An Analysis-Revision Cycle to Evolve Requirements Specifications by Using the SCTL-MUS Methodology*

Jorge García-Duque  Josée J. Pazos-Arias  Belén Barragán-Martínez
Telematic Engineering Department
University of Vigo. Spain
{jgd, jose, belen}@det.uvigo.es

Abstract

The development of requirements specifications can be supported by a cycle composed of two phases: analysis and revision. In this paper, we use the SCTL-MUS methodology to bridge the gap between these two phases. Analysis phase provides diagnostic information if a desirable property is not satisfied by the current system specification. A crucial aspect of the analysis-revision cycle is how to use the diagnostic information provided to generate alternative system refinements which can be included in the system specification to satisfy the property in question (revision phase). Our approach allows translating the diagnostic information into system requirements refinements closed to the system domain. It facilitates to the stakeholders the decision of what system requirements refinements must be included in the system requirements specification.

1. Introduction

In [13] is argued that models for reasoning about current alternatives and future changes should be at the heart of requirements engineering. In [5], authors propose a cycle composed of two phases (analysis and revision) to support the development of requirements specifications of state transition systems. The tasks of the analysis phase are: checking whether a desirable set of properties of a system is satisfied by its current (partial) specification; and providing appropriate diagnostic information when a property is violated by the specification. Revision phase uses the diagnostic information obtained in the analysis phase to obtain a new (partial) system specification which no longer violates the system’s property in question.

The framework proposed in [5] uses: abductive reasoning [9] during analysis to detect whether a system description \( D \) satisfies a property \( P \) (\( D \models P \)), generating diagnostic information (\( \Delta \)) if the property is not satisfied; and inductive learning during revision to change the description \( D \) into a new description \( D' \) (if \( D \) violates \( P \)) by using the diagnostic information (\( \Delta \)) to derive a number of training examples (\( \Delta' \)) for inductive learning [6]. The problem of finding whether \( D \models P \) is translated into the equivalent problem of showing that it is not possible to find a set (\( \Delta \)) of state transitions consistent with \( D \) and that, together with \( D \), proves the negation of \( P \) (\( D \cup \Delta \models \neg P \)). If \( \Delta \) (wrong state transitions) is found then \( \Delta \) acts as a counter-example to the validity of \( P \).

In this paper, we adopt the analysis-revision cycle proposed above by using a similar method to generate appropriate system behaviors (\( \Delta' \)) from the diagnostic information (\( \Delta \)) since \( \Delta' \) are generated by changing some elements of the counter-example \( \Delta \), which we obtain by using multi-valued model checking [3, 4]. Nevertheless, we take advantage of using the SCTL-MUS methodology [12, 7] in the analysis and revision phases. SCTL-MUS methodology proposes an incremental development model [10] which formalizes system evolutions by defining unspecified elements which can evolve into specified ones. Since SCTL-MUS methodology defines unspecified elements, it is possible to distinguish not-specified elements (partial specification) from those which are specified as false. Properties which involve unspecified elements at the current (partial) specification may be satisfied or not by the system depending on how these unspecified elements evolve in future system refinements. Our approach provides an adequate framework to facilitate the evolution of requirements specifications when this kind of properties are checked:

1. We can distinguish not-specified (unspecified) elements at the current specification, which are considered as false in the classical models with two values (true and false) of specification, from elements specified as false. It allows defining a multi-valued logic [2] with a new degree of satisfaction (called unspecified

---

* This work was partially supported by the Xunta de Galicia Basic Research Project PGIDT01PXI32203PR.
and denoted by $\frac{1}{2}$ of a property $\mathcal{P}$ with the following meaning: current (partial) system specification does not satisfy $\mathcal{P}$ and it also does not violate that property. Depending on how the system specification evolves, the property $\mathcal{P}$ will be or not satisfied [8]. From this reasoning, we can obtain diagnostic information ($\Delta$) including potential wrong state transitions and a set of evolutions which make that $\mathcal{P}$ is violated. Since a property $\mathcal{P}$ can involve both specified and unspecified elements in the current system specification, the SCTL (Simple Causal Temporal Logic) logic also includes two new degrees of satisfaction with the following meaning: $\frac{1}{2}$, property is partially unspecified at the current specification and it cannot become true; $\frac{3}{2}$, property is partially unspecified at the current specification and it cannot become false.

