Finding Mode Invariants in SCR Specifications

Zhenyi Jin*
ISSE Department
George Mason University
Fairfax, VA 22030
email: zjin@isse.gmu.edu

Abstract

This paper introduces an algorithm and a new graph, the Conditioned Transition Graph (CTG), to derive the mode invariants from an Software Cost Reduction (SCR) mode transition table. An SCR requirements document contains a complete description of the external behavior of the software system. Some system properties, such as mode invariants, can be used to describe safety features that must be ensured during software development. Current SCR requirements documents give independently derived invariants. The algorithm detects mode invariants by first transforming an SCR mode transition table into a CTG, and then applying defined computations on two matrices derived from the CTG incidence matrix. The algorithm was used to generate mode invariants from an SCR mode transition table, which were correct as compared to externally derived invariants. An SCR requirements document implies some invariant properties of the specified system. This method can also be used to provide test requirements and test cases that are independent of the software design structure.

1 Introduction

A software requirements document completely describes the external behavior of a software system [Dav93]. It provides a way to communicate with the other stages of the software development, and defines the contents and boundary of the software system. Errors in this document are difficult and expensive to correct if they propagate to the later stages in the software development lifecycle. A requirements specification is a behavioral specification of the system's activities. Also, the requirements specification implicitly or explicitly includes

*Partially supported by the National Science Foundation under grant CCR-93-11967.

some properties that are invariant properties of the system. Some of these invariants are important safety features that must be ensured during the software development.

The Software Cost Reduction (SCR) requirements notation was developed by a research group at the Naval Research Laboratory as part of a general Software Cost Reduction project [Hen80]. A complete SCR requirements specification describes the behavioral, functional, precision, and timing requirements of the software system under development. The model is designed for analysis of behavioral requirements only. Introduced more than a decade ago to describe the functional requirements of software, the SCR specifications has been applied to practical systems such as the A-7 Operational Flight Program [HPSK78] [AFB+88].

System properties should be reflected in the structure of SCR specifications independently of specific applications. In this paper, we propose an algorithm to detect system properties and invariants from an SCR specification table. We introduce a conditioned transition graph and mode transition matrices, which are derived from the SCR specifications, and then find the system invariants by comparing the derived mode transition matrices. This new invariant detection method provides a mechanical procedure that can be easily automated. It views the SCR specifications from the SCR structure's point of view and analyzes mode invariants as structural properties. It presents a new way to analyze software requirements documents and detects implicit system properties that can be very useful in understanding, developing and testing the software systems.

The outline of this paper is as follows. Section 2 discusses related work. Section 3 gives a brief introduction to the SCR specifications construct and introduces invariant properties. Section 4 proposes an algorithm to find mode invariants in the SCR specifications. The method is applied to a cruise control problem in Section 5. Section 6 summarizes and discusses some related issues. Conclusions are presented in Section 7.

2 Related Work

SCR specification has been used to specify practical systems [HLK93]. Also, research has been done to analyze requirements consistencies and to verify given system properties. Atlee and Gannon [AG93] use model checking to analyze SCR specifications specifications. Model checking of temporal logic, a sound technique used to verify safety properties in hardware systems, is used to verify a given set of safety properties for event-driven systems. A model checker [Bro89] determines whether the invariants hold or not. The result shows discrepancies between the SCR specifications and the safety properties. Atlee [Atl94] extends this work by presenting a logic-model semantics for SCR mode specifications, which enables native model-checking of SCR specifications. In other related work, Heitmeyer and Labaw [HLK93] propose a consistency checker that tests all tables and definitions in an SCR specification. Mode invariant properties are not discussed in their paper.

3 SCR Specifications

In this section, we briefly describe the SCR specifications as used in Atlee and Gannon's paper [AG93] and discuss some system properties.

3.1 SCR Constructs

The SCR specifications were developed by a research group at NRL as part of a general Software Cost Reduction project [Hen80]. An SCR document specifies a software system's behavior as a finite set of concurrent, event-driven state-transition machines called *modeclasses*. Each modeclass is composed of a set of *modes* that represent the system's different modes of operation, and transitions among the modes. Each modeclass describes one aspect of the system's behavior, and the global behavior of the entire system is defined by the composition of the system's modeclasses. The system is in exactly one modeclass at a time. The system's environment is represented by a set of boolean environmental conditions. These definitions are from Atlee and Gannon's paper [AG93].

- **Environmental Conditions** are the inputs to the system. All conditions are boolean.
- **An event** is a change to a condition's value. Events are detectable at the point in time at which they occur.
- **A trigger event** is a change in the value of one condition. For example, trigger event @T(C1) specifies the point in time when the values of condition C1 becomes true. Similarly, trigger event @F(C1) specifies the time when condition C1 becomes false.
- **A conditioned event** is an event whose occurrence depends on the values of other conditions. For example @T(C1) WHEN(C2) describes the event of condition C1 becoming true while condition C2 remains true.
- **System state** is the set of current values of the environmental conditions.
- **A mode** is a set of system states that share a common property.
- **A modeclass** is a set of modes; the union of the modes in a modeclass must cover the system's state space.
- **The system** is in exactly one mode of each modeclass at all times.
- **A mode transition** occurs between modes in the same mode class as a result of system state changes.

An event occurs when the values of a condition change from true to false or vice versa. Events therefore specify instants of time, while conditions specify intervals of time. Formally, a conditioned event @T(C1)WHEN(C2) occurs at time t if and only if C1 is false and C2 is true at time t-ϵ, and C1 and C2 are both true at time t. In the above conditioned event, C1 is called the event's triggering event and C2 is called the event's WHEN condition. SCR semantics propose three definitions for an event occurrence and allow the requirements designer to decide which definition pertains to each event [AG93]. We use the definition adopted in Atlee and Gannon's paper: WHEN conditions must be satisfied both before and at the time of the event and triggered conditions must be unsatisfied immediately before the event and satisfied at the time of the event.

Using modes allows us to abstract away details that do not contribute to the system's behavior; the values of all system conditions are not important at all times. In fact at the requirements level, a condition's value is important only when it can trigger a transition out of the current mode. A mode transition from one mode to another is activated by the occurrence of a conditioned event. If a conditioned event can activate more than one transition from the same mode, then the specification is non-deterministic; executing exactly one of the activated transitions satisfies the requirements.

