INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

... To my parents, Jacqueline and Negib Rafoul

God bless you

Detection of Feature Interaction Using Relational Algebra

© by
Elias Rafoul

A thesis
presented to the School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of M.Sc in Computer Science

University of Ottawa
Ottawa, Ontario, Canada
November 1998

Abstract

A relational method is presented for specifying a software system and detecting feature interaction. The method used allows for the independent specification of features and the detection of interactions between features. This method is based on the lattice properties of the refinement ordering between specifications. The specification of a feature is given by a relation between the inputs and the outputs of a system and the combination of the features is given by taking the greatest lower bound of all features.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to send my gratitude to the following friends, family members and colleagues. First and foremost, to Dr. Ali Mili who agreed to supervise me and introduced me to the area of relational algebra and formal specifications. It is through his help, dedication and guidance that I was able to complete this thesis. To Dr. Marc Frappier who was extremely responsive to every question I had. To Serge Colle, my research partner, with whom I spent countless hours discussing, disagreeing and researching for this thesis. To Fida for maintaining her patience and keeping me motivated. Last but not least, to my family members who believed in me. Specifically, Jacqueline, Negib, Joseph Abou Chakra, Rita, Joseph and Chafi. They have collectively inspired me to continue my education, persevere and face the challenges of life.

Part I

Background

Chapter 1

Introduction

1.1 Background

Software accounts for most of the cost in a telecommunication network [39]. As a result, software faults are very costly and catastrophic; this is because a software fault may cause many failures throughout the network. As systems continue to get bigger and more complex due to high consumer demands, the likelihood of problems occurring increases. Features and subsystems are continually being added or integrated into existing systems. Consequently, the coexistence of features in a system has proven to be problematic.

The problem of feature interaction has received considerable attention over the years. A *feature* is a set of functionalities that are added incrementally to an existing software system. *Feature interactions* are all interactions that interfere with the desired operation and functionality between a feature and its environment. This includes other features as well as other instances of the same feature [11].

This problem arises in many systems such as medical, banking and information management systems. One domain where this problem is especially acute is in telecommunication systems. Some systems offer up to 800 features. Users demand complex services from their service providers such as call display, call forwarding, call waiting, etc.

1.2 Effects of the Problem

The effect of the feature interaction problem is that features and services operating correctly cannot do so when they coexist in a network. The service providers may well be unaware of this problem until complaints from clients arise. At this stage, investigating the problem and resolving it translates into high labor costs and lost revenue from the service. Moreover, without adopting a technique for avoiding these interactions, it is likely that any new services will lead

to the same problem and costs. Generally, the problem is detected and resolved at the testing phase by expert testers. As an example, a temporary solution may be to disallow the activation of two features that interact such as call forwarding and call waiting services. However, as the number of services and complexity of services increase, this approach becomes ineffective.

1.3 Existing Solutions

Providing a solution to the feature interaction problem has proven to be difficult primarily because the problem cannot be easily isolated to a particular set of conditions. There are instances of feature interactions that involve varying number of features, subscribers, circumstances, components and networks.

Current approaches to the problem encompass techniques for each stage of the software engineering lifecycle, from the specification phase to the run-time phase. Furthermore, these techniques may be used for different levels of model abstraction [23, 24] such as application level, signalling level and transport level. With such a large number of views on the feature interaction problem, no single approach can attack the problem in its entirety. The most common approach is to tackle the problem at the *specification or requirement* stage. These approaches usually use two languages: one to represent the system being modeled and one to represent the properties sought. The following representations are commonly used:

- *SDL* [23, 24, 31]
- *LOTOS* [8, 17]
- *Z* [29]
- *Delphi* [5]
- *state machine and rule based representations* [15, 19, 21, 42]
- *extended specifications* [28, 43]
- *statistical and service usage* [26, 27]
- *temporal logic* [3, 4, 13]

Most of these techniques involve a form of state-space analysis and are proficient at detecting the problem. However, most require manual modifications of the specifications to resolve the interactions.

1.4 Our Approach

Fundamentally, it is important to detect feature interaction at the specification level because resolutions may be proposed before the features are implemented and integrated into the underlying system. This reduces cost and complexity for the service providers. In this thesis, we give attention to the specification generation and specification validation phase of the software engineering lifecycle. A method is introduced which allows each feature to be specified independently. It also allows for the detection of feature interaction at the validation phase. This method is based on the refinement of lattice specification whereby a feature specification is given by a relation between the inputs and outputs of a system. Combining system features is given by taking the greatest lower bound of all the features in the underlying system. By using this method, a definition of feature interaction can be established.

Formally, two or more features interact if they do not have a greatest lower bound in the refinement lattice. Therefore, detection of a feature interaction requires proving that the greatest lower bound does not exist. The complete feature specification is given by writing inductive definitions of relations using axioms in predicate calculus. A tabular format is introduced to store the set of axioms of the specification space.

The method used is employed by using POTS (Plain Old Telephone Service) as an illustrative example. Telecommunication features are added incrementally to POTS. These features include call waiting (CW), three way calling (TWC), abbreviated dialing (ABD), originating call screening (OCS) and call forwarding (CF). Each feature is specified independently and feature interaction between these features is discussed.

1.5 Categorization Of The Problem

In order to improve the understanding of the scope of the feature interaction problem in telecommunication systems, E.J. Cameron et al [11] have categorized feature interactions in two ways:

- by the *nature* of the interaction
- by the *cause* of the interaction

1.5.1 Categorization By the Nature of Interactions

The categorization by nature has three dimensions: the kind of features involved; the number of users involved; the number of network components.

The first dimension distinguishes interactions that involve only *customer features* from interactions that involve *system features*, instead of or in addition to customer features. Customer features include all of the call processing features that are visible to the general public, such as call waiting (CW) and call forwarding (CF).

The second dimension distinguishes *single-user* interactions from *multiple-user* interactions. Single-user interactions arise when different features interfere with one another while simultaneously activated by a single user. Multiple-user interactions arise when features activated by one user interfere with those activated by another user.

The third dimension distinguishes between *single-component* interactions, and *multiple-component* interactions. Single-component interactions arise when only one network component (eg. switching system, service node) is involved in the processing while multiple-component interactions arise when features supported on one network component interfere with the operations of features supported on another network component.

Possible interactions are interactions of the following types:

- **SUSC** (Single-User-Single-Component). These interactions occur because incompatible features are simultaneously in use by a single user in a single network component such as a switching element.
- **SUMC** (Single-User-Multiple-Component). These interactions occur when there is a coordination problem in the telephone network where features accessible to one customer are deployed in different network components. Interactions of this type arise when the existence of features is unknown to designers of features in other network components.
- **MUSC** (Multiple-User-Single-Component). These interactions can occur when two or more customers access the features associated with a physical line.
- **MUMC** (Multiple-User-Multiple-Component). These interactions can occur where two or more users access features supported on multiple network components.
- **CUSY** (CUstomer-SYstem). These interactions occur between a customer feature and any system feature for operations, administration services or maintenance.

1.5.2 Categorization By the Cause of Interactions

The categorization by cause includes: violation of assumptions about the system; limitations on network support; and problems due to large heterogeneous distributed systems.

All features in a telecommunication network operate under a set of assumptions. These assumptions include a specific call processing model, some architectural support, a special billing system or even how customers perceive the network operates.

1.5.2.1 Violation of Feature Assumptions

Assumptions about data availability, call control and signaling protocol are among several kinds of assumptions that once violated can result in some interactions.

The network maintains a large amount of information. Some features assume that certain kinds of data are widely available while other features may make the same data private. Features cannot work together unless they are able to obtain the data needed by their functionalities.

Call Control is the ability of a user to manipulate (eg. accept, refuse, terminate) a call. Features such as *911* establish a master-slave relationship. Once the call is made, only the *911* operator can terminate it. Other features allow a user to put other parties on hold. Interactions arise when the call control of one feature prevents other features from exercising their control.

In the past, a simple protocol was used to provide the status of the called party when a call attempt is in progress to the calling party. A busy signal indicates that the terminating line is in use while a ring-back indicates that the line is available. Interactions will occur when one feature needs to know the status of the other party but cannot tell exactly what the status is from the signal received.