2. We can obtain alternative system refinements ($\Delta'$) from $\Delta$ by only changing the set of evolutions which make that $\mathcal{P}$ is violated. It allows generating a new specification $\mathcal{D}'$ consistent with $\mathcal{D}$ since $\mathcal{P}$ is not satisfied by $\mathcal{D}$ because $\mathcal{P}$ involves unspecified elements in $\mathcal{D}$.

3. Alternative system refinements ($\Delta'$) obtained from the diagnostic information ($\Delta$) can be translated into system requirements refinements ($\mathcal{R}'$), closing the diagnostic information to the system domain, bridging the gap between analysis and revision.

The paper is organized as follows: Section 2 explains the reasoning method used during analysis to obtain the diagnostic information which is used to generate counterexamples. Section 3 shows how to use the diagnostic information obtained in the analysis phase to generate alternative system refinements. Section 4 describes the revision phase, including a procedure to translate the alternative system refinements into system requirements refinements closed to the system domain. Finally, in section 5 we draw some conclusions and directions for future work.

2. Generating Counter-examples

To illustrate, we provide a simple example, which is adopted from [5]. It also allows showing the advantages of the proposed methodology in relation with the framework proposed in [5]. Consider an electric circuit consisting of a single light bulb and two switches (A and B), all connected in series. The system’s description contains rules such as: if it is not the case that switch A is on at a current state, flicking switch A causes the light to come on at the next state, provided that the light is not already on. In our approach, that information is represented by using SCTL requirements [12]. SCTL has a causal semantics [11]: "If...

...(Premise)... is possible, then Simultaneously ($\Rightarrow$) / Previously ($\Box$) / Next ($\triangledown$) ... (Consequence)... must be possible”. That is the reason why we have defined a new truth value called contradictory or not-applicable ($\frac{1}{2}$) which is assigned to properties whose premise is not satisfied.

Therefore, $\Phi \triangleq \{0, \frac{1}{2}, \frac{3}{2}, 1\}$ is the ordered set of the degrees of satisfaction defined in the SCTL-MUS methodology. Let $a, b$ be two elements of $\Phi$, the sum ($\lor$) and product ($\land$) operations are defined as follows: $(a \lor b) = \text{Max}(a, b); (a \land b) = \text{Min}(a, b)$.

**Definition 1** A unary operation called complementation, denoted by $\overline{\bullet}$ or $\neg$, is defined on $\Phi$ by the table:

<table>
<thead>
<tr>
<th>$a$</th>
<th>$\overline{a}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\frac{1}{2}$</td>
<td>$\frac{3}{2}$</td>
</tr>
<tr>
<td>$\frac{3}{2}$</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Returning to the example, assume that a (possibly incorrect) description $\mathcal{D}$ of our electric circuit includes the following SCTL requirements:

$\mathcal{R}_1 \equiv \neg \text{Switch}_A \land \neg \text{Light}_\text{On} \land \text{Flick}_A \Rightarrow \Box \text{Light}_\text{On}$

$\mathcal{R}_2 \equiv \neg \text{Switch}_B \land \neg \text{Light}_\text{On} \land \text{Flick}_B \Rightarrow \Box \text{Light}_\text{On}$

$\mathcal{R}_3 \equiv \neg \text{Switch}_A \land \text{Flick}_A \Rightarrow \Box \text{Switch}_A \land \text{On}$

$\mathcal{R}_4 \equiv \neg \text{Switch}_B \land \text{Flick}_B \Rightarrow \Box \text{Switch}_B \land \text{On}$

$\mathcal{R}_5 \equiv \text{Switch}_A \land \Rightarrow \Box (\text{Switch}_A \land \lor \text{Flick}_A)$

$\mathcal{R}_6 \equiv \text{Switch}_B \land \Rightarrow \Box (\text{Switch}_B \land \lor \text{Flick}_B)$

In order to simplify the notation, atomic requirements (which have the following form: $\text{true} \Rightarrow \bigoplus [-] a_i$) are denoted by $\bigoplus [-] a_i$, where $\bigoplus$ is a temporal operator from $\{\text{Simultaneously, Previously ($\Box$), Next ($\triangledown$)}\}$, and $a_i$ is an action (event) of the specified system.

From the specification of these six SCTL requirements, we can synthesize a MUS (Model of Unspecified States) graph representing the specified behavior. It is made by using the synthesis algorithm developed in the SCTL-MUS methodology. The elements (states, actions or labels and arcs) of a MUS graph can be specified as: true (1); false (0); and unspecified ($\frac{1}{2}$) which can evolve into 1 or 0. MUS graphs do not need that the initial state of the system is specified. The initial state and the other unspecified parts of the
system will must be specified in the last system refinement, before system is implemented.

