Documentation of MRs for Testing Constraint Checking Implementations

This documentation is being developed now.

June 1, 2023

1 Symbol Conventions

For ease of narrative, in the following sections, we use D_0 and S_0 to denote the context pool and the constraint in source test input, Rt_0 and Rl_0 to denote the truth value and the link set in the source test output. Similarly, we use D_1, S_1, Rt_1 and Rl_1 to denote corresponding variables in the follow-up test.

2 MR-Data

There are two different MRs in this part, and they only transform the data under checking and keep the constraint unchanged.

2.1 Link

Before introducing the MR-Data, let us talk more about links.

Link explains how a constraint is violated or satisfied. A link is composed of two parts *type* and *bindings*. Type of a link is either *vio* or *sat*, indicating that this link is a violated link or a satisfied link respectively. Bindings of a link is a set of variable assignment, which can help explain how a constraint is violated or satisfied. For example, $l_1 :< vio, \{x = 1, y = 2\} >$ and $l_2 :< vio, \{x = 2, y = 2\} >$ are two different links. The type of l_1 is *vio* and the bindings of l_1 is $\{x = 1, y = 2\}$.

For ease of description, we use the following notation in this documentation. Let l be a link and $ls = \{l_1, l_2, ..., l_n\}$ be a link set. We use $type(l)$ to denote the type of l and $bindings(l)$ to denote the bindings of l. If $type(l_1) == type(l_2) \wedge bindings(l_1) \subseteq bindings(l_2)$, we say l_2 "covers" l_1 and denote it as $l_1 \preceq l_2$. If $\forall l_i \in ls_1 (\exists l_j \in ls_2 (l_i \preceq l_j))$, we say ls_2 "covers" ls_1 and denote it as $ls_1 \preceq ls_2$. It is easy to see that $\preceq$ is partial order relation and the inclusion relation of link sets ($\subseteq$) is a special case of the "cover" relation ($\preceq$). Besides that, we use $var(l)$ and $var(ls)$ to denote the set of the variables appeared in a link or a link set. And we use $val(ls, x)$ to denote all the value assigned to the variable x.

To illustrate these concepts more clearly, let's look at the example in Fig 1. $ls_1 \preceq ls_2$ is easy to understand because the only link l_1 in ls_1 is also in ls_2 and $l_1 \preceq ls_1$. On the other hand, $l_1 \preceq l_3$ and $l_2 \preceq l_4$, so $ls_2 \preceq ls_3$.

For more details of the rules for calculating links, please refer to the existing work[1].

2.2 Impact of data changes

Then, we talk about the impact of data changes.

According to the existing work[2], impact of data changes can divide the into two categories: inconsistency incurring and inconsistency-hidden(also named as inc+/inc-). The inconsistency incurring changes leads to more possibilities for incurring violation and inconsistency hidden changes

Let:

$$l_1 =< vio, \{x = 1\} >$$

$$l_2 =< vio, \{y = 4\} >$$

$$l_3 =< vio, \{x = 1, y = 3\} >$$

$$l_4 =< vio, \{x = 2, y = 4\} >$$

$$ls_1 = \{l_1\}$$

$$ls_2 = \{l_1, l_2\}$$

$$ls_3 = \{l_3, l_4\}$$

then:

$$ls_1 \preccurlyeq ls_2 \preccurlyeq ls_3$$

$$var(ls_3) = \{x, y\}$$

$$val(ls_3, x) = \{1,2\}$$

Figure 1: An example of cover relation among link sets

has the opposite effect. These two kinds of impacts of data changes depend on the constraint and can be derived from the constraint in a static way.

For example, constraint $S : \forall x \in X(not(\exists y \in Y(same(X, Y))))$ has the inconsistency-incurring changes set $\{< +, X >, < +, Y >\}$, which means if we add any element to X or Y, constraint s would tend to generate more violation.

For more details of the derivation rules for two impact of data changes, please refer to the existing work[2].