1.5.2.2 Limitations on Network Support

Network components have limited capabilities in communicating with other network components of processing calls. Interactions will arise when two features conflict over the reception of the same signal or the usage of the same functionality.

The set of signals for a switch may be limited to *, #, the ten digits 0-9, flashhook and disconnect. With the creation of many new services, the user needs to send signals for a variety of purposes to the switch. Since there are so few signals available, the same signal is often used to mean different things in different contexts. Interactions arise when the interpretation of a signal is ambiguous.

1.5.2.3 Intrinsic Problems in Distributed Systems

Many difficulties arise when dealing with feature interactions. One such difficulty is resource contention where the activation of one feature makes the other features unavailable. Another difficulty is the distribution of feature support in the network. In the past, one could assume that all features accessible by a single Customer Premise Equipment(CPE) were supported by the same network component. Therefore, the network component would be aware of all the feature logic, and could coordinate the execution of various features. This is not necessarily true for the present network architecture, as different features may be supported by different components even when the features are used by a single user. As a result, activating one feature in one component may prevent processing of features in another component.

Some features require each subscriber of the feature to assign values to feature parameters before using the feature. For example, the subscriber of Call Forwarding must provide the directory number to which incoming calls will be forwarded. The assignment of values to feature parameters defines the subset of the numbering plan in which the feature is interested. In some cases, a particular set of assignments can make two otherwise independent features collide with each other. This type of interaction is caused by personalized instantiation.

1.6 A Comparison With A Rule-Based Approach

In [37], the feature interaction problem is detected and resolved at the specification level. A rule-based method for the description of service specifications is adopted. This rule-based method specifies the service as a set of rules, each describing a feature of the service.

As previously discussed, telecommunication services can be modeled by a state transition machine, in which a state consisting of local states of the service users successively moves to a next state by a trigger of a user's event. Since multiple transitions can by executed for a certain pair of a state and a user's event, then a non-deterministic transition occurs that may cause an illegal state against the user's intention.

An algorithm was designed specifically for feature interaction detection for rule-based specifications. This algorithm is called **Algorithm EXH** [22]. This Algorithm is composed of two phases. The first phase enumerates all possible reachable states while the second phase detects a pair of rules which non-deterministically conflict with each other by checking each enumerated state. This method takes a lot of time and space since the number of reachable states increase exponentially (state explosion problem) depending on the complexity (number of rules) and

the number of users in the service. For certain services with many users, the state explosion problem is very serious and thus Algorithm EXH cannot be applied.

In order to overcome this problem, Nakamura et al have devised an alternative detection technique called Algorithm Ω which is based on the P-invariant and Petri-Net Model [38]. This method requires no enumeration and can be outlined as follows:

1) Transform the given service specifications to the Petri-Net Model

2) Obtain all pairs of rules which may cause the non-determinism

3) Based on the rules, determine a set of states at which the non-determinism may occur.

4) Check if the determined states are reachable from the initial state (Using the P-invariant and Petri-Net Model).

This method was scrutinized using two experiments. One experiment examines the detection quality and performance while the other experiment examines the scalability and applicability of Algorithm Ω compared to Algorithm EXH. The results of the experiments show that Algorithm Ω attained a high quality of detection, drastic performance improvements (in some cases about 28000 times faster) and good scalability with respect to the number of users.

Our method is based on the assumption that a feature interaction is a property of the feature specification, hence feature interactions must be detected in the requirements specification phase. It is based on the use of formal specifications to represent individual features. We specify features by writing inductive definitions of relations using axioms and rules. As the number of users increase, so do the number of rules and axioms in a relation.

When we compare our approach to the approach described for Algorithm Ω, it seems that the number of states is proportional to the number of users added to a service or feature when Algorithm Ω is applied. When our method is used, the number of valid input sequences belonging to a relation grows exponentially as the number of rules and axioms increase. In terms of performance and scalability, clearly Algorithm Ω is more efficient at detecting feature interactions as the number of users increases.

1.7 Research Contribution

The research contribution in this thesis can be summarized as follows:

- Relational specification of POTS and 5 telecommunication features.

- Validation of these specifications.
- Categorization of the feature interactions found.
- Detection of feature interactions between the specified features.
- Automation of the specification validation process using OTTER
- Automation of detecting feature interaction sequences using OTTER.

Chapter 2

Sets and Relations

Most of this chapter is taken directly from [33]. Here, we present notions of set theory and relations. Relations are presented from two distinct view points: first, as a set of pairs; second as an algebraic structure defined by operations such as union, intersection, inversion, complement and composition. We assume the reader is familiar with predicate logic and discrete mathematics.

2.1 Sets

2.1.1 Set Definition

A *set* is a collection of objects chosen from some universe. The universe is usually understood from the context.

The elementary sets used in this thesis are:

- natural, the set of natural numbers,
- integers, the set of integers,
- boolean, the set {true, false}

2.1.2 Operations on Sets and Notation

Let S and S' be two sets. The following table represents set operation and notations used in this thesis.

Operation	Notation
Union of S and S'	$S \cup S'$
Intersection of S and S'	$S \cap S'$
Complement of S	$\overline{S}$
Difference of S and S'	$S \backslash S'$
Cartesian Product of S by S'	$S \times S'$
Power Set of S	$P(S)$

Table 2.1: Set Operations and Notation

The cartesian product of S by S' is the set of pairs of the form (s, s') such that s belongs to S and s' belongs to S'. The power set of S is the set of all subsets of S.

2.2 Relations

2.2.1 Preliminary Definitions

A binary relation on a set S is a subset of $S \times S$. An element s is said to be an *antecedent* in R while s' is said to be an *image* in R if (s, s') is an element of relation R. The set of images of s by relation R is denoted by $s \bullet R$ and the set of antecedents of element s by relation R is denoted by $R \bullet s$. The set of all the antecedents of relation R is called the *domain* of relation R, and denoted by *dom(R)*. The formal definition is given by:

$dom(R)=\{s|\exists s' : (s, s') \in R\}$.

The set of all images of relation R is called the *range* of relation R and is denoted by *rng(R)*. The formal definition is given by:

$rng(R)=\{s|\exists t : (t, s) \in R\}$.

The following table defines relevant constant relations on a set S and the corresponding notation.

Relation	Definition	Notation
Universal Relation	$S \times S$	$L(S)$
Identity Relation	$\{(s, s')\|s \in S \wedge s' = s\}$	$I(S)$
Diversity Relation	$L(S) \backslash I(S)$	$V(S)$
Empty Relation	empty set	$\phi(S)$

Table 2.2: Constant Relations on a Set S

When the set S is implicit from the context, we may use the symbols L, I, V, and ϕ to denote

the universal, identity, diversity, and empty relations respectively. To maintain homogeneity, the symbol to represent the empty set is $\emptyset$ and the symbol to represent the empty relation is ϕ.

2.3 Operations on Relations

Since relations are sets, they adopt the same operations as those defined on sets. However, the following operations only apply to relations; let S be a set and R and R' denote relations on S.

- **Complement** The *complement* of relation R is the relation denoted $\overline{R}$ and defined by:

  $\overline{R} = L \backslash R.$

- **Inverse** The *inverse* of a relation R is the relation denoted by $\hat{R}$ and defined by:

  $\hat{R} = \{(s, s') | (s', s) \in R\}.$

- **Product** The *product* of relation R by relation R' is the relation denoted by $R \circ R'$ and defined by:

  $R \circ R' = \{(s, s') | \exists t : (s, t) \in R \wedge (t, s') \in R'\}.$

- **Restriction** The *prerestriction* of relation R to subset A of S is the relation denoted by ${}_{A\backslash}R$ and defined by:

  ${}_{A\backslash}R = I(A) \circ R$

The postrestriction of relation R to subset A of S is the relation denoted by $R_{/A}$ and defined by:

$R_{/A} = R \circ I(A).$

Chapter 3

Specifying With Relations

3.1 The Nature of Specifications

A specification can be defined as :

- The act or process of specifying.
- A detailed and precise representation of something.

From a literary viewpoint, the word *specification* refers to both a *process* and to a *product.* From a software engineering viewpoint, there is a crucial distinction between the process and the product. This distinction is a key feature of several software engineering goal structures.