Figure 1 shows the MUS graph \( \mathcal{M}_D \) of the system whose description \( D \) is given by the SCTL requirements \( \{ R_1, R_2, R_3, R_4, R_5, R_6 \} \). It has two similar parts which are not joined at the current development stage (partial specification). First part (see figure 1(a)) shows the behavior specified by the SCTL requirements \( R_1, R_3, R_4 \) and \( R_6 \): “From a state \( (E_1) \) in which actions SwitchA\_On and Light\_On are false and the action Flick\_A is true, system can evolve (if the action Flick\_A occurs in that state \( (E_1) \)) into a new state \( (E_2) \) in which actions SwitchA\_On and Light\_On are true”. The states in which system evolves if those actions occur in the state \( E_2 \) are unspecified. MUS graphs define a fixed unspecified state \( (E_{\text{unsp}}) \) to represent those unspecified states. The remainder of actions are also unspecified in the states \( E_1 \) and \( E_2 \) (unspecified actions are omitted from figures to simplify them).

(a) \hspace{2cm} (b)

![MUS Graph](image)

**Figure 1. MUS graph \( \mathcal{M}_D \).**

In a similar way, second part (see figure 1(b)) shows the behavior specified by the SCTL requirements \( R_2, R_4 \) and \( R_6 \): “From a state \( (E_3) \) in which actions SwitchB\_On and Light\_On are false and the action Flick\_B is true, system can evolve (if the action Flick\_B occurs in that state \( (E_3) \)) into a new state \( (E_4) \) in which actions SwitchB\_On and Light\_On are true”. The states in which system evolves if those actions occur in that state \( E_4 \) are unspecified. The remainder of actions are also unspecified in the states \( E_3 \) and \( E_4 \).

A system property that we would like the above description \( D \) to satisfy could be the following SCTL requirement \( \mathcal{P} \):

\[ \mathcal{P} \equiv \text{Light\_On} \Rightarrow \text{SwitchA\_On} \land \text{SwitchB\_On} \]

To find a set \( \Delta \) of state transitions such that \( D \cup \Delta = \mathcal{P} \) we first find the states of the MUS graph \( \mathcal{M}_D \) in which \( \mathcal{P} \) is applicable. These states are obtained by verifying the premise of \( \mathcal{P} \) in each state of \( \mathcal{M}_D \). Only the states in which the degree of satisfaction of the premise of \( \mathcal{P} \) is greater than \( \frac{1}{2} \) are considered states of applicability for the property \( \mathcal{P} \). Since according to the semantics defined by SCTL, requirements whose premise cannot be satisfied (0, \( \frac{1}{2} \) or \( \frac{3}{2} \)) have no sense.

The multi-valued model-checker developed in the SCTL-MUS methodology uses the operation called causal (denoted by \( \rightarrow \)) to obtain (recursively) the degree of satisfaction of a SCTL requirement from the degree of satisfaction of its premise and consequence (see definition 2). The operation called atomic satisfaction (denoted by \( \models \)) is used to obtain the degree of satisfaction of an action (see definition 3).

**Definition 2** Let \( a, b \) be two elements of \( \Phi \), an internal operation called causal, \( a \rightarrow b \), is defined as follows:

\[
\begin{array}{cccccc}
0 & \frac{1}{2} & 1 & \frac{3}{4} & \frac{1}{2} & 0 \\
0 & \frac{1}{2} & 1 & \frac{3}{4} & \frac{1}{2} & 0 \\
\frac{1}{2} & \frac{3}{4} & \frac{1}{2} & 1 & \frac{3}{4} & \frac{1}{2} \\
1 & 0 & \frac{1}{2} & \frac{3}{4} & \frac{1}{2} & 1 \\
\end{array}
\]

First operand (column) is the degree of satisfaction of the **Premise**, and second operand (row) is the degree of satisfaction of the **Consequence**.

**Definition 3** Let \( a, b \) be two elements of \( \Psi \equiv \{ 0, \frac{1}{2}, 1 \} \), an internal operation called atomic-satisfaction, \( \models (a, b) \), is defined as follows:

\[
\begin{array}{cccc}
\models & 0 & \frac{1}{2} & 1 \\
0 & 1 & \frac{1}{2} & 0 \\
\frac{1}{2} & 1 & 1 & 1 \\
1 & 0 & \frac{1}{2} & 1 \\
\end{array}
\]

First operand (column) is the **action** specification according to the specified property, and second operand (row) is the **action** specification according to the current system MUS graph.