2.3 MR-Data 1

Now we can elaborate our MR-Data 1 as follow:

MR-Data 1: If $Rt_0 == F$, we apply any change with an inconsistency incurring impact on S_0 to D_0 to obtain D_1. Then, in the follow-up execution, Rt_1 should still be F and Rl_1 should satisfy the condition: $Rl_0 \preceq Rl_1$.

2.4 MR-Data 2

MR-Data 2 is the other side of MR-Data 1:

MR-Data 2: If $Rt_0 == T$, we apply any change with an inconsistency hidden impact on S_0 to D_0 to obtain D_1. Then, in the follow-up execution, Rt_1 should still be T and Rl_1 should satisfy the condition: $Rl_0 \preceq Rl_1$.

3 MR-Cons

There are nine different MRs in this part, and they only transform the constraint for checking and keep the data unchanged.

3.1 MR-Cons 1

MR-Cons 1: If the root of the constraint S_0 is $\forall v \in C$, we transform it to $\exists v \in C$ to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 3 cases:

Case 1: If Rt_0 is T, Rt_1 should be T and $val(Rl_1, v) == C$.

Case 2: If Rt_0 is F and $val(Rt_0, v) == C$, Rt_1 should be F and Rt_1 should be empty.
Case 3: If Rt_0 is F and $val(Rt_0, v) \subset C$, Rt_1 should be T and $val(Rt_1, v) == C - val(Rt_0, v)$.

3.2 MR-Cons 2

MR-Cons 2: If the root of the constraint S_0 is $\exists v \in C$, we transform it to $\forall v \in C$ to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 3 cases:
Case 1: If Rt_0 is F, Rt_1 should be F and $val(Rl_1, v) == C$.
Case 2: If Rt_0 is T and $val(Rt_0, v) == C$, Rt_1 should be T and Rt_1 should be empty.
Case 3: If Rt_0 is T and $val(Rt_0, v) \subset C$, Rt_1 should be F and $val(Rt_1, v) == C - val(Rt_0, v)$.

3.3 MR-Cons 3

Before we introduce more MRs, let us talk more about the links of binary quantifier. Let $S : f_l\ op\ f_r$ be a constraint, where *op* is a binary quantifier (e.g. *and*, *or*, *implies*) and f_l/f_r is the left/right sub-constraint of *op*. According to the rules of link generation [1], the links of S (denoted as ls) is one of the following 5 cases: (1). ls is empty. (2). ls only contains links from left sub-constraint (denoted as ls_{f_l}). (3). ls only contains links from right sub-constraint (denoted as ls_{f_r}). (4). ls is the union of ls_{f_l} and ls_{f_r} (5). ls is the cartesian product of ls_{f_1} and ls_{f_r}. For the convenience of description, we use the $flag(ls)$ to denote these cases. For example, if $flag(ls) == 2$, it means links composition of constraint S is case (2) above, i.e. ls only contains links from its left sub-constraint.

Now we can explain our MR-Cons 3 as follows:

MR-Cons 3: If the root of the constraint S_0 is *and*, which can be formed of f_1 *and* f_2, we transform it to f_1 *or* f_2 to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 8 cases:
Case 1: If Rt_0 is T and $flag(Rl_0) == 1$, Rt_1 should be T and $flag(Rl_1) == 1$.
Case 2: If Rt_0 is T and $flag(Rl_0) == 2$, Rt_1 should be T and $flag(Rl_1) == 2$.
Case 3: If Rt_0 is T and $flag(Rl_0) == 3$, Rt_1 should be T and $flag(Rl_1) == 3$.
Case 4: If Rt_0 is T and $flag(Rl_0) == 5$, Rt_1 should be T and $flag(Rl_1) == 4$.
Case 5: If Rt_0 is F and $flag(Rl_0) == 4$, Rt_1 should be F and $flag(Rl_1) == 5$.
Case 6: If Rt_0 is F and $flag(Rl_0) == 2$, either $Rt_1 == F \wedge flag(Rl_1) == 2$, or $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 3)$.
Case 7: If Rt_0 is F and $flag(Rl_0) == 3$, either $Rt_1 == F \wedge flag(Rl_1) == 3$, or $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2)$.
Case 8: If Rt_0 is F and $flag(Rl_0) == 1$, either $Rt_1 == F \wedge flag(Rl_1) == 1$, or $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2 \vee flag(Rl_1) == 3)$.