Previous Mode	Ignited	Running	Toofast	Brake	Activate	Deactivate	Resume	New Mode
Off	@T	-	-	-	-	-	-	Inactive
Inactive	@F	-	-	-	-	-	-	Off
	t	t	-	f	@T	-	-	Cruise
Cruise	@F	-	-	-	-	-	-	Off
	t	@F	-	-	-	-	-	Inactive
	t	-	@T	-	-	-	-	
	t	t	f	@T	-	-	-	Override
	t	t	f	-	-	@T	-	
Override	@F	-	-	-	-	-	-	Off
	t	@F	-	-	-	-	-	Inactive
	t	t	-	f	@T	-	-	Cruise
	t	t	-	f	-	-	@T	

Table 1: SCR Specification for the Cruise Control System

Table 1 shows the SCR specification for an automobile cruise control system, presented by Atlee and Gannon [Atl94]. The system's environmental conditions indicate whether the automobile's ignition is on (Ignited), the engine is running (Running), the automobile is going too fast to be controlled (Toofast), the brake pedal is being pressed (Brake) and whether the cruise control level is set at Activate, Deactivate or Resume. The possible states of the cruise control are abstracted into four modes: Off, Inactive, Cruise, and Override. The system always starts in mode Off.

Each row in the table specifies a conditioned event that activates a transition from the mode on the left to the mode on the right. A table entry of @T or @F under a column header C represents a triggering event @T(C) or @F(C), a table entry of "t" or "f" represents a WHEN condition WHEN[C] or WHEN[$\neg$C]. If the value of an environmental condition C does not affect a conditional event, then the table entry is marked with a hyphen "-". For example, the last row in table 1 specifies a transition from mode Override to mode Cruise due to the occurrence of conditioned event @T(Resume)WHEN[Ignited$\wedge$Running$\wedge\neg$Brake]. If the system is in mode Override at time t-ϵ, and environmental conditions Ignite and Running are true and condition Brake and Resume are false; and if at time t, condition Ignited and Running are still true and Brake is not false, but condition Resume is now true; then at time t the system is in mode Cruise.

3.2 System Properties

Modes and mode transitions specify system properties that hold under certain conditions. A mode invariant must be true when the system enters the mode, must remain true while the system stays in the mode, and can either be true or become false when the system leaves the mode. Mode invariants are the invariant properties of a system mode. If certain conditions hold then either the system is in a particular mode or the next system transition will be into that mode. Whenever the system is in a particular mode, certain system conditions have invariant values.

For example, in the cruise control system, if the system is in mode Off, then the system is Not Ignited. This is a mode invariant.

SCR specification specifies software system functional behavior. Sometimes an SCR specification includes a set of invariants as safety assertions, but these system invariants are not additional constraints on the required behavior, they are included only as the goals that the tabular requirements are expected to enforce. Also, if these invariants are not explicitly listed, then we should be able to derive them from the specification. The derived invariants can also be compared with the explicit invariants as a verification exercise. These invariants can be used for different applications, which will be discussed in later sections.

4 Finding Invariants in SCR Specifications

In order to describe the general method that can derive invariants from SCR requirements, we need to formally define the SCR mode transition table, and introduce some new concepts to be used in this general method.

- Given a SCR specification,

  1. **MODE** is a finite set that contains all the modes in a mode class.
     MODE = { M_1, M_2, ..., M_m }
  2. **ENVIRONMENTAL** conditions is a finite set that contains all the environmental conditions in the specifications.
     ENVIRONMENTAL = { C_1, C_2, ..., C_n },
     Notice that the negation of an environmental condition is also an environmental condition.
  3. **EVENTS** is a finite set that contains all the events in mode transition. Trigger event Trigger(C_i), where Trigger is either @T or @F, can be represented as a special case of conditioned event Trigger(C_i)WHEN[true]. The conditioned event is used as a general term later in this paper. Also, trigger event Trigger(C_i) will make condition C_i become true at the time the event occurs.
     EVENTS = { e_1, e_2, ..., e_n },
  4. **Event Conditions**

     For a given event e_x that causes mode transition from $mode_i$ to $mode_j$, the event condition is defined as f_{ij}. f_{ij} may have different condition values with respect to the event occurrence time t. They are defined to have two values before and after event e_x occurs at time t:

     (a) $f_{ij} = e_x$ = Trigger(C_x)WHEN [$C_{x_1} \wedge ... \wedge C_{x_t}$].
     The conditioned event here represents the occurrence of trigger event C_x provided that condition ([$C_{x_1} \wedge ... \wedge C_{x_1 t}$]) holds. x1, ..., xt $\in$ {1, ..., n}.

     (b) f^*_{ij}, called the post-event condition, represents the conditions at time t when the events just occurred.

$$f^*_{ij} = \begin{cases} (C_x) \wedge [C_{x_1} \wedge ... \wedge C_{x_t}], Trigger = @T \\ (\neg C_x) \wedge [C_{x_1} \wedge ... \wedge C_{x_t}], Trigger = @F \end{cases}$$

     (c) $^*f_{ij}$, named the pre-event condition, represents the condition at time t-ϵ, before the event occurs.

$$^*f_{ij} = \begin{cases} (\neg C_x) \wedge [C_{x_1} \wedge ... \wedge C_{x_t}], Trigger = @T \\ (C_x) \wedge [C_{x_1} \wedge ... \wedge C_{x_t}], Trigger = @F \end{cases}$$

     In the cruise control problem described in Section 5, F_{12} = @T(Brake)WHEN[Ignite], F^*_{12} = Brake $\wedge$[Ignite]
     F_{23} = @F(Brake)WHEN[Ignite], thus $F^*_{23} = \neg$ Brake $\wedge$ [Ignite]
     Notice that it is possible that there are more than two events that can cause mode transition from $mode_i$ to $mode_j$. We apply the "exclusive or" operation

$\oplus$ on to them. Therefore, $f_{ij} = e_x \oplus ... \oplus e_l$, we then have similar f^*_{ij} and $^*f_{ij}$ representations.