3.1.1 The Product

As a *product*, a specification is a description of requirements that a user imposes on a software product [33]. While requirements potentially deal with several features of the desired product. In the thesis, we restrict our attention to *functional* requirements. Furthermore, while these requirements can be described in different notations, our attention is restricted to formal and mathematically-based representations. The specification as a product serves three primary functions:

- It is an agreement between the user, whose privileges it defines, and the programmer, whose obligations it defines. Hence, the specification is used to check the validity of the delivered product.
- It is the programmer's primary reference document, since it is used by the programmer to construct the proposed product by stepwise transformation of the specification.
- It is part of the user's manual, since it describes the function provided by the software product as well as its usage conventions.

As a *product* the word *specification* is used to describe software systems at arbitrary stages of refinement, ranging from requirements definitions all the way to detailed design. To clearly define what we consider to be a specification, we define a point in the software lifecycle when the system description is a *specification*; this point is defined by the criteria given below.

- **The task at hand.** In the specification phase, the task at hand is to add requirements to existing requirements. In the design phase, the task at hand is to apply decompositions to an existing aggregate of requirements.
- **The emphasis.** In the specification phase, the emphasis is on accumulating information. In the design phase, the emphasis is on refining information descriptions.
- **The source of information.** In the specification phase, the source of new information is the user. In the design phase, the source of new information is the programmer.
- **The kind of data added.** In the specification phase, functional data is added to existing functional data. In the design phase, structural data is added to functional data.
- **Stopping test.** In the specification phase, the stopping test is whether the user cares to add further requirements. In the design phase, the stopping test is whether the programmer must refine further.

Whether a description of a system is being specified or refined, it increases the volume of information. Figure 3.1 represents the evolution of the quantity of information cumulated during the specification phase and the design phase. The quantity of information is represented on the horizontal axis while time is represented on the vertical axis. The horizontal line in the middle of the triangle represents the quantity of information carried by the specification. The area above the line represents an incomplete specification while the area below the line represents more than a specification. Finally, the base of the triangle represents the final step of the design process which is the program itself.

3.1.2 The Process

The traditional model of the software lifecycle is structured along two orthogonal axes: *phases*, which define a chronological structuring of the process and *activities* which define an organizational structuring of the process. We apply the same premise to the specification process and provide that this process includes two phases and two activities. Three parties are involved in this process: the *user group,* the *specifier group,* and the *verification and validation group.* The *user group* is responsible for the application domain; the *specifier group* is responsible for

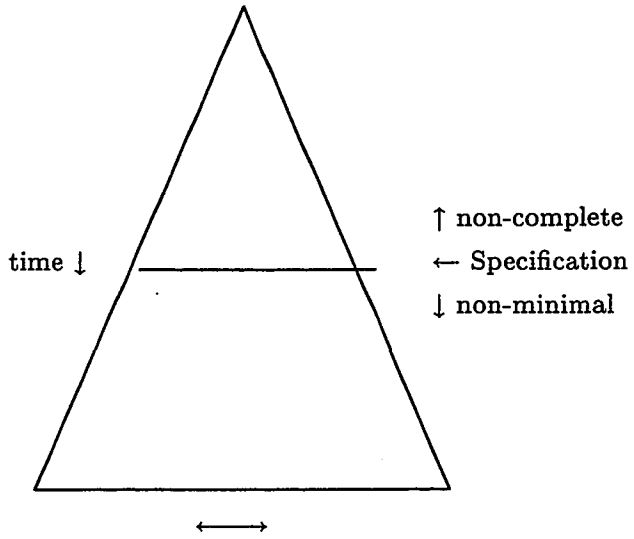

Figure 3.1: The Volume of Information: Specification to Design

the art of specifying, that is, eliciting and recording user requirements; the *verification and validation group* is responsible for validating specifications, that is, eliciting requirements from the user and matching them against the generated specification.

The specification process is comprised of two activities:

- **Specification Generation.** The specifier group along with the user group carry out this activity. It mainly consists of generating the specification from the viewpoint of the user.
- **Specification Validation.** The verification and validation group along with the user group carry out this activity. It consists of generating requirements from the user viewpoint and then using them to ensure the generated specification conforms to them.

The two phases of the specification process are the following:

- *Specification Generation.* In this phase, the specifier group along with the verification and validation group interact with the user group to elicit requirement information. While the former uses this information to derive a specification, the latter uses it to generate properties that the derived specification must conform to.

- *Specification Validation.* In this phase, the verification and validation group checks whether the specification derived by the user group in the previous phase satisfies the properties that were generated. Also in this phase, the specifier group may take corrective action and customize the specification, then deliver its final version.

3.2 Properties of Specifications

To effectively provide its complete functionality, a specification must satisfy a number of properties. In this section, we provide a list of such properties. We clearly distinguish between the properties of the specification *product* and the properties of the specification *process*.

3.2.1 Properties of the Product

Properties of the product are those that are intrinsic to the specification product itself, irrespective of the relation that it bears to the intention of the user. We identify and present three such properties below:

Formality. A specification is said to be formal if it is written in a language whose syntax and semantics are formally defined. This property is essential if the specification product is to play a role as an agreement between the user and the programmer, and its role as the programmer's reference document. It is also significant if the specification is to undergo any automatic processing.

Clarity. A specification is said to be clear if it is easy to read, analyze and understand. Clarity of a specification is vitally important. Programs are read and executed by computers and can be of some use even when they are unclear. However, specifications are read by people and are essentially useless if they are unclear.

Abstraction. A specification is said to be abstract if it contains no structural or implementation information. A specification is a description of the user's problem rather than a description of a possible solution to that problem. It must express *what* the user wants rather than *how* to realize what the user wants.

3.2.2 Properties of the Process

Properties of the process are those that cannot be solely determined in view of the product, but must be judged by matching the specification product against the user intentions. We identify and present two such properties below:

Completeness. A specification is complete if it captures all the user requirements. Completeness cannot be proved rather it can be progressively established between requirements information elicited by the specifier group and requirements information elicited by the verification and validation group.

Minimality. Not only is a specification expected to capture all the user requirements, it is also expected to capture nothing but the user requirements. A specification is said to be *minimal* if it contains only those requirements that have been established by the user.

3.3 Relational Specifications

Fundamentally, programs read input data from either a device or an input file, process the information and return results onto a device or an output file. The *input space* of program is defined to be the set of all inputs a program can read in. The *output space* of a program is the set of all outputs a program can return. The *internal space* of a program is defined as the set of values the program's variables may hold during its execution. When specifying a system we are merely concerned with the state transformation that takes place during the execution of a program rather than the mappings from the program's input space, to the internal space, finally to the output space. Having established this notion, we can say that we have defined a specification when we have given:

- *A space*, say S, typically defined by a set declaration.
- *A relation* on space S, say R, typically defined as a set of pair (s, s'), such that some property holds between s and s'.

This specification, represented by the pair (S, R), dictates requirements on a program whose space is S. The pair (s, s') is in R if and only if s is a possible initial state and s' is a correct corresponding final state for s. For any initial state s, more than one final state s' may exist in $s \bullet R$. R is non-deterministic.

Specifications may be represented solely by a relation R only when the space of S of a specification (S, R) is implicit from the context.

Consider the space S:

$S =$ **set**

$a, b, c :$ **natural**

end.

Example 1. Write specifications for the following user requirements:

a) Specify a program that places in a the smallest value among a, b, c.

$R = \{(s, s') | a(s)' \leq a(s) \wedge a(s') \leq b(s) \wedge a(s') \leq c(s) \wedge (a(s') = a(s) \vee a(s') = b(s) \vee a(s') = c(s))\}$.

The first line says $a(s')$ is less than or equal to a, b, c. The second line says it is one of them.

b) Specify a program that places in a the smallest value among a, b, c while preserving b and c.

$R = \{(s, s') | a(s)' \leq a(s) \wedge a(s') \leq b(s) \wedge a(s') \leq c(s) \wedge (a(s') = a(s) \vee a(s') = b(s) \vee a(s') = c(s)) \wedge b(s') = b(s) \wedge c(s') = c(s)\}$.