3.4 MR-Cons 4

MR-Cons 4: If the root of the constraint S_0 is *or*, which can be formed of f_1 *or* f_2, we transform it to f_1 *and* f_2 to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 8 cases:
Case 1: If Rt_0 is F and $flag(Rl_0) == 1$, Rt_1 should be F and $flag(Rl_1) == 1$.
Case 2: If Rt_0 is F and $flag(Rl_0) == 2$, Rt_1 should be F and $flag(Rl_1) == 2$.
Case 3: If Rt_0 is F and $flag(Rl_0) == 3$, Rt_1 should be F and $flag(Rl_1) == 3$.
Case 4: If Rt_0 is F and $flag(Rl_0) == 5$, Rt_1 should be F and $flag(Rl_1) == 4$.
Case 5: If Rt_0 is T and $flag(Rl_0) == 4$, Rt_1 should be T and $flag(Rl_1) == 5$.
Case 6: If Rt_0 is T and $flag(Rl_0) == 2$, either $Rt_1 == T \wedge flag(Rl_1) == 2$, or $Rt_1 == F \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 3)$.
Case 7: If Rt_0 is T and $flag(Rl_0) == 3$, either $Rt_1 == T \wedge flag(Rl_1) == 3$, or $Rt_1 == F \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2)$.

Case 8: If Rt_0 is T and $flag(Rl_0) == 1$, either $Rt_1 == T \wedge flag(Rl_1) == 1$, or $Rt_1 == F \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2 \vee flag(Rl_1) == 3)$.

3.5 MR-Cons 5

MR-Cons 5: If the root of the constraint S_0 is *or*, which can be formed of f_1 *or* f_2, we transform it to f_1 *implies* f_2 to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 8 cases:

Case 1: If Rt_0 is F and $flag(Rl_0) == 1$, Rt_1 should be T and $flag(Rl_1) == 1$.

Case 2: If Rt_0 is F and $flag(Rl_0) == 2$, Rt_1 should be T and $flag(Rl_1) == 2$.

Case 3: If Rt_0 is F and $flag(Rl_0) == 3$, Rt_1 should be T and $flag(Rl_1) == 1$.

Case 4: If Rt_0 is F and $flag(Rl_0) == 5$, Rt_1 should be T and $flag(Rl_1) == 2$.

Case 5: If Rt_0 is T and $flag(Rl_0) == 4$, Rt_1 should be T and $flag(Rl_1) == 3$.

Case 6: If Rt_0 is T and $flag(Rl_0) == 3$, Rt_1 should be T and $flag(Rl_1) == 3 \vee flag(Rl_1) == 4$.

Case 7: If Rt_0 is T and $flag(Rl_0) == 2$, either $Rt_1 == T \wedge flag(Rl_1) == 1$, or $Rt_1 == F \wedge (flag(Rl_1) == 2 \vee flag(Rl_1) == 5)$.

Case 8: If Rt_0 is T and $flag(Rl_0) == 1$, either $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2)$, or $Rt_1 == F \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 3)$.

3.6 MR-Cons 6

MR-Cons 6: If the root of the constraint S_0 is *implies*, which can be formed of f_1 *implies* f_2, we transform it to f_1 *or* f_2 to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 8 cases:

Case 1: If Rt_0 is F and $flag(Rl_0) == 1$, Rt_1 should be T and $flag(Rl_1) == 1$.

Case 2: If Rt_0 is F and $flag(Rl_0) == 2$, Rt_1 should be T and $flag(Rl_1) == 2$.

Case 3: If Rt_0 is F and $flag(Rl_0) == 3$, Rt_1 should be T and $flag(Rl_1) == 1$.

Case 4: If Rt_0 is F and $flag(Rl_0) == 5$, Rt_1 should be T and $flag(Rl_1) == 2$.

Case 5: If Rt_0 is T and $flag(Rl_0) == 4$, Rt_1 should be T and $flag(Rl_1) == 3$.