- The SCR mode transition table

The mode transition table is shown in table 2. There are n environmental conditions C_1, C_2, ..., C_n, and m modes M_1, M_2, .., M_m. The new modes are also one of the modes in MODE. This table specifies the events and conditions needed for mode M_i to transition to a new mode.

Previous Mode	C_1 C_2 ..... C_n	New Mode
M_1	f_{11}	M_{11}
	....	....
	f_{1k_1}	M_{1k_1}
M_2	f_{21}	M_{21}
	....	....
	f_{2k_1}	M_{2k_2}
....	....	....
M_m	f_{m1}	M_{m1}
	....	....
	f_{mk_m}	M_{mk_m}

Table 2: The SCR Mode Transition Table

f_{11},f_{1k_1}, f_{21}, f_{2k_2}, f_{m1},f_{k1}, f_{k2}, ..., f_{km} are event conditions.
$_{11}$,$_{1k_1}$, $_{21}$, $_{2k_2}$, $_{m1}$,$_{k1}$, $_{k2}$, ..., $_{km}$ $\in$ {1, 2, ..., n}
M_{11},M_{1k_1}, M_{21}, M_{2k_2}, M_{m1}, M_{k1}, M_{k2}, ..., M_{km} $\in$ MODE,
C_1, ..., C_n $\in$ ENVIRONMENTAL.

Also, there could have more than one conditioned event that can make a previous mode transition to a new mode. Thus, in table 2 there could be more than one row in the current f_{ij} row. Thus, f_{ij} is a set of disjunctions of conditioned events that make mode M_i to transition to mode M_{ij}. where i, j $\in$ {1, 2, ..., n }.
So, we have $f_{ij} = e_1 \vee e_2 \vee ... \vee e_k$, $e_k \in$ EVENTS, k=1, ..., k)

- Conditioned Transition Graph (CTG)

This paper introduces a Conditioned Transition Graph (CTG) to represent the SCR mode transition table. A CTG is obtained from the SCR mode transition table using the following steps:

1. choose (or given) a mode in the Previous Mode column as the initial mode; use it as a node in the graph.

2. from the initial mode node, if there are condition events that make the initial mode transition to new modes, then there are arcs going out from the initial mode node to each new mode node and annotated with the condition event on each arc respectively.
3. find each new mode node transitioned from the initial mode, find out what new modes it can transition to with respect to the SCR table. Repeat this step until every mode transition has been covered.

A CTG has the following definition:

CTG = (Nodes, Arcs, Conditions), where

- **Nodes** is a finite set of mode nodes, each belonging to MODE, and each appearing in the SCR mode transition table either as a previous mode or a new mode. Several nodes can have the same mode name (this is used in the invariants detection). These nodes appear as different nodes in the graph, but those nodes with same names are treated as one node in the mathematical representation of the graph. This is used in the invariant detection.
- **Arcs** and **Conditions**: if there is a transition from Mode_i to Mode_j under condition f_{ij}, (i, j : 1, ..., n), then there is an output arc coming out from Mode_i and going into Mode_j. This arc is annotated with condition f_{ij}.

For example, figure 1 gives a subgraph of CTG of the cruise control problem; Off is the initial mode, it can transition to mode Inactive if event @T(Ignite) occurs, also mode Inactive can transition back to mode Off under event @F(Ignite) or it can go to mode Cruise if event @T(Activate)WHEN[Ignited∧Running∧¬Brake] occurs. The two Off nodes are put in different place in the CTG, but semantically, they are one node Off.

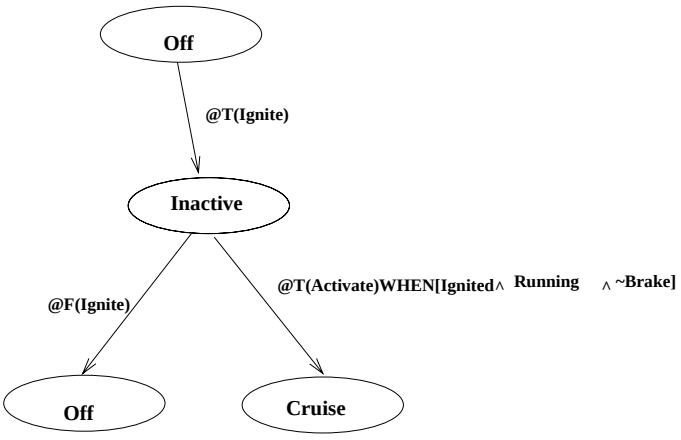

Figure 1: A subnet of the CTG

- Conditioned Incidence Matrix – M

  An incidence matrix is a mathematical representation of a graph. To represent CTG, we introduce the Conditioned incidence matrix as an extension of the traditional incidence matrix to mathematically describe CTG graphs.

  Given a CTG G=(N, A, C), m=|N|, the conditioned incidence matrix is represented by an $m \times m$ matrix denoted as M where the rows represent the condition events that make one mode on the left-hand side of the matrix transition to the modes that appear in the columns.

  Let F_{ij} denote the element in the i-th row and the j-th column in the matrix; it has one of the following values:

  - $F_{ij} = 0$ if there is no transition from $Mode_i$ to $Mode_j$.
  - $F_{ij} = 0$ if i equals j.
  - $F_{ij} = f_{ij}$ if the condition for transition from $Mode_i$ to $Mode_j$ is f_{ij}.

$$\begin{array}{c} \\ M_1 \\ M_2 \\ \dots \\ M_m \end{array} \begin{array}{c} \begin{array}{cccc} M_1 & M_2 & \dots & M_m \end{array} \\ \begin{pmatrix} 0 & F_{12} & \dots & F_{1m} \\ F_{21} & 0 & \dots & F_{2m} \\ \dots & \dots & \dots & \dots \\ F_{m1} & F_{m2} & \dots & 0 \end{pmatrix} \end{array}$$

- After-transition Matrix – A

  We introduce the After-transition Matrix to represent the possible conditions that a mode should satisfy at time t after one of the other modes transitions into this mode.

  An After-transition matrix A is obtained from the Conditioned incidence matrix by applying the "*exclusive or*" operation $\oplus$ to every element in the conditioned incidence matrix within each column.