**Premise** of property \( \mathcal{P} \) (which is the atomic requirement \( \mathcal{P}_P \)) is true \( \Rightarrow \) Light\_On, denoted by Light\_On, is not satisfied in the states \( E_1 \) and \( E_2 \) because its premise is true and its consequence (the action Light\_On) is specified as false in these states. However, it is satisfied in the states \( E_2 \) and \( E_4 \) because the action Light\_On is specified as true in these states. Therefore, \( \mathcal{E}_{ap} = \{ E_2, E_4 \} \).
Once we have the states of applicability of the property \( \mathcal{P} \), the procedure to find \( \Delta \) starts by negating the property \( \mathcal{P} \) to obtain \( \neg \mathcal{P} \):

\[
\neg \mathcal{P} \equiv \neg (\text{Light}_\text{On} \Rightarrow (\text{SwitchA}_\text{On} \land \text{SwitchB}_\text{On}))
\]

**Theorem 1** Let \( \mathcal{R}_1 = \neg \mathcal{P} \otimes \mathcal{C} \) and \( \mathcal{R}_2 = \mathcal{P} \otimes \mathcal{C} \) be two SCTL requirements, and let \( E_j \) a state of a MUS graph \( \mathcal{M} \) such that \( \models (\mathcal{P}, E_j) \implies \frac{1}{2} \) \( \models (\mathcal{P}, E_j) \in \{\frac{1}{2}, 1\} \), then:

\[
\models (\mathcal{R}_1, E_j) = \models (\mathcal{R}_2, E_j)
\]

**Proof.** Trivial from definitions 1 and 2.

Since we will verify \( \mathcal{P} \) in the states in which the degree of satisfaction of its premise is greater than \( \frac{1}{2} \), we can apply theorem 1 to property \( \neg \mathcal{P} \) as follows:

\[
\neg \mathcal{P} \equiv \text{Light}_\text{On} \Rightarrow (\neg \text{SwitchA}_\text{On} \lor \neg \text{SwitchB}_\text{On})
\]

\( \neg \mathcal{P} \) yields two parts \( (\neg \mathcal{P}_1 \land \neg \mathcal{P}_2) \) such that:

\[
\models (\neg \mathcal{P}_1, E) = \models (\neg \mathcal{P}_1, E) \lor \exists \models (\neg \mathcal{P}_2, E), \text{where}
\]

\[
\neg \mathcal{P}_1 \equiv \text{Light}_\text{On} \Rightarrow \neg \text{SwitchA}_\text{On}, \quad \neg \mathcal{P}_2 \equiv \text{Light}_\text{On} \Rightarrow \neg \text{SwitchB}_\text{On} \text{ and } E_i \in \mathcal{E}_{apr}. \text{ It is trivially proved from definition 2 and the operations sum, product and complementation defined by the SCTL logics.}
\]

Therefore, to find \( \Delta \) such that \( \mathcal{D} \cup \Delta \models \neg \mathcal{P} \), we can firstly trying to find \( \Delta \) such that \( \mathcal{D} \cup \Delta \models \neg \mathcal{P}_1 \). To achieve it, we check the satisfaction of \( \neg \mathcal{P}_1 \) in all \( E_i \in \mathcal{E}_{apr} \). Equations 1 and 2 describe the steps made by the model-checker to obtain the degree of satisfaction of property \( \neg \mathcal{P}_1 \) in the states \( E_2 \) and \( E_4 \), respectively (see definitions 2 and 3):

\[
\models (\neg \mathcal{P}_1, E_2) = \models (\text{Light}_\text{On}, E_2) \implies (1, 1) \rightarrow (0, 1)
\]

\[
= 1 \rightarrow 0
\]

\[
= 0
\]

\[
\models (\neg \mathcal{P}_1, E_4) = \models (\text{Light}_\text{On}, E_4) \implies (\neg \text{SwitchA}_\text{On}, E_4)
\]

\[
= (1, 1) \rightarrow (0, \frac{1}{2})
\]

\[
= 1 \rightarrow \frac{1}{2}
\]

\[
= \frac{1}{2}
\]