Case 6: If Rt_0 is T and $flag(Rl_0) == 3$, Rt_1 should be T and $flag(Rl_1) == 3 \vee flag(Rl_1) == 4$.

Case 7: If Rt_0 is T and $flag(Rl_0) == 2$, either $Rt_1 == T \wedge flag(Rl_1) == 1$, or $Rt_1 == F \wedge (flag(Rl_1) == 2 \vee flag(Rl_1) == 5)$.

Case 8: If Rt_0 is T and $flag(Rl_0) == 1$, either $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2)$, or $Rt_1 == F \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 3)$.

3.7 MR-Cons 7

MR-Cons 7: If the root of the constraint S_0 is *and*, which can be formed of f_1 *and* f_2, we transform it to f_1 *implies* f_2 to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 8 cases:

Case 1: If Rt_0 is T and $flag(Rl_0) == 1$, Rt_1 should be T and $flag(Rl_1) == 1$.

Case 2: If Rt_0 is T and $flag(Rl_0) == 2$, Rt_1 should be T and $flag(Rl_1) == 1$.

Case 3: If Rt_0 is T and $flag(Rl_0) == 3$, Rt_1 should be T and $flag(Rl_1) == 3$.

Case 4: If Rt_0 is T and $flag(Rl_0) == 5$, Rt_1 should be T and $flag(Rl_1) == 3$.

Case 5: If Rt_0 is F and $flag(Rl_0) == 4$, Rt_1 should be T and $flag(Rl_1) == 2$.

Case 6: If Rt_0 is F and $flag(Rl_0) == 2$, Rt_1 should be T and $flag(Rl_1) == 2 \vee flag(Rl_1) == 4$.

Case 7: If Rt_0 is F and $flag(Rl_0) == 3$, either $Rt_1 == T \wedge flag(Rl_1) == 1$, or $Rt_1 == F \wedge (flag(Rl_1) == 3 \vee flag(Rl_1) == 5)$.

Case 8: If Rt_0 is F and $flag(Rl_0) == 1$, either $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 3)$, or $Rt_1 == F \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2)$.

3.8 MR-Cons 8

MR-Cons 8: If the root of the constraint S_0 is *implies*, which can be formed of f_1 *implies* f_2, we transform it to f_1 *and* f_2 to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 8 cases:

Case 1: If Rt_0 is F and $flag(Rl_0) == 1$, Rt_1 should be F and $flag(Rl_1) == 1$.

Case 2: If Rt_0 is F and $flag(Rl_0) == 2$, Rt_1 should be F and $flag(Rl_1) == 1$.

Case 3: If Rt_0 is F and $flag(Rl_0) == 3$, Rt_1 should be F and $flag(Rl_1) == 3$.

Case 4: If Rt_0 is F and $flag(Rl_0) == 5$, Rt_1 should be F and $flag(Rl_1) == 3$.

Case 5: If Rt_0 is T and $flag(Rl_0) == 4$, Rt_1 should be F and $flag(Rl_1) == 2$.

Case 6: If Rt_0 is T and $flag(Rl_0) == 2$, Rt_1 should be F and $flag(Rl_1) == 2 \vee flag(Rl_1) == 4$.

Case 7: If Rt_0 is T and $flag(Rl_0) == 3$, either $Rt_1 == F \wedge flag(Rl_1) == 1$, or $Rt_1 == T \wedge (flag(Rl_1) == 3 \vee flag(Rl_1) == 5)$.

Case 8: If Rt_0 is T and $flag(Rl_0) == 1$, either $Rt_1 == F \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 3)$, or $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2)$.

3.9 MR-Cons 9

MR-Cons 9: If the root of the constraint S_0 is *implies*, which can be formed of f_1 *implies* f_2, we transform it to f_2 *implies* f_1 to obtain S_1. Then, in the follow-up execution, Rt_1 and Rl_1 should satisfy different conditions according to the following 8 cases:

Case 1: If Rt_0 is F and $flag(Rl_0) == 1$, Rt_1 should be T and $flag(Rl_1) == 1$.