  Given a conditioned incidence matrix M, the After-transition matrix is represented by an $m \times m$ matrix denoted as A. A_{ij} denotes the element in the i-th row and the j-th column in the matrix; it has one of the following values:

  - $A_{ij} = 0$ if i $\neq$ j.
  - $A_{ij} = F^*_{1i} \oplus F^*_{2i} \oplus ... \oplus F^*_{mi}$ if i = j where F^*_{ij} is post-event condition of F_{ij}.

  The After-transition matrix has the following form:

$$\begin{array}{c} \\ M_1 \\ M_2 \\ \\ M_m \end{array} \begin{array}{c} \begin{array}{cccc} M_1 & M_2 & \dots & M_m \end{array} \\ \begin{pmatrix} F^*_{11} \oplus F^*_{21} \oplus ... \oplus F^*_{m1} & & & \\ & F^*_{12} \oplus F^*_{22} \oplus ... \oplus F^*_{m2} & & \\ & & \dots & \\ & & & F^*_{1m} \oplus F^*_{2m} \oplus ... \oplus F^*_{mm} \end{pmatrix} \end{array}$$

- Leave-mode Matrix —- B

  The Leave-mode matrix is obtained from the conditioned incidence matrix by disjuncting every element in the conditioned incidence matrix in each row. It shows the possible condition events that make the current mode leave for another mode.

  Given a conditioned incidence matrix M, the Leave-mode matrix is represented by an $m \times m$ matrix denoted as B. B_{ij} denotes the element in the i-th row and the j-th column in the matrix; it has one of the following values:

  - $B_{ij} = 0$ if i $\neq$ j.
  - $B_{ij} = F_{i1} \vee F_{i2} \vee ... \vee F_{im}$ if i = j.

  The Leave-mode matrix has the following form:

$$\begin{array}{c} \\ M_1 \\ M_2 \\ \dots \\ M_m \end{array} \begin{array}{c} \begin{array}{cccc} M_1 & M_2 & \dots & M_m \end{array} \\ \begin{pmatrix} F_{11} \vee F_{12} \vee ... \vee F_{1m} & & & \\ & F_{21} \vee F_{22} \vee ... \vee F_{2m} & & \\ & & \dots & \\ & & & F_{m1} \vee F_{m2} \vee ... \vee F_{mm} \end{pmatrix} \end{array}$$

- The Invariant Detection Algorithm

  With respect to a graph, the After-transition matrix and the Leave-mode matrix shows the input conditions to a mode and the output events for leaving a mode respectively. For example, figure 2 shows the input conditions and output events of Mode 1.

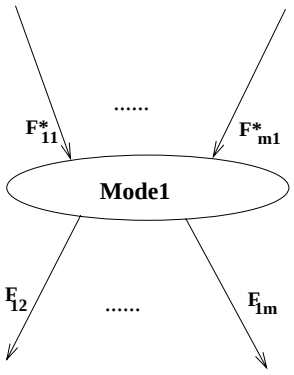

Figure 2: Input Conditions and output events of Mode 1

A mode invariant must be true when the system enters the mode, and must remain true while the system stays in the mode, and once it is false, the system must leave the mode. Negations of the Leave-mode conditions are candidate invariant conditions if the trigger condition satisfies one of the after-transition conditions, and also the

WHEN conditions of the leave-mode condition should also satisfy the after-transition conditions. This means, if a condition has to be satisfied when entering the mode, and if it is not satisfied the mode must transition to another mode, it is a mode invariant.

Therefore, the invariant detection algorithm can be described in the following steps.

1. Draw the CTG graph based on SCR mode transition table.
2. Obtain M, the Conditioned incidence matrix for the CTG.
3. Obtain A, the After-Transition matrix from the Conditioned incidence matrix.
4. Obtain B, the Leave-mode matrix from the Conditioned incidence matrix.
5. Compare Matrix A and B to find the mode invariants.

step 1 For each A_{ii}, (i=1 to n in matrix A)

$A_{ii} = F^*_{1i} \oplus F^*_{2i} \oplus ... \oplus F^*_{mi}$

$= (C_{x_1}) \wedge [\ C_{x_1 1} \wedge ... \wedge C_{x_1 t1}] \oplus ... \oplus (C_{xi}) \wedge [\ C_{xi1} \wedge ... \wedge C_{xiti}]$

$\oplus$ ...

$\oplus (C_{y_1}) \wedge [\ C_{y_1 1} \wedge ... \wedge C_{y_1 s1}] \oplus ... \oplus (C_{yi}) \wedge [\ C_{yi1} \wedge ... \wedge C_{yisi}]$

$x_1, x_{11}, x_1 t_1, ..., x_i, x_{i1}, x_i t_i, ..., y_1, y_{11}, y_1 s_1, ... y_{i1}, y_i s_i \in \{ 1, ..., n\}$.

The postcondition set (**Postset**) contains all the environmental conditions that appear in the post-event conditions of an particular element in matrix A_{ii}, where $i \in \{ 1, ..., n \}$.

$Postset(A_{ii}) = \{ \{ C_{x1}, C_{x_1 1}, ..., C_{x_1 t1} \}, ..., , \{ C_{xi}, C_{x_i 1}, ..., C_{x_i ti} \},$
$\{ C_{y1}, C_{y_1 1}, ..., C_{y_1 s1} \}, ..., , \{ C_{yi}, C_{y_i 1}, ..., C_{y_i si} \}\}$

Step 2 $B_{ii} = F_{i1} \vee F_{i2} \vee ... \vee F_{im}$

each $F_{ij} = Trigger(C_{yi}) \wedge [\ C_{yi1} \wedge ... \wedge C_{yik}]$

Define the Leave-Condition **LC** as the event condition at time t-ϵ before the trigger event occurs:

$$LC(F_{ij}) = \begin{cases} \neg C_{yi}, Trigger = @T \\ C_{yi}, Trigger = @F \end{cases}$$

Define the WHEN condition set **WC** as the environmental conditions that appear as the WHEN condition of F_{ij}.

$WC(F_{ij}) = \{C_{yi1}, ..., C_{yik}\}$

Step 3 For each F_{ij} (j= 1 to m in B_{ii})

if LC(F_{ij}) $\in$ each element of Postset(A_{ii}) and WC(F_{ij}) $\subseteq$ each element of Postset(A_{ii}) then $\mathbf{Mode}_i \rightarrow \mathbf{LC(F}_{ij}\mathbf{)}$ is an invariant.
Repeat Step 3 until j=m.

