   * For this reason, an input sequence that is valid for a given
user, should not lead to the same output when it is executed by
another user, if it includes access to a URL with these
characteristics.
   *
   * The metamorphic relation iterates over all the actions of an
input sequence.
   *
   * The 1st parameter of IMPLIES is made of three clauses.
   * The 1st clause checks whether the user in User() is not a
supervisor of the user performing the current action.
   * The 2nd clause verifies that the user cannot retrieve the URL
of the action through the GUI (based on the data collected by the
crawler).
   * The 3rd clause defines a follow-up input that matches the
source input except that the credentials of User() are used in this
case.
   *
   * The 2nd parameter of IMPLIES verifies the result. It is
implemented as an OR operation where
   * The 1st parameter verifies that the follow-up input leads to an
error page.
   * The 2nd parameter verifies that the output generated by the
action containing the URL indicated above leads to two different
outputs in the two cases.
   */
MR OTG_AUTHZ_002 {
 {
   for ( Action action : Input(1).actions() ){
      IMPLIES(
         //1st par of IMPLIES
         (!isSupervisorOf(User(), action.user)) &&
         cannotReachThroughGUI( User(), action.url ) &&
         EQUAL( Input(2), changeCredentials(Input(1), User()) )
         ,
         OR(     //2nd par of IMPLIES
            isError(Output(Input(1),action.position)),
            NOT(Output(Input(1),action.position).equals(
```

```
                Output(Input(2),action.position)))
       )); //end-IMPLIES
    } //end-for

 }
 } //end-MR
}//end-package
```

```
import static smrl.mr.language.Operations.*
import smrl.mr.language.actions.ClickOnNewRandomElement
import smrl.mr.language.Action;

package smrl.mr.owasp {

/**
 * If a redirecting URL cannot be reached by a user while navigating
the user interface, the same URL, if directly requested to the
server, should not enable the same user to access a page where the
click on one of its elements (e.g., a warning message) enables the
user to access the content of the URL.
 *
 * The metamorphic relation iterates over all the actions of an
input sequence.
 *
 * The 1st parameter of IMPLIES is made of three clauses.
 * The 1st clause checks whether the user in User() is not a
supervisor of the user performing the current action.
 * The 2nd clause verifies that the user cannot retrieve the URL of
the action through the GUI (based on the data collected by the
crawler).
 * The 3rd clause defines a follow-up input that matches the source
input except that the credentials of User() are used in this case.
 *
 * The 2nd parameter of IMPLIES verifies the result. It is made of
three clauses.
 * The 1st clause verifies that the original URL does not perform
any redirect.
 * The 2nd clause verifies that the original URL does not perform
any redirect.
 * The 3rd clause verifies that the follow up input does not lead to
the same redirect from the original input.
 */
MR OTG_AUTHZ_002a {
 {
   for ( Action action : Input(1).actions() ){
      var pos = action.getPosition();
      IMPLIES(
         (!isSupervisorOf(User(), action.user)) && // 1st par
         cannotReachThroughGUI(User(), action.url) &&
         EQUAL( Input(2), changeCredentials( Input(1), User() ) )
         ,
         ( Output(Input(1), pos).redirectURL()===null ||
           Output(Input(2), pos).redirectURL()===null ) ||
           NOT(
            EQUAL (
```

```
                Output(Input(2), pos).redirectURL(),
                Output(Input(1), pos).redirectURL()))
        ); //end-IMPLIES
    }//end-IMPLIES
  }
  } //end-MR
} //end-package
```