Therefore, \( \neg \mathcal{P}_1 \) is not satisfied in \( E_2 \), being not possible finding a counter-example \( \Delta \) to the validity of \( \mathcal{P} \). Nevertheless, \( \neg \mathcal{P}_1 \) is unspecified in \( E_4 \). This means that system can evolve (in future refinements of the requirements specifications), without violating the current system description \( \mathcal{D} \), making that property \( \neg \mathcal{P}_1 \) be satisfied in \( E_4 \). It occurs (see equation 2) if \( E_4[\text{SwitchA}_\text{On}] \sim 0 \) \( (a \sim b \text{ means that } a \text{ evolves into } b) \). From this reasoning, we can obtain a counter-example \( \Delta \) to the validity of \( \mathcal{P} \) which can be used as diagnostic information.

In our approach, the counter-example does not only consist of a set of state transitions, but it also includes a set of evolutions (losing unspecification) of the specification of unspecified actions, referred to as actions refinements (an action refinement is the specification as true or false of an unspecified action):

\[
\Delta_1 = \{E_3 \xrightarrow{\text{FlickA}} E_4, \ E_4[\text{SwitchA}_\text{On}] \sim 0\}
\]

The specification of each action \( a_i \) in every state \( E_j \) is easily obtained from the MUS graph \( \mathcal{M}_D \). Observing figure 1, we can conclude that the states \( E_3 \) and \( E_4 \) contain the following specification of the actions:

\[
E_3[] = \{[\text{SwitchA}_\text{On}] = \frac{1}{2}, [\text{SwitchB}_\text{On}] = 0, [\text{FlickA}] = \frac{1}{2}, [\text{FlickB}] = 1, \}
\]

\[
E_4[] = \{[\text{SwitchA}_\text{On}] = 0, [\text{SwitchB}_\text{On}] = \frac{1}{2}, [\text{FlickA}] = \frac{1}{2}, [\text{FlickB}] = \frac{1}{2}, \}
\]

3 Generating Alternative System Refinements

A crucial aspect of the analysis-revision cycle is how to use the diagnostic information provided (\( \Delta \)) to generate al-
ternative system refinements ($\Delta'$) which can be included in the system description $D$ to obtain a new system description $D'$ that guarantees that $\Delta$ is no longer an explanation for the violation of the property $P$.

The counter-example obtained in previous section ($\Delta_1$) includes a state transition ($E_3 \xrightarrow{\text{Flick.B}} E_4$) and an action refinement ($E_4[\text{Switch.A.On}] \leadsto 0$). This is because the degree of satisfaction obtained of $\neg P_1$ in $E_3$ was $\frac{1}{2}$. This means that current (partial) system description $D$ does not satisfy $\neg P_1$ and it also does not violate that property, but system description $D$ can be refined into a new system description $D'$ which satisfies or violates $\neg P_1$ (depending on how $D$ is refined), being $D'$ consistent with $D$.

We have based on previous reasoning to obtain $\Delta_1$. We can obtain from this diagnostic information an alternative system refinement $\Delta'_1$ by changing the action refinement of $\Delta_1$ since this action refinement makes that $\neg P_1$ is ($E_4[\text{Switch.A.On}] \leadsto 0$) or not ($E_4[\text{Switch.A.On}] \leadsto 1$) satisfied. Therefore $\Delta'_1 = \{E_3[\text{Switch.A.On}] \leadsto 1\}$ is an alternative system refinement which makes that $\Delta_1$ is no longer an explanation for the violation of the property $P$. Nevertheless, this action refinement implies making another action refinement ($E_3[\text{Switch.A.On}] \leadsto 1$) to be consistent with $\mathcal{R}_5$ in that state: $E_4(\mathcal{R}_5 \equiv \text{Switch.A.On} \Rightarrow \emptyset (\text{Switch.A.On} \lor \text{Flick.A}))$. Therefore, the alternative system refinement $\Delta'_1$ consists of two actions refinements: $\Delta'_2$ is obtained in a similar way.

$$\Delta'_1 = \{E_3[\text{Switch.A.On}] \leadsto 1, E_4[\text{Switch.A.On}] \leadsto 1\}$$
$$\Delta'_2 = \{E_3[\text{Switch.B.On}] \leadsto 1, E_2[\text{Switch.B.On}] \leadsto 1\}$$