Step 4 Repeat Step 1 to Step 3 until i=n, then stop.

5 An Application

We now apply the invariant detection algorithm to the cruise control problem described in Section 5. The SCR mode transition table is given in table 2. In order to find out mode invariants from table 2, we apply the invariant algorithm described in Section 4 and each step is shown as follows.

1. Obtain the CTG for the cruise control problem from the SCR mode transition table. The CTG is shown in figure 3.

   In this CTG, mode Off is the given initial mode; it can transition to mode Inactive if @T(Ignite) occurs at time t. Also, node Inactive has two output arcs, one to Off node when @F(Ignite) occurs, and another to Cruise mode when @T(Activate)WHEN[Ignite $\wedge$ Running $\wedge$ $\neg$ Brake] occurs. Similarly, Cruise node goes to three different nodes, and Override node has three output arcs. From Cruise to Inactive node, there are actually two events that can make the transition happen, they are connected by an "exclusive or" operator $\oplus$ annotated on the arc.

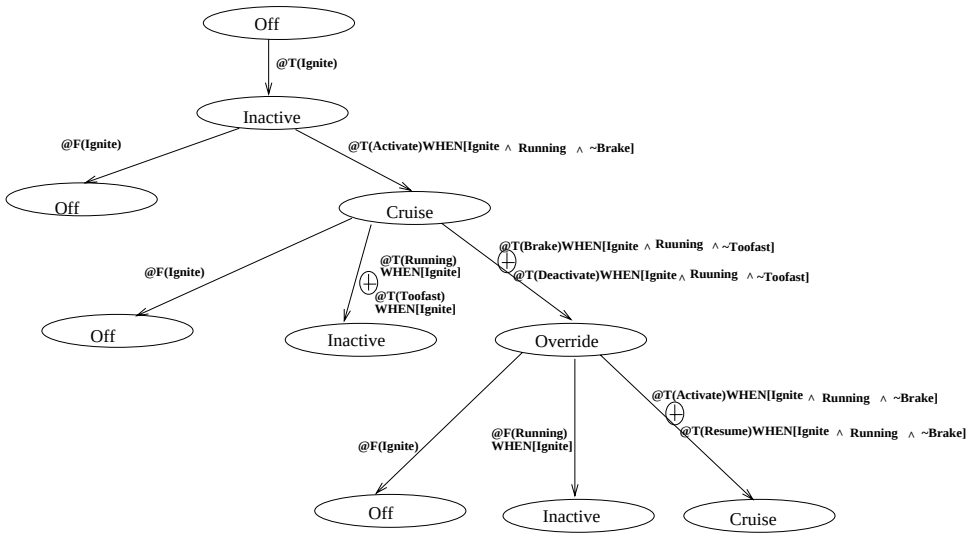

Figure 3: CTG graph for cruise control problem

2. Obtain matrix M, the conditioned incidence matrix, from the CTG.

   The Conditioned incidence matrix M shows the nodes and input/output arc relations in the CTG. For instance, for mode Off to transition to mode Inactive, @T(Ignite) has to occur, while no transition goes directly from mode Off to mode Cruise, so the corresponding matrix element is 0. The conditioned incidence matrix for CTG is shown as:

$$
\begin{array}{c}
 \\
Off \\
Inactive \\
Cruise \\
Override
\end{array}
\begin{array}{c}
\begin{array}{cccc}
Off & Inactive & Cruise & Override
\end{array} \\
\left(\begin{array}{cccc}
0 & @T(Ignite) & 0 & 0 \\
@F(Ignite) & 0 & @T(Activate)WHEN & 0 \\
 & & [Ignite \wedge Running \wedge \neg Brake] & \\
@F(Ignite) & @F(Running)WHEN[Ignite] & 0 & @T(Brake)WHEN \\
 & \oplus @T(Toofast)WHEN[Ignite] & & [Ignite \wedge Running \wedge \neg Toofast] \\
 & & & \oplus @T(Deactivate)WHEN \\
 & & & [Ignite \wedge Running \wedge \neg Toofast] \\
@F(Ignite) & @F(Running)WHEN[Ignite] & @T(Activate)WHEN & 0 \\
 & & [Ignite \wedge Running \wedge \neg Brake] & \\
 & & \oplus @T(Resume)WHEN & \\
 & & [Ignite \wedge Running \wedge \neg Brake] &
\end{array}\right)
\end{array}
$$

3. Obtain matrix A, the After-transition matrix, from matrix M.

   Under each mode column in matrix M, each element in the column corresponds to one input event that makes other modes transition to the current mode. If any one of these conditions is satisfied, the system will be in the mode under the current column. For instance, Off, Cruise, and Override can each transition to a mode Inactive if one of the corresponding events occurs. The system will then be in mode Inactive and it will satisfy the enter condition. Inactive thus will satisfy the following conditions:

   Ignite $\oplus$(Ignite $\wedge$ $\neg$ Running) $\oplus$ (Ignite $\wedge$ Toofast) $\oplus$ (Ignite $\wedge$ $\neg$ Running).

   The matrix A is shown as follows.

$$
\begin{array}{c}
 \\
Off \\
Inactive \\
Cruise \\
Override
\end{array}
\begin{array}{c}
\begin{array}{cccc}
Off & Inactive & Cruise & Override
\end{array} \\
\left(\begin{array}{cccc}
\neg Ignite & & & \\
\oplus \neg Ignite & & & \\
\oplus \neg Ignite & & & \\
 & (Ignite) & & \\
 & \oplus((\neg Running) \wedge [Ignite]) & & \\
 & \oplus((Toofast) \wedge [Ignite]) & & \\
 & \oplus((\neg Running) \wedge [Ignite]) & & \\
 & & ((Activate) \wedge [Ignite \wedge Running \wedge \neg Brake]) & \\
 & & \oplus((Activate) \wedge [Ignite \wedge Running \wedge \neg Brake]) & \\
 & & \oplus((Resume) \wedge [Ignite \wedge Running \wedge \neg Brake]) & \\
 & & & ((Brake) \wedge [Ignite \\
 & & & \wedge Running \wedge \neg Toofast]) \\
 & & & \oplus((Deactivate) \wedge [Ignite \\
 & & & \wedge Running \wedge \wedge \neg Toofast])
\end{array}\right)
\end{array}
$$