```
import static smrl.mr.language.Operations.*

package smrl.mr.owasp {

   /**
   * If a certain action is not available to a given user, this user
should not be able to perform the action.
   *
   * Assume we have two users, user a and user b. Given (1) a source
input as a sequence of actions performed by a user 'a' which
contains an action y that is dedicated to user a (i.e., it is not
visible to user b) and (2) a follow-up input that is a copy of that
sequence which, however, includes, before action y, an action that
matches action y (e.g., same URL requested) but is performed by user
'b'.
   * The result of action y should not be different when performed
in the source input (i.e., without any action of b) or in the
follow-up input (i.e., when performed also by user b).
   * In other words, the action of user b should be ignored by the
system and not interfere with the action of user a.
   *
   * This MR contains two loops. The first iterates over the actions
of the source input to identify a login operation (action x) for
user a, the second iterates over the remaining y-th actions.
   *
   * The 1st parameter of implies defines the follow-up input.
   * The 1st clause checks whether the user in User() is not a
supervisor of the user performing the y-th action.
   * The 2nd clause checks that action y cannot be accessed by user
b (User())
   * The 3rd clause defines Input(2) which just performs a login.
   * The 4th clause defines Input(3) which just performs a login as
user b.
   * The 5th clause creates a copy of Input(1) with a login as b
before action y (this way action y is performed as User b).
   * The 6th clause adds after action y+1 (the original action y now
shifted) new copy of action y (now performed by user b).
   * The 7th clause adds after the new copy of action y a new login
as user a.
   *
   * The 2nd parameter of IMPLIES checks that the output of the
action y in the two sequences remains the same when performed by
user a (in the follow-up sequence the action of user a is shifted by
three because three actions are introduced, the login of user b, the
current action and a new login for user a).
   */
MR OTG_AUTHZ_002b {
```

```
 {
   for(var x = 0; x < Input(1).actions().size() ; x++){
      for (var y = x+1;
         isLogin(Input(1).actions().get(x)) &&
          (y < Input(1).actions().size()); y++) {
         IMPLIES(
            //1st par of IMPLIES including 7 clauses
            (!isSupervisorOf(
               User(),
               Input(1).actions().get(y).user)) &&
            cannotReachThroughGUI(
               User(),
               Input(1).actions().get(y).getUrl()) &&
            EQUAL(Input(2), Input(1).actions().get(x)) &&
            EQUAL(Input(3), changeCredentials(Input(2), User())) &&
            EQUAL(
               Input(4), addAction(Input(1), y,
               Input(3).actions().get(0))) &&
            EQUAL(
               Input(5), addAction(Input(4), y+1,
               Input(1).actions().get(y))) &&
            EQUAL(
               Input(6), addAction(Input(5), y+2,
               Input(1).actions().get(x) ) )
            ,
            //2nd par of IMPLIES
            EQUAL(
               Output(Input(1), y)
               ,
               Output(Input(6), y+3 ))
         ); //end-IMPLIES
      } //end-for
   } //end-for
 }
 } //end-MR
} //end-package
```

```
import static smrl.mr.language.Operations.*

package smrl.mr.owasp {

/**
   * A URL that cannot be reached by a user while navigating the
user interface should not be available to that same user even when
she directly requests the URL to the server.
   *
   * The metamorphic relation iterates over all the actions of an
input sequence.
   *
   * The 1st parameter of IMPLIES is made of four clauses.
   * The 1st clause checks whether the user in User() is not a
supervisor of the user performing the y-th action.
   * The 2nd clause verifies that the y-th action is performed after
a login.
   * The 3rd clause verifies that the follow-up user cannot retrieve
the URL of the action through the GUI (based on the data collected
by the crawler).
   * The 4th clause defines a follow-up input that performs the
login as the follow-up user and then performs the given action.
   *
   * The 2nd parameter of IMPLIES verifies the result. It is
implemented as an OR operation where
   * The 1st parameter checks if the y-th action from the source
input leads to an error page.
   * The 2nd parameter verifies if the output generated by the
action containing the URL indicated above, lead to two different
outputs in the two cases.
   */
MR OTG_AUTHZ_002c {
 {
   for(var y = Input(1).actions().size()-1; ( y > 0 ); y--){
      IMPLIES(
         //1st par of IMPLIES including 4 clauses
          (!isSupervisorOf(
            User(),
            Input(1).actions().get(y).user)) &&
         afterLogin(Input(1).actions().get(y)) &&
         cannotReachThroughGUI(
            User(),
            Input(1).actions().get(y).getUrl()) &&
         EQUAL(
            Input(2),
            Input(loginAction(User()), Input(1).actions().get(y)))
         ,
```

```
            //2nd par of IMPLIES
            OR(
                isError(Output(Input(1), y)),
                different(
                    Output(Input(1), y),
                    Output(Input(2), 1)
                )
            )
        ); //end-IMPLIES
    } //end-for
 }
 } //end-MR
} //end-package
```