The first line says $a(s')$ is less than or equal to a, b, c. The second line says it is one of them. The third line says b and c are preserved.

c) Preserve the sum of a and b while increasing c.

$R = \{(s, s') | a(s) + b(s) = a(s') + b(s') \wedge c(s') > c(s)\}$.

Example 2. Preserve the sum of x and y while increasing z. For each of the following specifications, and tell whether it is complete and (if complete) whether it is minimal.

a) $R = \{(s, s') | x(s) + y(s) = x(s') + y(s') \wedge z(s') \geq z(s)\}$

not complete

b) $R = \{(s, s') | x(s) + y(s) = x(s') + y(s') \wedge z(s') > z(s)\}$

complete and minimal

c) $R = \{(s, s') | x(s) = x(s') \wedge y(s) = y(s') \wedge z(s') > z(s)\}$

complete but not minimal

d) $R = \{(s, s') | x(s) = x(s') + 1 \wedge y(s) = y(s') - 1 \wedge z(s') > z(s)\}$

complete but not minimal

e) $R = \{(s, s') | x(s) + y(s) = x(s') + y(s') \wedge z(s') = z(s) + 1\}$

complete but not minimal

Example 3. Given an array $a : array[1..N]$ of type *real*, and variable x of type real, write

specifications for the following user requirements:

a) Place in x an arbitrary value of array a.

$R = (s, s')|(\exists h : 1 \leq h \leq N : x(s') = a(s)[h])$.

b) Place in x a value that is greater than all the cells in a (note: this is a non-deterministic specification; if $a(s) = (3, 1, 7, 5, 2, 8, 2)$ then $x(s')$ may be equal to 8, but may also be equal to 20).

$R = \{(s, s')|(\forall h : 1 \leq h \leq N : x(s') \geq a(s)[h])\}$.

c) Place in x the largest value of a.

$R = \{(s, s')|(\forall h : 1 \leq h \leq N : x(s') \geq a(s)[h]) \wedge (\exists h : 1 \leq h \leq N : x(s') = a(s)[h])\}$.

First line: $x(s')$ is greater than or equal to all the cells of a.

Second line: $x(s')$ is equal to one of them.

3.4 Refinement Ordering

From the previous section, we established that specifications can be represented by relations. On the other hand, we can compare specifications according to the amount of input-output information they hold. The following definition emphasizes this.

1 Definition.: Relation R is said to be *more-defined* than relation R', denoted $R \geq R'$, if and only if $R'L \subseteq RL \wedge R'L \cap R \subseteq R'$.

We may say that R refines R' if and only if R is a refinement of R'. We can interpret this definition in relation theoretic terms as:

$$dom(R') \subseteq dom(R) \wedge (\forall s \in dom(R) : s \bullet R \subseteq s \bullet R')$$

A relation is said to be *more-defined* if the following conditions hold:

i) It has a larger domain (knows more input sets).

ii) It has smaller image sets (more precise in its assignment of outputs to inputs).

There are two cases when the more-defined ordering takes a simple form: If R and R' have the

same domain then R is more-defined than R' if and only if $R \subseteq R'$. If R and R' are deterministic then R is more-defined than R' if and only if $R' \subseteq R$.

3.4.1 Ordering Properties

Trivially, R is *more-defined* than R. The relation *more-defined* is *reflexive*, however, we can establish the *antisymmetry* and the *transitivity* of the more-defined relation using the antisymmetry and the transitivity of the inclusion relation (subset). Since the more-defined relation is reflexive, antisymmetric, and transitive we conclude that is a partial ordering. Hence, we may refer to this as the refinement ordering.

3.4.2 Lattice Properties

Given two relations R and R' on S, we are interested in whether they have a least upper bound, $R \sqcup R'$ and a greatest lower bound, $R \sqcap R'$. The next proposition is given without proof:

Proposition 3.1: If R and R' satisfy the condition $(R \cap R')L = RL \cap R'L$, then R and R' have a unique least upper bound, which is $R \sqcup R' = R \cap \overline{R'L} \cup R' \cap \overline{RL} \cup R \cap R'$

Interpreting condition (cs), we determine that two relations R and R' have a least upper bound if for those elements that belong to both *dom(R)* and *dom(R')*, the set of images by R and R' have at least one element in common; if $s \bullet R$ and $s \bullet R'$ had no element in common for some element s, this would mean that relations R and R' contradict each other. Condition (cs) means R and R' are not mutually inconsistent. This is referred to as the consistency condition, whence its abbreviation.

Proposition 3.1 gives a sufficient condition for the existence of a least upper bound. The proposition below, presented without proof, provides that this condition is also necessary.

Proposition 3.2: If R and R' have a least upper bound, then they satisfy the consistency condition.

From Proposition 3.2, we find that the least upper bound of two relations R and R' represents the total input-output information contained in them; these may be *summed* if they are mutually consistent (hence the *consistency condition*, which must be satisfied for the least upper bound to exist).

Proposition 3.3 Any pair of relations R and R' have a greatest lower bound, which is given by the expression:

$$R \sqcap R' = RL \cap R'L \cap (R \cup R')$$

We can interpret the greatest lower bound of two relations R and R' as the duplicated information of R and R'; that is, the information they have in common. The least upper bound and the greatest lower bound take a fundamental form under two conditions which are presented in the form of corollaries.

Corollary 3.1: If R and R' satisfy the condition $RL \cap R'L \cap R = RL \cap R'L \cap R'$, then R and R' have a least upper bound which is $R \sqcap R' = R \cap R'$

Corollary 3.2: If R and R' satisfy the condition $RL = R'L = (R \cap R')$ then R and R' have a least upper bound which is

$R \sqcup R' = R \cup R'$.

Also, under the same condition,

$R \sqcap R' = R \cap R'$

Example: Let S be the space defined by:

$$x, y, z : \textbf{integer}$$

and let R and R' be the following relations:

$$R = \{(s, s') | x(s) \geq y(s) \wedge z(s') = x(s)\}$$

$$R' = \{(s, s') | x(s) \leq y(s) \wedge z(s') = y(s)\}$$

Then the least upper bound of R and R' is:

$$= R \cup R'$$

$$= \{(s, s') | x(s) \geq y(s) \wedge z(s') = x(s)\} \cup \{(s, s') | x(s) \leq y(s) \wedge z(s') = y(s)\}$$

$$= \{(s, s') | z(s) = max(x(s), y(s))\}$$

This relation captures the total information over R and R'

The greatest lower bound of R and R' is:

$$= R \cap R'$$

$= \{(s, s') | x(s) = y(s) \wedge z(s') = x(s)\}$

Both R and R' account for the case when $x(s) = y(s)$. This is provided in R when $x(s) \geq y(s)$ and in R' when $x(s) \leq y(s)$. Also, both provide the same output condition, namely $z(s') = x(s)(= y(s))$.

2 Definition. : The demonic join operator is defined if and only if the following condition called the *consistency condition* is satisfied:

$cs(R, R') \hat{=} RL \cap R'L = (R \cap R')L$

This means that R and R' are mutually consistent. This implies that they do not contradict each other. Two relations contradict each other when they provide disjoint sets of outputs for some input where they are both defined.

The symbol "$\hat{=}$" represents a definition or an axiom while "$=$" represents a property or a theorem which is deducible or provable from the definitions or axioms.

Whenever the consistency condition is satisfied, the join is defined by:

$R \sqcup R' \hat{=} (R \cap \overline{R'L}) \cup (R \cap R') \cup (\overline{RL} \cap R')$

This means that the specification that carries all the requirements of R and R' and nothing more. In effect, the join operator is simply the adding up of the specifications.

Chapter 4

A Specification Methodology

4.1 Proving Completeness

Let R be a specification, and let V be the relation derived to capture a completeness property. We pose the following definition.

3 Definition. Specification R is said to be *complete with respect to property* V if and only if R is more-defined than V.

The validity of this definition stems from the way V is derived (to capture a property which R is expected to have) and the definition of relation *more-defined* (all the information of V is captured in R). We illustrate this definition with three examples, of increasing complexity.