4 Refining Requirements Specifications

Since the alternative system refinements ($\Delta'_1$ and $\Delta'_2$) have been obtained by changing actions refinements into the counter-examples ($\Delta_1$ and $\Delta_2$), these alternative system refinements are consistent (they are not in contradiction) with the system description $D$. Therefore, we can obtain a new system description $D'$ including the previous description $D$ and the provided alternative system refinements. This is possible because the system MUS graph distinguishes actions which are specified in the current (partial) system description from those which are not specified (the unspecified actions), in comparison with the classical models with only two specification values (true or false) where not-specified actions are considered as false. Nevertheless, the form in which these alternative system refinements are expressed (by indicating actions refinements which are consistent with $D$) does not provide enough information to decide if they can be adopted by the stakeholders. Our approach allows closing these kind of alternative system refinements to the system domain, facilitating to the stakeholders the decision of accepting the proposed alternative system refinements.

To achieve it, we translate the actions refinements of the alternative system refinements ($\Delta'_1$ and $\Delta'_2$) into system requirements refinements ($R'_i$), specializing the SCTL requirements which specify the system description $D$. This is possible because, as we have explained above, the alternative system refinements are consistent (they are not in contradiction) with the current (partial) system description $D$.

This translation process can obtain different possible system requirements refinements. Stakeholders will be required to decide what of them are accepted. The main advantage of our approach here is that the diagnostic information ($\Delta$) is used to provide system refinement information ($R'$) closed to the system domain.

Next, we outline how to obtain the system requirements refinements from the alternative system refinements $\Delta'_1$ and $\Delta'_2$. In general, given a set of SCTL requirements $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ specifying a system description $D$, the SCTL-MUS methodology provides an algorithm which synthesizes a MUS graph $\mathcal{M}$ representing the specified behavior. Every state $E_j$ of the MUS graph $\mathcal{M}$ has a link to the SCTL requirements $\{(\mathcal{R}_{E_j})\}$ which are synthesized in that state. Remark that, each of these requirements can be a specified SCTL requirement $\mathcal{R}_k$, or any SCTL requirement which is included in the specified ones. In particular, if a SCTL requirement $\mathcal{R}_k$ is synthesized in a state $E_j$ of $\mathcal{M}$, then the premise of $\mathcal{R}_k$ (another SCTL requirement denoted by $P_{\mathcal{R}_k}$) is also synthesized in $E_j$, and its consequence (another SCTL requirement denoted by $C_{\mathcal{R}_k}$) is synthesized in the states of applicability according to the temporal operator of $\mathcal{R}_k$.

Let $\Delta = \{E_j[\alpha_i] \leadsto 1(0)\}$ be an alternative system refinement obtained according to the method proposed in section 3. It can be translated into a system requirement refinement ($\mathcal{R}'_i$) as follows: if $\mathcal{R}_k$ is synthesized in $E_j$ then the specification of the action $\alpha_i$ (if it is unspecified in $P_{\mathcal{R}_k}$) can be added as true (false) to the premise of $\mathcal{R}_k$, obtaining a new system requirement refinement $\mathcal{R}'_k$; if the consequence of $\mathcal{R}_k$ is synthesized in $E_j$ then the specification of the action $\alpha_i$ (if it is unspecified in $C_{\mathcal{R}_k}$) can be added as true (false) to the consequence of $\mathcal{R}_k$, obtaining a new system requirement refinement $\mathcal{R}'_k$.

Returning to the electrical circuit example, table 1 shows the requirements synthesized in each state of the MUS graph $\mathcal{M}_D$.

Using the method described above, we can translate the alternative system refinement $\Delta'_1 = \{E_3[\text{Switch.A.On}] \leadsto 1, E_4[\text{Switch.A.On}] \leadsto 1\}$ into the following system requirement refinement $R'_1$:

$$\mathcal{R}'_1 \equiv \text{SwitchA.On} \land \neg \text{SwitchB.On} \land \neg \text{Light.On} \land \text{Flick.B} \Rightarrow \emptyset (\text{SwitchA.On} \land \text{Light.On})$$
Table 1. Requirements synthesized in each state of \( M_D \).