4. Obtain Matrix B, the leave-mode matrix. For each mode row in matrix M, it shows the events that make the current mode leave for a different mode. If these events occur, then the mode may transition to one of the other modes. For example, mode Cruise can leave for mode Off if @F(Ignite) occurs; or it can turn to mode Inactive if @F(Running)WHEN[Ignite]∨ @T(Toofast)WHEN[Ignite] occurs; or Cruise mode can transition to mode Override if @T(Brake) WHEN[Ignite ∧ Running ∧ ¬ Toofast] occurs. The events that make Cruise mode leave for other modes are: @F(Ignite) ∨ (@F(Running)WHEN[Ignite]∨ @T(Toofast)WHEN[Ignite]) ∨ @T(Brake) WHEN[Ignite ∧ Running ∧ ¬ Toofast].

	Off	$Inactive$	$Cruise$	$Override$
Off	$@T(Ignite)$			
$Inactive$		$(@F(Ignite))$ $\vee(@T(Activate)WHEN$ $[Ignite \wedge Running \wedge \neg Brake])$		
$Cruise$			$(@F(Ignite))$ $\vee(@F(Running)WHEN[Ignite])$ $\vee(@T(Toofast)WHEN[Ignite])$ $\vee(@T(Brake)WHEN$ $[Ignite \wedge Running \wedge \neg Toofast])$ $\vee(@T(Deactivate)WHEN$ $[Ignite \wedge Running \wedge \neg Toofast])$	
$Override$				$(@F(Ignite))$ $\vee(@F(Running)WHEN[Ignite])$ $\vee(@T(Activate)WHEN$ $[Ignite \wedge Running \wedge \neg Brake])$ $\vee(@T(Resume)WHEN$ $[Ignite \wedge Running \wedge \neg Brake])$

5. Compare matrix A and matrix B and find invariants.

   (a) Compare A_{11} and B_{11}:
   A_{11} = ¬ Ignite ⊕ ¬ Ignite ⊕ ¬ Ignite
   Postset(A_{11}) = { {¬ Ignite}}
   B_{11} = @T(Ignite)
   LC(f_{11}) = ¬ Ignite
   WC(f_{11}) = { }
   So, LC(f_{11}) ∈ {¬ Ignite}(element in Postset(A_{11})),
   and WC(f_{11}) ⊆ {¬ Ignite} so Off → LC(f_{11}) is an invariant. Therefore, we have the mode invariant for Off as:
   Off → ¬ Ignite

   (b) Compare A_{22} and B_{22}:
   A_{22} = (Ignite ∧ ¬ Running) ⊕ (Ignite ∧ Toofast) ⊕ (Ignite ∧ ¬ Running)
   Postset(A_{22}) ={ { Ignite, ¬ Running}, { Ignite, Toofast }}
   B_{22} = @F(Ignite) ⊕ @F(Activate)WHEN[Ignite ∧ Running ∧ ¬ Brake]

i. $LC(f_{21})$ = Ignite, $WC(f_{21})$ = { },
$LC(f_{21}) \in$ { Ignite,$\neg$ Running}, and $LC(f_{21}) \in$ { Ignite,Toofast}
also, $WC(f_{21}) \subseteq$ { Ignite, $\neg$ Running} and $WC(f_{21}) \subseteq$ { Ignite, Toofast}
so Inactive $\rightarrow$ $LC(f_{21})$ is an invariant. Therefore, we have the mode invariant for Inactive as:
Inactive → Ignite

ii. $LC(f_{22})$ = Activate,
$WC(f_{22})$ = { Ignite,Running, $\neg$ Brake}
$LC(f_{22}) \notin$ { Ignite,$\neg$ Running}, and $LC(f_{22}) \notin$ { Ignite,Toofast}
also, $WC(f_{22}) \not\subseteq$ { Ignite, $\neg$ Running} and $WC(f_{22}) \not\subseteq$ { Ignite, Toofast}
so, Inactive $\rightarrow$ $LC(f_{22})$ (Inactive $\rightarrow$ Activate) is not an invariant.

(c) Mode Cruise, compare A_{33} and B_{33}:

A_{33} = (Activate $\wedge$ Ignite $\wedge$ Running $\wedge$ $\neg$ Brake)
$\oplus$ (Activate $\wedge$ Ignite $\wedge$ Running $\wedge$ $\neg$ Brake)
$\oplus$ (Resume $\wedge$ Ignite $\wedge$ Running $\wedge$ $\neg$ Brake)

Postset(A_{33}) = { {Activate, Ignite, Running, ¬Brake}, {Resume, Ignite, Running, ¬Brake} }
B_{33} = @F(Ignite) $\oplus$ @F(Running)WHEN[Ignite]
$\oplus$ @T(Toofast)WHEN[Ignite]
$\oplus$ @T(Brake)WHEN[Ignite $\wedge$ Running $\wedge$ $\neg$ Toofast]
$\oplus$ @T(Deactivate)WHEN[Ignite $\wedge$ Running $\wedge$ $\neg$ Toofast]

i. $LC(f_{31})$ = Ignite
$WC(f_{31})$ = { }
$LC(f_{31}) \in$ {Activate, Ignite, Running, ¬Brake}
$LC(f_{31}) \in$ {Resume, Ignite, Running, ¬Brake}
also, $WC(f_{31}) \subseteq$ {Activate, Ignite, Running, ¬Brake}
and $WC(f_{31}) \subseteq$ {Resume, Ignite, Running, ¬Brake}.
So, Cruise $\rightarrow$ $LC(f_{31})$ is an invariant.
Cruise → Ignite

ii. $LC(f_{32})$ = Running $WC(f_{32})$ = {Ignite}
$LC(f_{32}) \in$ {Activate, Ignite, Running, ¬Brake}
and $LC(f_{32}) \in$ {Resume, Ignite, Running, ¬Brake}
also, $WC(f_{32}) \subseteq$ {Activate, Ignite, Running, ¬Brake}
and $WC(f_{32}) \subseteq$ {Resume, Ignite, Running, ¬Brake}.
So, Cruise $\rightarrow$ $LC(f_{32})$ is an invariant.
Cruise → Running

iii. $LC(f_{33}) = \neg$ Toofast $WC(f_{33}) = \{$Ignite$\}$
$LC(f_{33}) \notin \{$Activate, Ignite, Running, $\neg$Brake$\}$
and $LC(f_{33}) \notin \{$Resume, Ignite, Running, $\neg$Brake$\}$
also, $WC(f_{33}) \not\subseteq \{$Activate, Ignite, Running, $\neg$Brake$\}$
and $WC(f_{33}) \not\subseteq \{$Resume, Ignite, Running, $\neg$Brake$\}$.
So, Cruise $\rightarrow LC(f_{33})($ Cruise $\rightarrow \neg$ Toofast) is not an invariant.