Example: A Closed form Specification. We let space S be defined by three variables, say $< a, b, c >$, of type integer, and let the user requirement be: *permute the values of variables a, b and c in ascending order.* We assume that, from this text the specifier (group) has generated the following specification:

$$\begin{aligned} R = \{(s, s') \mid\ & a(s') < b(s') < c(s') \wedge \\ & max(s) = max(s') \wedge \\ & min(s) = min(s') \wedge \\ & sum(s) = sum(s')\}. \end{aligned}$$

Also, from this same requirements text (not from the specification) the verification and validation group has generated the following property:

> *The specification must imply that if the input is sorted then the output is identical to the input.*

Notice how arbitrarily partial, and arbitrarily weak this property is. This property can be captured by the following relation.

$V = \{(s, s') \mid a(s) < b(s) < c(s) \wedge s' = s\}$.

In order to establish the completeness of R with respect to V, we must prove the following clauses:

$dom(V) \subseteq dom(R)$,

$\forall s \in dom(V), s.R \subseteq s.V$.

Because $dom(R) = S$, we only have to prove the second clause, which we rewrite as

$s \in dom(V) \wedge (s, s') \in R \Rightarrow (s, s') \in V$.

We write the left hand side as:

$$
\begin{aligned}
& a(s) < b(s) < c(s) \wedge a(s') < b(s') < c(s') \wedge \\
& max(s') = max(s) \wedge min(s') = min(s) \wedge \\
& sum(s) = sum(s') \\
\Rightarrow\ & a(s') = a(s) \wedge c(s') = c(s) \wedge b(s') = b(s) \wedge \\
& a(s) < b(s) < c(s) \\
\Rightarrow\ & s' = s \wedge a(s) < b(s) < c(s).
\end{aligned}
$$

Hence $(s, s') \in V$.

Example: Axiomatic Specification. We consider the specification of the *stack* data type, of which we have given an axiom and a rule; the reader can imagine what the full specification looks like,(its full text is given in [6]). Also, we consider the following completeness property:

> *the specification must indicate that when the same element is push-ed twice, there are indeed two copies of it in store (rather than one as would be the case for a set-like data type).*

We refer to this property by the name *listfeature*. The relation that we generate to reflect this property is,

$$
\begin{aligned}
listfeature = \{(Q, t) \mid \exists Q' :\ & \\
& Q = Q'.init.push(t).push(t).pop.top\}.
\end{aligned}
$$

If the relation defined by specification *stack* can be established to be more-defined than V, then

we can claim that *stack* is complete with respect to V.

Example: Axiomatic Specification. We consider yet another example of checking the completeness of the stack specification with respect to some property. We assume that after inspecting the user requirements, the verification and validation group generates the following completeness property:

The stack specification must provide that the output for the input sequence

$$Q.init.pop.push(a).push(T).top.top$$

is the value of T, where T is a variable.

Among the errors that this property can catch, we mention: if the specifier did not consider the pop operation void (because it follows init); if the specifier made a pop cancel a push that follows it (rather than precede it); if the specifier did not understand that top operations do not change the state; if the specifier did not understand (or did not capture properly) the LIFO property of the stack.

We refer to this property by the name *poptop*. This property can be written under the form of the following relation:

$$poptop = \{(q,y) \mid y = T \wedge \exists Q :$$

$$q = Q.init.pop.push(a).push(T).top.top\}.$$

The completeness of the stack specification with respect to this property can be established by proving that the relation defined by specification *stack* is more-defined than relation *poptop*.

4.2 Proving Minimality

Let R be a specification, and let W be the relation derived to capture a minimality property. We pose the following definition [7].

4 Definition. Specification R is said to be *minimal with respect to property W* if and only if R is not more-defined than W.

The validity of this definition stems from the way W is derived (to capture a property which R is expected *not* to have) and the definition of relation *more-defined* (the information of W must not be captured in R).

For the sake of readability, we illustrate this definition on a simple specification from S to S; in a second example, we consider a specification from X^* to Y.

Example: Closed Form Specification. As an illustrative example, we take the following user requirements, defined on space S made up of three natural variables, say a, b and c: *preserve the sum of a and b and do not decrease c.*

In view of this user requirement, the specifier generates the validation group may think:

> *the specifier may have thought that one way to preserve the sum of a and b is to preserve variables a and b individually but the user does not necessarily require that a and b be preserved individually.*

In order to catch specifiers who may have over specified the preservation of $a+b$, the verification and validation group proposes:

$$W = \{(s, s') \mid a(s') = a(s) \wedge b(s') = b(s)\}.$$

In order to check whether R is minimal with respect to W, we investigate whether R is *not* more-defined than W. We find that R and W have the same domain (they are both total), and that for all s in the domain of W (which is all of S),

$$s.W = \{s' \mid a(s') = a(s) \wedge b(s') = b(s)\},$$
$$s.R = \{s' \mid a(s') + b(s') = a(s) + b(s) \wedge c(s') > c(s)\}.$$

Clearly, it is not the case that $s.R$ is a subset of $s.W$; hence R is indeed minimal with respect to W. Informally, this means that the author of specification R is not guilty of the over specification which is described in W. We leave it to the reader to check that, had the specifier defined the specification as:

$$R = \{(s, s') \mid a(s') = a(s) \wedge b(s') = b(s) \wedge c(s') > c(s)\}$$

the verification and validation would have found it to be non minimal with respect to W; which is a valid diagnosis. □

Example: Axiomatic Specification. We consider the specification of the stack, given in the previous section, and we consider the following minimality property:

> *the specifier may have thought that when a* pop *operation is performed on an empty stack, it causes an error message when in fact the user has not asked for this specific message.*

We refer to this property by the name *silent,* as it means that the data type must remain silent (rather than declare an error) when a pop is invoked on an empty stack. The relation that we generate in light of this property is:

$$silent = \{(Q, y) \mid \exists Q' : Q = Q'.init.pop \wedge y = error\}.$$

If the relation defined by specification *stack* is more-defined than W, then *stack* is not minimal with respect to W. In the next section, we show how the axiomatic specification of *stack* can be matched against such a property to establish minimality in an automated fashion.

4.3 Complete and Minimal Specifications

By deriving a number of completeness properties (V) and minimality properties (W), the verification and validation group provides a band within which candidate specifications may lie. This band grows narrower and narrower as the verification and validation group provides more and more properties. Recent studies we have conducted on the lattice properties of the ordering *more-defined* [6] provide for the existence of a least upper bound of two specifications under some conditions which are normally met by completeness properties; it is a well known property of lattices that a specification R is greater than $V_1, V_2, ..V_n$ if and only if it is greater than the least upper bound of $\{V_1, V_2, ..V_n\}$. Hence, given the completeness properties $V_1, V_2, ..V_n$, and the minimality properties $W_1, W_2, ..W_n$, candidate specifications R are allowed to lie within the range indicated in figure 4.1, where R is shown.

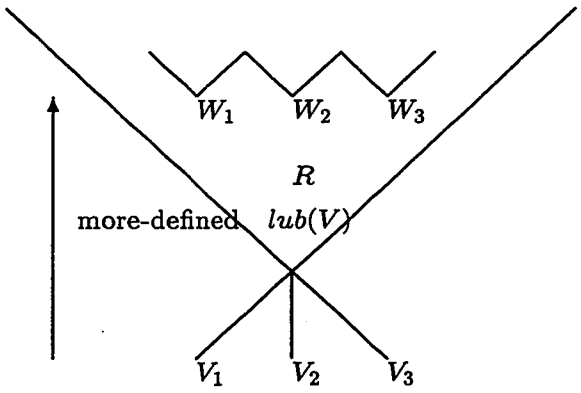

Figure 4.1: The Range of Complete and Minimal Specifications

Chapter 5

Specifying Dynamic Software Systems

In the previous chapter, we defined a specification as a relation between inputs and outputs. This definition does not allow us to describe the external behavior of a software system because the output corresponding to any given input cannot always be determined by mere inspection of the input, but may have to be determined by the history of previous inputs. Hence the specification of a software system can be described by a relation from the sequence of inputs to the outputs. Whence the formal method described in subsequent sections.

5.1 Specification Model

The specification of a software system may be described by a relation from the sequences of inputs to outputs. We can specify a software system by using three components:

1) Input Space I, defined as the set of all possible inputs of the system.