```
import static smrl.mr.language.Operations.*

package smrl.mr.owasp {

 /**
 * A user should not be able to overwrite an admin file by writing
its path in a file form.
 *
 * The first loop iterates over all the actions of an input
sequence.
 * The second loop looks for an action that contains a form that
appear to be used to specify paths.
 *
 * The 1st parameter of IMPLIES is made of four clauses.
 * The 1st clause verifies that the follow-up user is not an admin
(admin may access any file).
 * The 2nd clause verifies that the selected text input in a form
contains a file path (or a file name).
 * The 3rd clause verifies defines a follow-up input that is a copy
of the source input.
 * The 4th clause puts a randomly selected path of an admin file in
the selected form input of the follow-up sequence.
 *
 * The 2nd parameter of IMPLIES verifies the result. It is
implemented as an OR operation where
 * The 1st parameter checks if the y-th action from the source input
leads to an error page.
 * The 2nd parameter verifies if the output generated by sequence
containing the path to the admin file is different than the output
of the sequence performed by the original user. We do not check only
for the output of action x because the error might be observed
afterwards (e.g., during execution).
 */
MR OTG_AUTHZ_002d {
 {
   for ( var x=0; Input(1).containFormInputForFilePath() &&
         x<Input(1).actions().size; x++) {
      var action = Input(1).actions.get(x);
      var randomPath = RandomAdminFilePath();
      var formInputs = action.getFormInputs();
      for(var i=0; action.containFormInputForFilePath() &&
            i< formInputs.size(); i++){
         var formInput =
               action.getFormInputs().get(i).getAsJsonObject();
         IMPLIES(
            //1st par of IMPLIES including 4 clauses
            ! isAdmin(action.user) &&
```

```
            isFormInputForFilePath( formInput ) &&
            EQUAL(Input(2), Input(1)) &&
            updateStringFormInput(
               Input(2).actions.get(x).getFormInputs()
                  .get(i).getAsJsonObject(),
               randomPath)
            ,
            //2nd par of IMPLIES
            OR(
               isError(Output(Input(1))),
               different(
                  Output(Input(1)),
                  Output(Input(2))
               )
            ); //end-IMPLIES
         } //end-for
      } //end-for
   }
   } //end-MR
} //end-package
```