Case 2: If Rt_0 is F and $flag(Rl_0) == 2$, Rt_1 should be T and $flag(Rl_1) == 2$.

Case 3: If Rt_0 is F and $flag(Rl_0) == 3$, Rt_1 should be T and $flag(Rl_1) == 3$.

Case 4: If Rt_0 is F and $flag(Rl_0) == 5$, Rt_1 should be T and $flag(Rl_1) == 4$.

Case 5: If Rt_0 is T and $flag(Rl_0) == 4$, Rt_1 should be F and $flag(Rl_1) == 5$.

Case 6: If Rt_0 is T and $flag(Rl_0) == 2$, either $Rt_1 == F \wedge flag(Rl_1) == 2$, or $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 3)$.

Case 7: If Rt_0 is T and $flag(Rl_0) == 3$, either $Rt_1 == F \wedge flag(Rl_1) == 3$, or $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2)$.

Case 8: If Rt_0 is T and $flag(Rl_0) == 1$, either $Rt_1 == F \wedge flag(Rl_1) == 1$, or $Rt_1 == T \wedge (flag(Rl_1) == 1 \vee flag(Rl_1) == 2 \vee flag(Rl_1) == 3)$.

4 MR-All

There are two different MRs in this part, and they transform the data under checking and the constraint simultaneously.

4.1 Positive quantifier and negative quantifier

Let's start with an in-depth analysis of the quantifier(e.g. $\forall$, $\exists$, *and*, *or*) in the constraint. A constraint's quantifiers can be into two parts: *positive* and *negative*, according to their impact to the whole constraint. A positive quantifier supports the root's violation or satisfaction while a negative quantifier supports it in the opposite way. It means if a positive quantifier is F under a given data, the whole constraint tends to be F while if a positive quantifier is F, the whole constraint tend to be T. In brief, a positive quantifier with T or a negative quantifier with F contributes to the whole constraint taking the truth value of T; a positive quantifier with F or a negative quantifier with T contributes to the whole constraint taking the truth value of F.

Let us consider the example in the Fig. 2. In the figure, quantifiers colored with green stands for positive quantifiers and quantifiers colored with red stands for negative quantifiers. Quantifier *not* in constraint S is positive, which means if it is T, it tends to make S be T while if it is F, it tends to make S be F.

In definitions, only the quantifier that is transitively through *implies* from the left or *not* from the root can reverse its supporting direction initialized to be positive, which has been clearly illustrated in Fig. 2.

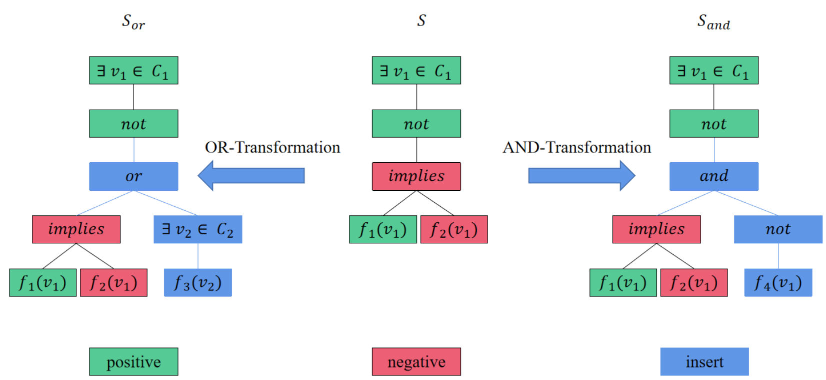

Figure 2: An example of constraint transformation

4.2 Constraint transformation

Since we want to transform both the data part and constraint part at the same time. To control the transformation impact, we control to insert a new formula which naturally requires new data being added into the data part. We propose two kinds of transformations: *AND-Transformation* and *OR-Transformation*. AND-Transformation means linking a quantifier with a newly added *and*, which links to a newly added constraint as another branch. Similarly, OR-Transformation means linking quantifier with a newly added OR. Note that the newly added constraint can be arbitrary. Fig 2 has clearly shown two kinds of transformations.