2) Output Space O, defined as the set of all possible outputs of the system.

3) Input-Output Relation R, defined as the relation from the set I^* of sequences on I to a set O, where I^* is the set of sequences over I including the empty sequence.

As an example, consider the relational specification of a minimum spanning tree (MST) in an n-node network. We make the assumption that there exists at least one arc between every pair of nodes in the network. The following is the set of requirements:

i) Initially, there are no arcs or nodes in the network.

ii) **add_node** adds a node in the network.

iii) **del_node** deletes a node from the network.

iv) **num_node** returns the number of nodes in the network.

v) **num_arc** returns the number of arcs in the network.

The input set for the MST is:

$$I \hat{=} \{ \textbf{add\_node}, \textbf{del\_node}, \textbf{num\_node}, \textbf{num\_arc} \}$$

The output set for the MST is:

$$O \hat{=} \textbf{Integer} \cup \{\tau\}$$

The $\hat{=}$ can be interpreted as "is defined as". The symbol τ is used when there is no visible output produced by a set of input sequences. Next, some examples of elements of relation R specifying the MST are provided. The dot (".") is used to denote the concatenation of input sequences, while $s \lhd R \rhd s'$ denotes the pair (s, s') is an element of relation R.

a) **num_node** $\lhd R \rhd 0$

b) **num_arc** $\lhd R \rhd 0$

c) **add_node** $\lhd R \rhd \tau$

d) **del_node** $\lhd R \rhd \tau$

e) **add_node.add_node.num_arc** $\lhd R \rhd 1$

f) **add_node.del_node.add_node .add_node.add_node.num_node** $\lhd R \rhd 3$

Pairs *a*) and *b*) satisfy requirement *i*) when there are no nodes or arcs in the network. Pairs *c*) and *d*) indicate that there is no visible output after receiving the input **add_node** or **del_node**. Pair *e*) represents the case when there are two nodes present in the network. If the input **num_arc** is encountered after this, we have 1 as the corresponding output. Finally, in pair *f*), the first two elements of the input sequence simply cancel out each other. The remainder of the input sequence adds three nodes to the network and yields 3 as an output when **num_node** is encountered.

The complete specification of the above example cannot be defined by generating all enumerations of its input-output pairs. This is impossible and impractical since there exists an infinite number of input-output pairs. Instead, specifications are constructed inductively using

an axiomatic representation.

5.2 Axiomatic Representation of Specifications

In the previous section, we discussed why we cannot satisfy requirements by enumerating inputs and outputs over the specification space. The solution is to use an axiomatic approach to define a specification R. The language of first order logic is employed with the usual connectives. The connectives have the following binding order from highest to lowest:

$$\neg, \wedge, \vee, \{\Leftarrow, \Rightarrow\}, \Leftrightarrow, \{\forall, \exists, \wedge_j, \vee_j\}$$

A tuple is denoted by (,...,) and {,...,} is used to denote a set. Variables x and x' range over the input space I^*. Upper case letters are relational constants which denote a specification (for example CW for call waiting). By using this language, we are able to construct an inductive definition of a relation. The set of axioms that compose the specification take the form $(\wedge_{i=1}^{n} A_i) \Rightarrow B$, n is a natural number and A_i and B are predicate variables. For the sake of clarity and readability, axioms are represented as:

$$\frac{A_1 \wedge \ldots \wedge A_n}{B}$$

To illustrate the use of this method, an inductive definition of the MST specification is given below.

(mst-1): $\dfrac{}{\mathsf{num\_node}\triangleleft MST \triangleright 0}$

(mst-2): $\dfrac{}{\mathsf{num\_arc}\triangleleft MST \triangleright 0}$

(mst-3): $\dfrac{}{x.\mathsf{add\_node}\triangleleft MST \triangleright \tau}$

(mst-4): $\dfrac{}{x.\mathsf{del\_node}\triangleleft MST \triangleright \tau}$

(mst-5): $\dfrac{x.\mathsf{num\_node}\triangleleft MST \triangleright y}{x.\mathsf{add\_node.num\_node}\triangleleft MST \triangleright y+1}$

(mst-6): $\dfrac{x.\mathsf{num\_node}\triangleleft MST \triangleright y}{x.\mathsf{del\_node.num\_node}\triangleleft MST \triangleright y-1}$

(mst-7): $\dfrac{x.\mathsf{num\_node}\triangleleft MST \triangleright y}{x.\mathsf{num\_arc}\triangleleft MST \triangleright y-1}$

(mst-8): $\dfrac{x.x' \lhd MST \rhd y}{x.\mathsf{num\_node}.x' \lhd MST \rhd y}$

(mst-9): $\dfrac{x.x' \lhd MST \rhd y}{x.\mathsf{num\_arc}.x' \lhd MST \rhd y}$

(mst-10): $\dfrac{x.\mathsf{num\_arc} \lhd MST \rhd y}{x.\mathsf{add\_node.num\_arc} \lhd MST \rhd y+1}$

(mst-11): $\dfrac{x.\mathsf{num\_arc} \lhd MST \rhd y}{x.\mathsf{del\_node.num\_arc} \lhd MST \rhd y-1}$

Axioms mst-1 to mst-4 are trivial. Axiom mst-5 describes that **add_node** will increment the number of nodes in the network by 1. Similarly, axiom mst-6 describes that **del_node** will decrement the number of nodes in the network by 1. Axiom mst-7 describes that the number of arcs in an n-node network for an MST is n - 1. Axiom mst-8 and mst-9 show that **num_node** and **num_arc** do not change the number of nodes or arcs in the network respectively. Axiom mst-10 describes that adding a node increases that number of arcs in the network by one. Similarly mst-11 describes that deleting a node decreases the number of arcs by 1. These axioms can be used to prove whether a pair (s, s') is in a relation R. As an example, consider the input-output pair below:

sequence : **add_node.num_arc.add_node.num_node.add_node.num_arc** $\lhd MST \rhd$ 2

Proof :

add_node.num_arc.add_node.num_node.add_node.num_arc $\lhd MST \rhd$ 2
$\Leftarrow$ (mst-7)
add_node.num_arc.add_node.num_node.add_node.num_node $\lhd MST \rhd$ 3
$\Leftarrow$ (mst-5)
add_node.num_arc.add_node.num_node.num_node $\lhd MST \rhd$ 2
$\Leftarrow$ (mst-8)
add_node.num_arc.add_node.num_node $\lhd MST \rhd$ 2
$\Leftarrow$ (mst-5)
add_node.num_arc.num_node $\lhd MST \rhd$ 1
$\Leftarrow$ (mst-8)
add_node.num_arc $\lhd MST \rhd$ 1
$\Leftarrow$ (mst-10)
num_arc $\lhd MST \rhd$ 0

$\Leftarrow$ (mst-2)

true

5.3 Tabular Notation and Axiomatic Representation

Previously, we presented each axiom belonging to a specific relation using plain mathematical notation of premise over conclusion. As the number of axioms increase in a relation, it becomes more and more tedious to read specifications. For this reason, we use a tabular format to represent axioms.

5.3.1 Cumulative Tables

Table 5.1 represents the general format used for a *cumulative* table. Each row in the table represents one axiom.

No	Premise	Input	Output
A-1	*premise*1	(input1)	{(output1)}
A-2	*premise*2	(input2)	{(output2)}
A-3	*premise*3	(input3)	{(output3)}
A-4	*premise*4	(input4)	{(output4)}

Table 5.1: Cumulative Table: General Format

Let p_n, s_n and s'_n be respectively the premise, the input and the output of line n.

Then the rule associated with line n is:

$$\frac{p_1 \wedge \ldots \wedge p_n}{s_1.\ldots.s_n \triangleleft REL \triangleright s'_n} .$$

For instance, the axioms associated with rows A-2 and A-4 are respectively:

$$\text{(REL-A-2):} \quad \frac{premise2}{(\text{input1}).(\text{input2}) \triangleleft REL \triangleright \{(\text{output2})\}} ,$$

$$\text{(REL-A-4):} \quad \frac{premise4}{(\text{input1}).(\text{input2}).(\text{input3}).(\text{input4}) \triangleleft REL \triangleright \{(\text{input4})\}} .$$

The tabular format presented avoids the unnecessary repetition of the premises and the input sequences from axiom to axiom. Also, it provides a quick and clear visualization of the behavior for prototypical input sequences.