```
import static smrl.mr.language.Operations.*

/**
 * This MR matches OTG_AUTHZ_002b with the difference that in this
case we check for the content provided by action (y+1).
 *
 * The MR verifies that the content provided to the original user is
either an error or is not anomalous (i.e., was already observed).
 */
package smrl.mr.owasp {
MR OTG_AUTHZ_002e {
 {
  for(var x = 0; (x < Input(1).actions().size() ); x++){
   for (var y = x+1;
     (isLogin(Input(1).actions().get(x)) &&
     y < Input(1).actions().size()); y++) {
     IMPLIES(
      //1st par of IMPLIES including 5 clauses
      (!isSupervisorOf(User(), Input(1).actions().get(y).user)) &&
     cannotReachThroughGUI(
      User(),
      Input(1).actions().get(y).getUrl()) &&
     EQUAL(Input(2), Input(1).actions().get(x)) &&
     EQUAL(Input(3), changeCredentials(Input(2), User())) &&
     EQUAL(
      Input(4),
      addAction(Input(1), y, Input(3).actions().get(0)))
     ,
     //2nd par of IMPLIES
     OR(
      userCanRetrieveContent(
       Input(4).actions().get(y+1).getUser(),
       Output(Input(4), y+1))
      ,
      Output(Input(4), y+1).isError()
     )
   ); //end-IMPLIES
   } //end-for
  } //end-for
 }
 } //end-MR
} //end-package
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * If an action is not expected to be executed by a user 'u'
(because is not available in his GUI), then user 'u' should not be
able to execute that action even if he changes the user id parameter
in that action (i.e., the action should lead to different results
when executed by a valid and invalid user).
 *
 * The first loop iterates over all the actions of the input
sequence.
 * The second iterates over all the parameters of the action to
identify a parameter that specifies the user id.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with four clauses joined with logical conjunctions.
 * The 1st clause checks if the current action contains a user ID.
 * The 2nd clause defines the follow-up input as a copy of the
source input.
 * The 3rd clause changes the user ID to the one of User
 * The 4th clause changes the login credentials of the follow-up
input to the one of User
 *
 * The 2nd parameter of IMPLIES checks if the output generated by
the action is different in the two cases.
 */
MR OTG_AUTHZ_003 {
{
   for ( Action action : Input(1).actions() ){
      for (var par=0; par < action.getParameters().size(); par++ ){
         var pos = action.getPosition();
         IMPLIES (
            isUserIdParameter(action,par,action.getUser() ) &&
            ( equal ( Input(2), Input(1) ) &&
            Input(2).actions().get(pos)
               .setParameterValue(par,User()) ) &&
            equal (Input(3), changeCredentials(Input(1), User()) )
            ,
            different (Output(Input(2),pos), Output(Input(3),pos)) )
      }
   }
 }
}}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * By randomly changing the parameter values passed to URLs, a user
should not be able to retrieve content she cannot retrieve from GUI.
 *
 * The first loop iterates over all the actions of the input
sequence.
 * The second iterates over all the parameters of the action.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with two clauses joined with logical conjunctions.
 * The 1st clause defines the follow-up input.
 * The 2nd clause set a parameter value to a random value.
 *
 * The 2nd parameter of IMPLIES checks if the content of the output
generated by the login operation is either an error message or some
content that can be retrieved from the GUI.
 */
MR OTG_AUTHZ_004 {
 {
   for ( Action action : Input(1).actions() ){
      for (var par=0; par < action.getParameters().size(); par++){
         var pos = action.getPosition();
         IMPLIES(
            EQUAL ( Input(2), Input(1) )
                        //1st par of IMPLIES (1st clause)
            &&  Input(2).actions().get(pos)
                        //2nd par of IMPLIES (2nd clause, 1st part)
               .setParameterValue(par,
                        //2nd par of IMPLIES (2nd clause, 2nd part)
               RandomValue( typeOf( action.getParameterValue(par))))
                        //2nd par of setParameterValue
            ,
            OR( Output(Input(2),pos).isError()
               ,
               userCanRetrieveContent(action.user,
                  Output(Input(2),pos)))
         );//end-IMPLIES
      }//end-for
   }//end-for
 }
}//end-MR
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * Some URLs are expected to be used only once. These URLs can be
identified (by the data collection framework) by checking if a same
action (e.g., clicking on a button) triggers always different
(e.g., the button URL is always different) over different
executions.
 * In this case a user should not be able to reuse the URL multiple
times (e.g., sending POST data to the same URL).
 *
 * The loop iterates over all the actions of the input.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with four clauses joined with logical conjunctions.
 * The 1st clause checks if the URL of the current action changes
over multiple executions.
 * The 2nd clause defines the follow-up input as a copy of the
source input where the action above is duplicated.
 *
 * The 2nd parameter of IMPLIES checks if the output generated by
the second action different than in the case of the first action.
 */
MR OTG_BUSLOGIC_005 {
 {
   for ( Action action : Input(1).actions() ){
      var pos = action.getPosition();
      IMPLIES( (
         urlOfActionChangesOverMultipleExecutions( action )
         && equal ( Input(2), addAction( Input(1), pos, action ))
         ,
         different( Output(Input(1),pos), Output(Input(2), pos) ) )
   }
 }
}
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp {

/**
 * An action with strict transport security header should not be
available on the http channel.
 *
 * The loop iterates over all the actions of the input.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with three clauses joined with logical conjunctions.
 * The 1st clause defines the follow-up input.
 * The 2nd clause checks if the output of the source input has
strict transport security header.
 * The 3rd clause set the channel of the action to http
 *
 * The 2nd parameter of IMPLIES checks that if the modified action
has not been redirected to httpsthen the output generated by the
action should be different than in the case of the source input.
 */
MR OTG_CONFIG_007 {
 {
   for ( Action action : Input(1).actions() ) {
      var pos = action.getPosition();
      IMPLIES(
          ( equal ( Input(2) , Input(1) ) &&
         Output(Input(1),pos).hasStrictTransportSecurityHeader() &&
         Input(2).actions().get(pos).setMethod("http") )
         ,
         AND (
            equal ( Output(Input(2),pos).getChannel(), "https" ),
            equal ( Output(Input(1),pos) , Output(Input(2),pos))))
   }
 }
}
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp {

/**
 * Weak encryption algorithms should not be available.
 *
 * The loop iterates over all the actions of the input.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with three clauses joined with logical conjunctions.
 * The 1st clause checks if the action works on the encrypted
channel.
 * The 2nd clause defines a follow-up input.
 * The 3rd clause set the encryption algorithms to a weak one.
 *
 * The 2nd parameter of IMPLIES checks that the output generated by
the action using the weak encryption algorithm lead to different
results.
 */
MR OTG_CRYPST_004 {
 {
   for ( Action action : Input(1).actions() ){
      IMPLIES (
          (isEncrypted( action ) &&
         equal ( Input(2) , Input(1) ) &&
         Input(2).actions().get(action.position)
               .setEncryption( WeakEncryption() ) )
         ,
         different ( Output( Input(1) ), Output( Input(2) ) ) )
   }
 }
}
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp {

/**
 * This MR checks that  actions available with one HTTP method
(e.g., POST ) should not be available with another method (e.g.,
DELETE).
 *
 * The metamorphic relation iterates over all the actions of an
input sequence.
 *
 * The 1st parameter of IMPLIES is made of two clauses.
 * The 1st clause verifies that the user cannot retrieve the URL of
the action through the GUI (based on the data collected by the
crawler).
 * The 2nd clause defines a follow-up input in which the selected
action is performed using a different HTTP method.
 *
 * The 2nd parameter of IMPLIES verifies that the output generated
by the modified action is different in the two cases.
 */
MR OTG_INPVAL_003 {
 {
   for ( Action action : Input(1).actions() ) {
      var pos = action.getPosition();
      IMPLIES(
         ( EQUAL( Input(2) , Input(1) ) &&
         Input(2).actions().get(pos).setMethod( HttpMethod() ))
         ,
         different ( Output(Input(1),pos),  Output(Input(2),pos) ))
   }
 }
}
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * Duplicating a parameter value should not lead to a different
behaviour in the system.
 *
 * The first loop iterates over all the actions of an input
sequence.
 * The second loop iterates over all the parameters.
 *
 * The 1st parameter of IMPLIES is made of two clauses.
 * The 1st clause defines a follow-up input.
 * The 2nd clause duplicates one parameter.
 *
 * The 2nd parameter of IMPLIES verifies that the output generated
by the modified action is the same in the two cases.
 */
MR OTG_INPVAL_004 {
 {
   for ( Action action : Input(1).actions() ){
      for (var par=0; par < action.getParameters().size(); par++ ){
         var pos = action.getPosition();
         IMPLIES (
            ( equal ( Input(2), Input(1) ) &&
            Input(2).actions().get(pos).addParameter(
                     action.getParameterName(par),
                     action.getParameterValue(par) ))
            ,
            equal ( Output(Input(1) ) , Output( Input(2) )))
      }
   }
 }
}
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
* A login action performed by a user already authenticated should
always trigger
 * the generation of a new session ID.
 *
 * This metamorphic relation contains two nested loops; the first
iterates over the inputs to find a sign up action, the second
iterates over the actions that follow the sign up.
 *
 * The second loop is necessary to check that a sign up action
repeated at any point of the action sequence leads to a new session
ID.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with two clauses joined with logical conjunction.
 * The 1st clause checks if the current action has been performed
after a login.
 * The 2nd clause defines a follow-up input with the sign up action
being duplicated in a certain position.
 *
 * The 2nd parameter of IMPLIES checks if the session ID of the
response page sent after the two successive login actions is
different.
 */
MR OTG_SESS_003 {
 {
   for( Action signup : Input(1).actions() ){
      for ( var i=0;
         isSignup(signup) && i < Input(2).actions().size; i++ ) {
         var f = Input(2).actions().get(i);
         var pos = f.getPosition();
         IMPLIES(
            afterLogin( f ) &&   //1st par of IMPLIES (1st clause)
            EQUAL(
               Input(3),
               addAction( Input(2), pos+1, signup ))  //(2nd clause)
            ,
            different(     //2nd par of IMPLIES
               Output(Input(3), pos).getSession(),
               Output(Input(3), pos+1).getSession())
         );//end-IMPLIES
       }//end-for
```