iv. $LC(f_{34}) = \neg$ Brake $WC(f_{34}) = \{$Ignite, Running, $\neg$Toofast$\}$
$LC(f_{34}) \in \{$Activate, Ignite, Running, $\neg$Brake$\}$
and $LC(f_{34}) \in \{$Resume, Ignite, Running, $\neg$Brake$\}$
but, $WC(f_{34}) \not\subseteq \{$Activate, Ignite, Running, $\neg$Brake$\}$
and $WC(f_{34}) \not\subseteq \{$Resume, Ignite, Running, $\neg$Brake$\}$.
So, Cruise $\rightarrow LC(f_{34})($ Cruise $\rightarrow \neg$ Brake) is not an invariant.

v. $LC(f_{35}) = \neg$ Deactivate, $WC(f_{35}) = \{$Ignite, Running, $\neg$Toofast$\}$
$LC(f_{35}) \notin \{$Activate, Ignite, Running, $\neg$Brake$\}$,
and $LC(f_{35}) \notin \{$Resume, Ignite, Running, $\neg$Brake$\}$
but, $WC(f_{35}) \not\subseteq \{$Activate, Ignite, Running, $\neg$Brake$\}$
and $WC(f_{35}) \not\subseteq \{$Resume, Ignite, Running, $\neg$Brake$\}$.
So, Cruise $\rightarrow LC(f_{35})($ Cruise $\rightarrow \neg$ Deactivate) is not an invariant.

So, we have the mode invariant for Cruise mode as:

Cruise $\rightarrow$ Ignite $\wedge$ Running

(d) Mode Override, compare A_{44} and B_{44}:

A_{44} = (Brake $\wedge$ Ignite $\wedge$ Running $\wedge \neg$ Toofast)
$\oplus$ (Deactivate $\wedge$ Ignite $\wedge$ Running $\wedge \neg$ Toofast)

Postset(A_{44}) ={ { Brake, Ignite, Running, $\neg$ Toofast }, { Deactivate, Ignite, Running, $\neg$ Toofast }}
B_{44} = @F(Ignite)
$\oplus$ @F(Running)WHEN[Ignite]
$\oplus$ @T(Activate)WHEN[Ignite $\wedge \neg$ Brake]
$\oplus$ @T(Resume)WHEN[Ignite $\wedge \neg$ Brake]

i. $LC(f_{41})$ =Ignite $WC(f_{41}) = \{\ \}$
$LC(f_{41}) \in \{$ Brake, Ignite, Running, $\neg$ Toofast $\}$
and $LC(f_{41}) \in \{$ Deactivate, Ignite, Running, $\neg$ Toofast $\}$

also, WC(f_{41}) $\subseteq$ { Brake, Ignite, Running, $\neg$ Toofast }
and WC(f_{41}) $\subseteq$ { Deactivate, Ignite, Running, $\neg$ Toofast }}
So, Override $\rightarrow$ LC(f_{41})(Override $\rightarrow$ Ignite) is an invariant.

Override $\rightarrow$ Ignite

ii. LC(f_{42}) =Running WC(f_{42}) = {Ignite}
LC(f_{42}) $\in$ { Brake, Ignite, Running, $\neg$ Toofast }
and LC(f_{42}) $\in$ { Deactivate, Ignite, Running, $\neg$ Toofast }
also, WC(f_{42}) $\subseteq$ { Brake, Ignite, Running, $\neg$ Toofast }
and WC(f_{42}) $\subseteq$ { Deactivate, Ignite, Running, $\neg$ Toofast }}
So, Override $\rightarrow$ LC(f_{42})(Override $\rightarrow$ Running) is an invariant.

Override $\rightarrow$ Running

iii. LC(f_{43}) =$\neg$ Activate WC(f_{43}) = {Ignite, Running, $\neg$ Brake}
LC(f_{43}) $\notin$ { Brake, Ignite, Running, $\neg$ Toofast }
and LC(f_{43}) $\notin$ { Deactivate, Ignite, Running, $\neg$ Toofast }
also, WC(f_{43}) $\not\subseteq$ { Brake, Ignite, Running, $\neg$ Toofast }
and WC(f_{43}) $\not\subseteq$ { Deactivate, Ignite, Running, $\neg$ Toofast }}
So, Override $\rightarrow$ LC(f_{43})(Override $\rightarrow$ $\neg$ Activate) is not an invariant.

iv. LC(f_{44}) =$\neg$ Resume WC(f_{44}) = {Ignite, Running, $\neg$ Brake}
LC(f_{44}) $\notin$ { Brake, Ignite, Running, $\neg$ Toofast }
and LC(f_{44}) $\notin$ { Deactivate, Ignite, Running, $\neg$ Toofast }
also, WC(f_{44}) $\not\subseteq$ { Brake, Ignite, Running, $\neg$ Toofast }
and WC(f_{44}) $\not\subseteq$ { Deactivate, Ignite, Running, $\neg$ Toofast }}
So, Override $\rightarrow$ LC(f_{44})(Override $\rightarrow$ $\neg$ Resume) is not an invariant.

So, overall, we have the invariants in mode Override as:
Override $\rightarrow$ Ignite $\wedge$ Running

In summary, the mode invariants derived from the SCR mode transition table are:
Off $\rightarrow$ $\neg$ Ignite
Inactive $\rightarrow$ Ignite
Cruise $\rightarrow$ Ignite $\wedge$ Running
Override $\rightarrow$ Ignite $\wedge$ Running

These invariants match the results in Atlee and Gannon's paper [AG93].