5.3.2 Non-Cumulative Tables

Another type of table used for representing axioms is a *non-cumulative* table. In this table, each row is independent of all other rows.

No	Premise	Input	Output
A-5	*premise*5	(input5)	{(output5)}
A-6	*premise*6	(input6)	{(output6)}
A-7	*premise*7	(input7)	{(output7)}
A-8	*premise*8	(input8)	{(output8)}

Table 5.2: Non-Cumulative Table: General Format

The axiom associated with row n is given by:

$$\frac{p_n}{s_n \triangleleft POTS \triangleright s'_n} \ .$$

For instance, the axiom associated to line A-8 is:

$$\text{(REL-A-8):} \ \frac{premise8}{\text{(input8)} \triangleleft REL \triangleright \{\text{(output8)}\}} \ .$$

5.3.3 Representation of Rules

Axioms provide the output for basic input sequences and for complex input sequences, *rules* are used. *Reduction* and *swap* rules are presented in the two subsequent sections.

5.3.3.1 Reduction Rules

Reduction rules are used to eliminate symbols from the input sequence.

No	Condition	Prefix	Discard	Postfix
R-1	*premise*1	x	(discard1)	(postfix1)
R-2	*premise*2	x	(discard2)	(postfix2)
R-3	*premise*3	x	(discard3)	(postfix3)
R-4	*premise*4	x	(discard4)	(postfix4)

Table 5.3: Reduction Rules: General Format

Table 5.3 is a non-cumulative table where each row represents a reduction rule. Let p, *pre*, *discard*, *post* be respectively the condition, the prefix, the discard and the postfix of row n, and let y be a variable with respect to row n.

Then the associated axiom is:

$$\frac{p \wedge pre.post \triangleleft REL \triangleright y}{pre.discard.post \triangleleft REL \triangleright y} .$$

For instance, the rule associated with rows R-2 and R-4 are respectively:

$$\text{(REL-R-2):} \quad \frac{premise2 \wedge x.(\text{postfix2}) \triangleleft REL \triangleright y}{x.(\text{discard2})(\text{postfix2}) \triangleleft REL \triangleright y} ,$$

$$\text{(REL-R-4):} \quad \frac{premise4 \wedge x.(\text{postfix4}) \triangleleft REL \triangleright y}{x.(\text{discard4})(\text{postfix4}) \triangleleft REL \triangleright y} .$$

5.3.3.2 Swap Rules

Previously, reduction rules were used to reduce complex rules into axioms. Interestingly enough, not all complex sequences can be reduced by applying these reductions rule. By applying swap rules, a complex sequence can be transformed into an equivalent sequence. Swap rules are used for swapping input elements with each other until a reduction rule may be applied.

No	Condition	Input 1	Input 2
S-1	*premise*1	(input1)	(input2)
S-2	*premise*2	(input3)	(input4)
S-3	*premise*3	(input5)	(input6)
S-4	*premise*4	(input7)	(input8)

Table 5.4: Swap Rules: General Format

Again, swap tables are non-cumulative just like reduction tables. Two rules are associated with a line n in a swap table in the following manner: let p, s_1, s_2 be respectively the condition, input element 1, and input element 2 of line n, and let x, x', y be three variables; then the associated rules are:

$$\frac{p \wedge x' \neq \varepsilon \wedge x.s_1.s_2.x' \triangleleft REL \triangleright y}{x.s_2.s_1.x' \triangleleft REL \triangleright y} \quad \frac{p \wedge x' \neq \varepsilon \wedge x.s_2.s_1.x' \triangleleft REL \triangleright y}{x.s_1.s_2.x' \triangleleft REL \triangleright y} .$$

For instance, the two rules associated with row R-3 are respectively:

$$\text{(REL-S-3 - 1):} \quad \frac{premise3 \wedge x' \neq \varepsilon \wedge x.(\text{input5}).(\text{input6}).x' \triangleleft REL \triangleright y}{x.(\text{input6}).(\text{input5}).x' \triangleleft REL \triangleright y} ,$$

$$\text{(REL-S-3 - 2):} \quad \frac{premise3 \wedge x' \neq \varepsilon \wedge x.(\text{input6}).(\text{input5}).x' \triangleleft REL \triangleright y}{x.(\text{input5}).(\text{input6}).x' \triangleleft REL \triangleright y} .$$

5.4 Specification Validation

Once the axioms and rules have been presented for a relation, we must validate our specifications. This will ensure whether or not our specifications make sense. To establish this, we must prove that the axioms and rules capture all the user requirements (ie. proving completeness) and the axioms and rules capture nothing but the user requirements (ie. proving minimality). Since we want to focus more on the feature interaction problem, we will only check for completeness and not for minimality.

Recall from section 4.1, a specification R is *complete with respect to property* V if and only if R is *more-defined* than V. This amounts to proving two clauses:

- $dom(V) \subseteq dom(R)$
- $\forall s \in dom(V), s.R \subseteq s.V$

Rather than deriving a completeness property V and proving that R is more-defined than V, we present a theorem V of the form:

$$InputSequence \triangleleft R \triangleright Output$$

and then by using the axioms and rules of the relation, we prove that this is a theorem. This amounts to proving that R is valid with respect to V where V is defined as:

$$V = \{(h, y)\}.$$

where h is an input sequence and y is the corresponding output. Such a property V is a singleton since it contains a single pair. An example of a property V for the MST specification is presented below:

add_node.num_arc.add_node.num_node.add_node.num_node $\triangleleft MST \triangleright$ 3

Proof :

add_node.num_arc.add_node.num_node.add_node.num_node $\triangleleft MST \triangleright$ 3
$\Leftarrow$ (mst-5)
add_node.num_arc.add_node.num_node.num_node $\triangleleft MST \triangleright$ 2
$\Leftarrow$ (mst-8)
add_node.num_arc.add_node.num_node $\triangleleft MST \triangleright$ 2
$\Leftarrow$ (mst-5)
add_node.num_arc.num_node $\triangleleft MST \triangleright$ 1

$\Leftarrow$ (mst-9)

add_node.num_node $\triangleleft MST \triangleright$ 1

$\Leftarrow$ (mst-5)

num_node $\triangleleft MST \triangleright$ 0

$\Leftarrow$ (mst-1)

true

Part II

Specification and Feature Interaction Detection

Chapter 6

Specifying Telecommunication Services

In this chapter, we provide a complete specification for POTS and telecommunication features. The technique introduced has been used by [18] however, there are slight differences. In [18], each feature has its own input space and the input space strictly ranges over its respective feature. In our framework, we define one input space I, that ranges over the entire set of features. In addition, for each feature, we define an independent input space $I_{FEATURE}$ which is a subset of I. This approach helps us discard unrelated input sequences when validating feature specifications. This notion will become clearer in the succeeding sections. In this chapter, we also provide a complete specification for five telecommunication features as well as POTS. These features are three way calling (TWC), call waiting (CW), originating call screening (OCS), abbreviated dialing (ABD) and call forwarding (CF).

Axioms and rules are presented in a tabular format. This presentation makes the specification simpler to read. Axiom tables contain a *Premise*, an *Input* and an *Output* column. The premise states the condition that must be satisfied in order for the corresponding relation between the input and output to be true. Frequently, we use the premise $unique(n_1, n_2, n_3)$ which implies:

$n_1 \neq n_2$ and

$n_1 \neq n_3$ and

$n_2 \neq n_3$

Similarly $unique(n_1, n_2, n_3, n_4)$ implies:

$n_1 \neq n_2$ and

$n_1 \neq n_3$ and

$n_1 \neq n_4$ and

$n_2 \neq n_3$ and

$n_2 \neq n_4$ and

$n_3 \neq n_4$

We begin by presenting a relational specification of POTS. The system is comprised of communication switches and a communication network. Each telephone switch is viewed as a black box. Inputs to the system are sent and outputs are received by telephone units. There is no limit as to the number of users or the number of connections in this system. When the system receives an input, an output is immediately produced before the system receives the next input.