4.3 MR-All 1

Now, we can elaborate our MR-All 1 using concepts above as follow:

MR-All 1: If $Rt_0 == F$, we apply AND-Transformation to any positive quantifier or apply OR-Transformation to any negative quantifier in S_0 to obtain D_1 and S_1. Then, in the follow-up execution, Rt_1 should still be F and Rl_1 should satisfy the condition: $Rl_0 \preceq Rl_1$.

4.4 MR-All 2

MR-All 2 is the other side of MR-All 1:

MR-All 2: If $Rt_0 == T$, we apply OR-Transformation to any positive quantifier or apply AND-Transformation to any negative quantifier in S_0 to obtain D_1 and S_1. Then, in the follow-up execution, Rt_1 should still be T and Rl_1 should satisfy the condition: $Rl_0 \preceq Rl_1$.

5 MR-CCT

If more internal states can be observed, e.g., CCT status like substantial parts accessible in existing work[3], we can additional design MRs with respect to measure the impact for such detailed CCT statuses under different transformations between source and follow-up executions.

5.1 CCT and SCCT

Normally, a constraint for checking can be expanded to a *Consistency Computation Tree* (CCT) based on both its static constraint structure and the current data under checking. SCCT [3] is substantial parts in CCT, which is the minimum set of nodes contributing to the link generation. Fig. 3 has shown an example of CCT and SCCT . In the figure, the nodes colored with purple are SCCT nodes and the links of this constraint is entirely up to them. By comparing the SCCT in the source output and the follow-up output through different transformations, we can desgin the MR-CCT.

In the next section, we will use $SCCT_0$ and $SCCT_1$ to denote the set of SCCT nodes in the source output and follow-up output seperately.

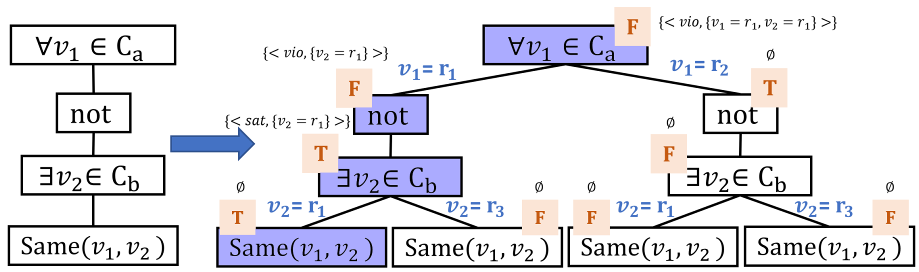

Figure 3: An example of CCT and SCCT

5.2 MR-CCT 1

MR-CCT 1 uses the transformation similar to MR-Data 1 and can be elaborated as follow:

MR-CCT1: If $Rt_0 == F$, we apply any change with an inconsistency incurring impact on S_0 to D_0 to obtain D_1. Then, in the follow-up execution, $SCCT_1$ should contain $SCCT_0$, i.e. $SCCT_0 \subseteq SCCT_1$.

5.3 MR-CCT 2

Similarly, if we use the transformation mentioned in MR-All 1, then we can derive the MR-CCT 2:

MR-CCT2: If $Rt_0 == F$, we apply AND-Transformation to any positive quantifier or apply OR-Transformation to any negative quantifier in S_0 to obtain D_1 and S_1. Then, in the follow-up execution, $SCCT_1$ should contain $SCCT_0$, i.e. $SCCT_0 \subseteq SCCT_1$.

References

[1] C. Xu, S. C. Cheung, W. K. Chan, and C. Ye, "Partial constraint checking for context consistency in pervasive computing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 19, no. 3, pp. 1–61, 2010.

[2] H. Wang, C. Xu, B. Guo, X. Ma, and J. Lu, "Generic adaptive scheduling for efficient context inconsistency detection," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 464–497, 2019.

[3] C. Chen, H. Wang, L. Zhang, C. Xu, and P. Yu, "Minimizing link generation in constraint checking for context inconsistency detection," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2022, pp. 13–24.