Atlee and Gannon [AG93] pointed out that some of the independently given safety properties do not match the model checking results originated from the SCR table. We revised the mode transition table as shown in table 3.

Cur Mode	*Ignited*	*Running*	*Toofast*	*Brake*	*Activate*	*Deact*	*Resume*	New Mode
OFF	@T	—	—	—	—	—	—	INACTIVE
INACTIVE	@F	**f**	—	—	—	—	—	OFF
INACTIVE	@F	**@F**	—	—	—	—	—	OFF
INACTIVE	t	t	f	f	@T	**f**	**@F**	CRUISE
INACTIVE	t	t	f	f	@T	**@F**	**f**	CRUISE
CRUISE	@F	**@F**	—	—	—	—	—	OFF
CRUISE	t	@F	—	—	—	—	—	INACTIVE
CRUISE	t	—	@T	—	—	—	—	INACTIVE
CRUISE	t	t	f	@T	—	—	—	OVERRIDE
CRUISE	t	t	f	—	**@F**	@T	**f**	OVERRIDE
CRUISE	t	t	f	—	**f**	@T	**@F**	OVERRIDE
OVERRIDE	@F	**@F**	—	—	—	—	—	OFF
OVERRIDE	t	@F	—	—	—	—	—	INACTIVE
OVERRIDE	t	t	f	f	@T	**f**	**@F**	CRUISE
OVERRIDE	t	t	f	f	@T	**@F**	**f**	CRUISE
OVERRIDE	t	t	f	f	**f**	**@F**	@T	CRUISE
OVERRIDE	t	t	f	f	**@F**	**f**	@T	CRUISE

Initial Mode : OFF

Table 3: Mode Transitions for Adjusted Cruise Control Specifications

The invariants calculated from this SCR mode transition table are as follows. The mode Cruise invariant satisfies the safety assertion.

- **Off** $\rightarrow \neg$ **Ignite**
- **Inactive** $\rightarrow$ **Ignite**
- **Cruise** $\rightarrow$ **Ignite** $\wedge$ **Running** $\wedge \neg$ **Toofast** $\wedge \neg$ **Brake**
- **Override** $\rightarrow$ **Ignite** $\wedge$ **Running**

6 Discussion

It can be seen from previous sections that system properties such as mode invariants can be derived directly from the SCR mode transition table. The calculated invariant results match those in Atlee and Gannon's work [AG93]. The derivation method is used in this paper is completely different. The invariant detection algorithm presented in this paper is actually a mechanical procedure that can be easily automated and the computation complexity is low (the size of each of the three matrices is the number of modes defined in the SCR mode transition table.). The automated procedure can be used to check the invariant properties

or to verify the independently given assertions. Also, we can use the CTG and matrix M, A, and B for other kinds of analysis. Due to time and space limit, these analysis are only introduced and highlighted here.

1. **Generate test cases from the CTG**. For CTG graphs, we can develop test requirements that are independent of the design for the system. We can choose a type of coverage, for example, branch coverage, to generate test requirement (coverage paths). For the cruise control system, we list the test requirements, the branch coverage paths, test cases are therefore the sequences of events that can cause each path to be covered. For instance, test requirements requires to cover the following paths:

   - Off→Inactive
   - Off→Inactive→Off
   - Off→Inactive→Cruise
   - Off→Inactive→Cruise→Off
   - Off→Inactive→Cruise→Inactive
   - Off→Inactive→Cruise→Override
   - Off→Inactive→Cruise→Override→Off
   - Off→Inactive→Cruise→Override→Inactive
   - Off→Inactive→Cruise→Override→Cruise

2. **Non-deterministic case**. This does not matter in the invariant detection algorithm. It can be viewed that two input arc to a mode has the same condition event. At time t, only only one arc can be chosen. They have been represented as "exclusive or" in the CTG.

3. **Multi-mode class case**. In this paper, we only discuss the mode invariants in a modeclass, in case of a multi-mode class, we can find out mode invariants under different modeclass case. Because the system will always be at one modeclass at a time, so we can view the system at a particular time period as in one modeclass. This will not affect the invariant detection method.

7 Conclusions

This paper uses SCR specifications mode transition tables to derive mode invariants for the specified system. The Conditioned Transition Graph(CTG) and two corresponding matrices are introduced to help to find the invariants. This invariant detection method is applied to the cruise control problem and results. This method provides a mechanical procedure

that can be easily automated. It views the SCR specifications from the SCR structure's point of view and analyzes mode invariants as structural properties. It presents a new way to analyze software requirements documents and detects implicitly implied system property information that can be very useful and important in understanding, developing, and testing the software system. Related issues such as generating test cases that are independent of the design structure based on the SCR specifications are discussed to reveal possible research problems and directions in the future.

References

[AFB+88] T. Alspaugh, S. Faulk, K. Britton, R. Parker, D. Parnas, and J. Shore. Software requirements for the A-7E aircraft. Technical report, Naval Research Laboratory, 1988.

[AG93] J. M. Atlee and J. Gannon. State-based model checking of event-driven system requirements. *IEEE Transactions on Software Engineering*, 19(1):24–40, January 1993.

[Atl94] J. M. Atlee. Native model-checking of SCR requirements. In *Fourth International SCR Workshop*, November 1994.

[Bro89] M. Browne. *Automated Verification of Finite State Machines Using Temporal Logic*. PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[Dav93] A. M. Davis. *Software Requirements*. PTR Prentice Hall, Englewood Cliffs, NJ, 1993.

[Hen80] K. Heninger. Specifying software requirements for complex systems: New techniques and their applications. *IEEE Transactions on Software Engineering*, SE-6(1):2–12, January 1980.

[HLK93] C. Heitmeyer, B. Labaw, and D. Kiskis. Consistency checks for SCR-style requirements specifications. Technical report, Naval Research Laboratory, December 1993.

[HPSK78] K. Heninger, D. Parnas, J. Shore, and J. Kallander. Software requirements for the A-7E aircraft. Technical report, Naval Research Laboratory, 1978.