<table>
<thead>
<tr>
<th>State</th>
<th>Synthesized Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_1 )</td>
<td>( { R_1, P_{R_1}, R_3, P_{R_3}, C_{R_3} } )</td>
</tr>
<tr>
<td>( E_2 )</td>
<td>( { C_{R_1}, R_3, R_5, P_{R_5} } )</td>
</tr>
<tr>
<td>( E_3 )</td>
<td>( { R_2, P_{R_2}, R_4, P_{R_4}, C_{R_4} } )</td>
</tr>
<tr>
<td>( E_4 )</td>
<td>( { C_{R_2}, C_{R_3}, R_6, P_{R_6} } )</td>
</tr>
</tbody>
</table>

\( R_2 \) is synthesized in the state \( E_3 \) of the MUS graph \( M_D \). Therefore, its premise \( P_{R_2} \) is also synthesized in \( E_3 \). The action \( SwitchA\_On \) is unspecified in \( P_{R_2} \), so the first action refinement of \( \Delta_1' \) (\( E_3[\text{SwitchA\_On}] \sim 1 \)) can be included in the premise of \( R_2 \). The second action refinement of \( \Delta_1' \) (\( E_4[\text{SwitchA\_On}] \sim 1 \)) can be also included in the consequence of \( R_2 \), because it (\( C_{R_2} \)) is synthesized in \( E_4 \), and the action \( SwitchA\_On \) is unspecified in \( C_{R_2} \).

Nevertheless, we can obtain other system requirements refinements from \( \Delta_1' \), \( R_4 \) is synthesized in \( E_3 \), its consequence is synthesized in \( E_4 \), and the action \( SwitchA\_On \) is unspecified in its premise and in its consequence. Therefore, other system requirement refinement extracted from \( \Delta_1' \) is the following:

\[ R_4' \equiv \text{SwitchA\_On} \land \neg \text{SwitchB\_On} \land \text{Flick\_B} \Rightarrow \Box (\text{SwitchA\_On} \land \text{SwitchB\_On}) \]

In a similar way, we can obtain other two system requirement refinements \( R_4' \) and \( R_5' \) from \( \Delta_2 \). These system requirements refinements are closed to system domain, and they are good alternative to evolve the requirements specifications.

Stakeholders can learn from the provided system requirements refinements. In the electric circuit example, the error was caused because the SCTL requirement \( R_2 (R_1) \) describing part of the system was not totally specified. Stakeholders can decide that \( R_2 (R_1) \) must be specialized into \( R_2' (R_1') \), and \( R_4' (R_5') \) must be discarded. This reasoning can be made since the diagnostic information is provided as refinement of the requirements which describe the system.

Moreover, the system requirements refinements \( R_2' \) and \( R_4' \) make that property \( P \) is satisfied in its states of applicability \( E_2 \) and \( E_4 \). Therefore, we can conclude that the new system description \( D' \), given by the SCTL requirements \( \{ R_1, R_2, R_3, R_4, R_5, R_6 \} \) is consistent with property \( P \).

5 Conclusions and Future Work

In this paper, we have used the SCTL-MUS methodology to facilitate the evolution of requirements specifications of state transition systems. We have also compared our approach with the proposed in [5], showing the advantages of using a third value of specification, which allows distinguishing not-specified elements (partial specification) from those which are specified as false.

The analysis phase is carried out by using the multi-valued model-checker developed in the SCTL-MUS methodology. Our approach takes advantage of using a third value of specification when the property, that we would like the system specification to satisfy, involves actions (events) which are unspecified in the current system specification. In this case, the provided diagnostic information includes potential wrong state transitions and a set of actions refinements.

The revision phase is carried out in two steps. Firstly, we change the actions refinements of the provided diagnostic information, obtaining alternative system refinements. Secondly, we translate these alternative system refinements into system requirements refinements, which are closed to the system domain. It allows facilitating to the stakeholders the decision of what system requirements refinements must be included in the system requirements specification.

Nowadays, our work is focused on formalizing a method to select the possible system requirements refinements. The translation process, from alternative system refinements into system requirements refinements can obtain several different system requirements refinements, as we have shown in section 4. It is necessary to filter the system requirements refinements, so stakeholders can reason about an adequate number of possible refinements. One possibility is to select system requirements refinements which include a simple action refinement since they can be easily adopted by the stakeholders. Nevertheless, it is also interesting to select the system requirements refinements which translate the provided alternative system refinements (\( \Delta' \)) into the minimal specializations of the system requirements.

On the other hand, in order to support real time requirements, we are defining an extension of MUS (MUS-T [14]) which introduces time as a dense domain, since dense models are more expressive and suitable for composition and refinement [1]. It is also our goal to extend the SCTL logic (SCTL-T) in order to express time restrictions in a requirement.

References