```
    }//end-for
  }
  }//end-MR
}
```

```
import static smrl.mr.language.Operations.*;

package smrl.mr.owasp{

/**
 * A logout action should always lead to a new session.
 *
 * This MR iterates over all the actions to find a logout action.
 * The second loop iterates over all the actions to find an action
performed after login.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with two clauses joined with logical conjunction.
 * The 1st clause checks if the current action x is a logout
operation.
 * The 2nd clause checks that the action y is performed after a
login.
 * The 3rd clause checks that the action y is not a login.
 * The 4th clause defines a follow-up input with the logout action
being duplicated in position y.
 *
 * The 2nd parameter of IMPLIES checks if the session ID before and
after executing the logout is different.
 */
MR OTG_SESS_006 {
{
   for ( var x=0; x < Input(1).actions().size() ; x++ ){
      for ( var y=0; y < x ; y++ ){
         IMPLIES (
            isLogout( Input(1).actions().get(x) ) &&
            afterLogin( Input(1).actions().get(y) ) &&
            ! isLogin( Input(1).actions().get(y) ) &&
            EQUAL ( Input(2) , copyActionTo( Input(1), x, y ) )
            ,
            different(Session(Input(2),y-1), Session(Input(2),y))) ;
      }
   }
 }
}
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * After a session timeout the user should not be able to perform an
action that requires to be logged in.
 *
 * This MR iterates over all the actions to find actions executed
within a session, after login.
 *
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with three clauses joined with logical conjunction.
 * The 1st clause checks that the action is generally not available
without login.
 * The 2nd clause checks if the session is not null.
 * The 3rd clause checks that a session timeout is set
 * The 4th clause defines a follow-up input where the selected
action is executed after timeout (usually simulated).
 *
 * The 2nd parameter of IMPLIES checks if the output of the action
generated after timeout is different than in the case in which it is
executed before the timeout.
 */
MR OTG_SESS_007 {
 {
   for ( Action action : Input(1).actions() ){
      IMPLIES (
         notAvailableWithoutLoggingIn(action) &&
         NOT ( NULL ( action.session ) ) &&
         action.session.timeout > 0 &&
         EQUAL ( Input(2) ,
            addAction( Input( 1 ),
               action.position,
               Wait(action.session.timeout) ))
         ,
         different (
            Output( Input(1), action.position ),
            Output( Input(2), action.position ) ));
   }
 }
}
}
```

```
import static smrl.mr.language.Operations.*;
import smrl.mr.language.Action;

package smrl.mr.owasp{

/**
 * An action that (1) is available without logging in and (2)
generates a session, should not enable a user to execute an action
that requires to be logged in.
 *
 * This MR iterates over all the actions of the input.
 *
 * The 1st parameter of the operator IMPLIES is a boolean expression
with three clauses joined with logical conjunction.
 * The 1st clause checks that the current action is not available
without being logged in.
 * The 2nd clause looks for an action available without being
logged-in that generates a session.
 * The 3rd clause defines a follow-up input that executes two
actions, the action available without being logged in, and
 * the selected action (i.e., the one available only by being
logged-in).
 *
 * The 2nd parameter of IMPLIES checks that the output of the action
is different when execute with and without being logged in (even if
after an action that does not require a log-in but generates a
session).
 */
MR OTG_SESS_008 {
{
   for ( Action action : Input(1).actions() ){
      IMPLIES(
         notAvailableWithoutLoggingIn( action ) &&
         NOT (NULL(ActionAvailableWithoutLogin().getSession() ) ) &&
         EQUAL( Input(2) ,
            Input( ActionAvailableWithoutLogin(), action ) )
         ,
         different (
            Output( Input(1), action.position ),
            Output( Input(2), 1 ) ) );
   }
 }
}
}
```