6.1 The Specification Space

An input $(\mathsf{OffHook}, n)$ where n represents the telephone number, is used to uniquely identify the calling party. This input is sent to the switch whenever the user lifts the handset. The identification number, which uniquely identifies a telephone is the same as the telephone number. Input (OnHook, n) is sent to the switch when user n hangs up the telephone. All users must be initialized in the system. The first (OnHook, n) informs the switch that user n is capable of receiving a call. Therefore an input of (OnHook, n_i) must be entered for every i user. This is referred to as the initialization step and is defined as (Init). This input will appear in every rule and it initializes all users in the system. Input $(\mathsf{Dial}, n_1, n_2)$ is sent when user n_1 dials the phone number of user n_2. The input $(\mathsf{FlashHook}, n)$ is used to switch between circuits. This input is used for the TWC and CW features which will be described in more detail later. Other inputs that contain the substring "Act" and "Deact" simply activate and deactivate the corresponding feature respectively. For example, (CFAct, n) activates the call forward feature for user n while $(\mathsf{CFDeact}, n)$ deactivates the same feature for user n.

The following is the complete input space for the telephone system:

$$
\begin{aligned}
I \triangleq\ & \{\mathsf{Init}\} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{FlashHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{TWCAct}\} \times \mathbf{PhoneId} \cup
\end{aligned}
$$

$\{\text{TWCDeact}\} \times \mathbf{PhoneId} \cup$

$\{\text{CWAct}\} \times \mathbf{PhoneId} \cup$

$\{\text{CWDeact}\} \times \mathbf{PhoneId} \cup$

$\{\text{ABDAct}\} \times \mathbf{PhoneId} \cup$

$\{\text{ABDDeact}\} \times \mathbf{PhoneId} \cup$

$\{\text{ABDAdd}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup$

$\{\text{ABDDel}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup$

$\{\text{OCSAct}\} \times \mathbf{PhoneId} \cup$

$\{\text{OCSDeact}\} \times \mathbf{PhoneId} \cup$

$\{\text{OCSAdd}\}\mathbf{PhoneId} \times \mathbf{PhoneId} \cup$

$\{\text{OCSDel}\}\mathbf{PhoneId} \times \mathbf{PhoneId} \cup$

$\{\text{CFAct}\}\mathbf{PhoneId} \cup$

$\{\text{CFDeact}\}\mathbf{PhoneId} \cup$

$\{\text{CFEna}\}\mathbf{PhoneId} \times \mathbf{PhoneId} \cup$

$\{\text{CFDis}\}\mathbf{PhoneId} \times \mathbf{PhoneId}$

The complete output space of the telephone system is defined as follows:

$$
\begin{aligned}
O \triangleq\ & \wp(\{\text{DialTone}\} \times \mathbf{PhoneId} \cup \\
& \{\text{Ring}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\text{BusyTone}\} \times \mathbf{PhoneId} \cup \\
& \{\text{RingTone}\} \times \mathbf{PhoneId} \cup \\
& \{\text{ErrorTone}\} \times \mathbf{PhoneId} \cup \\
& \{\text{Conn}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\text{Disc}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\text{Hold}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\_\})
\end{aligned}
$$

Typically, there exists one output for each input sequence. However for some features such as TWC and CW, there are multiple outputs for some input sequences. The $\wp$ suggests that the output for a given input sequence may be a subset of the output space including the arbitrary output denoted (_). This output appears when the input for a specific feature is undefined and

hence the system will yield an arbitrary output.

Output element $(\mathsf{DialTone}, n)$ corresponds to the dial tone that user n receives after the system has processed the input (OnHook, n). This is received after the user has lifted the handset and is ready to place a call. Output $(\mathsf{Ring}, n_1, n_2)$ causes telephone unit n_2 to produce an audible ring, typically indicating that there is an incoming call while telephone unit n_1 receives a ring tone. The $(\mathsf{BusyTone}, n)$ signal indicates to caller n that the dialed telephone unit is busy. Output $(\mathsf{RingTone}, n)$ is specific to the CW feature and is received by user n when there is an incoming call and user n is engaged in a call. The $(\mathsf{ErrorTone}, n)$ is received by user n when the switch receives an invalid sequence of inputs. Output $(\mathsf{Conn}, n_1, n_2)$ corresponds to the establishment of a connection between users n_1 and n_2, after which these two users can start a conversation. Output $(\mathsf{Disc}, n_1, n_2)$ terminates a connection between n_1 and n_2, typically after an input (HangUp, n_i) has been sent by one of the parties. The output $(\mathsf{Hold}, n_1, n_2)$ will suspend the connection between n_1 and n_2, without breaking the connection. This allows for n_1 to connect to another user.

6.2 Relational Specification of POTS

Fundamentally, there are two types of rules when writing specifications. These rules are called *basic rules* and *complex rules*. Basic rules are referred to as axioms and complex rules are referred to as rules. Complex rules can be divided into two classes: *reduction rules* and *swap rules*. Moreover, each class can be written using a *tabular* format making the specification more readable than the usual plain mathematical notation of premise over conclusion.

The set of inputs, I, defined in the previous section is the complete set of inputs ranging over the entire system. POTS employs only a subset of I.

$$
\begin{aligned}
I_{POTS} \triangleq\ & \{\mathsf{Init}\} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId}
\end{aligned}
$$

where $I_{POTS} \subseteq I$.

6.2.1 POTS Axioms

Two tables for the axioms are presented. Table 6.1 is a cumulative table containing the set axioms when a connection between two users is successful, while Table 6.2 is a non-cumulative table containing axioms when the connection is unsuccessful.

No	Premise	Input	Output
A-1		(Init)	$\emptyset$
A-2		$(\mathsf{OffHook}, n_1)$	$\{(\mathsf{DialTone}, n_1)\}$
A-3	$n_1 \neq n_2$	$(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{Ring}, n_1, n_2)\}$
A-4		$(\mathsf{OffHook}, n_2)$	$\{(\mathsf{Conn}, n_1, n_2)\}$
A-5	$n \in \{n_1, n_2\}$	(OnHook, n)	$\{(\mathsf{Disc}, n_1, n_2)\}$
A-6	$n' \in \{n_1, n_2\} \wedge n' \neq n$	(OnHook, n')	$\emptyset$

Table 6.1: POTS Axioms: Cumulative Table for a Successful Connection

Recall that the rule associated with line n is:

$$\frac{p_1 \wedge \ldots \wedge p_n}{s_1.\ldots.s_n \triangleleft POTS \triangleright s'_n} .$$

For instance, the axioms associated with lines A-3 and A-5 are respectively:

$$\text{(POTS-A-3):} \quad \frac{n_1 \neq n_2}{(\mathsf{Init}).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2) \triangleleft POTS \triangleright \{(\mathsf{Ring}, n_1, n_2)\}} ,$$

$$\text{(POTS-A-5):} \quad \frac{n_1 \neq n_2 \wedge n \in \{n_1, n_2\}}{(\mathsf{Init}).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_2).(\mathit{ioeOnHook}, n) \triangleleft POTS \triangleright \{(\mathsf{Disc}, n_1, n_2)\}} .$$

Next, we present the non-cumulative table for POTS axioms for unsuccessful connection attempts.

$$\mathsf{c1} \triangleq (\mathsf{Init}).(\mathsf{OffHook}, n_1)$$

No	Premise	Input	Output
A-7	$n_1 \neq n_2$	$\mathsf{c1}.(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{BusyTone}, n_1)\}$
A-8		$\mathsf{c1}.(\mathsf{Dial}, n_1, n_1)$	$\{(\mathsf{BusyTone}, n_1)\}$
A-9	$\mathit{unique}(n_1, n_2, n_3)$	$\mathsf{c1}.(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_1, n_3).(\mathsf{Dial}, n_2, n_3)$	$\{(\mathsf{BusyTone}, n_2)\}$
A-10		$\mathsf{c1}.(\mathsf{OnHook}, n_1)$	$\emptyset$
A-11	$\mathit{unique}(n_1, n_2, n_3)$	$\mathsf{c1}.(\mathsf{Dial}, n_1, n_2).(\mathsf{Dial}, n_1, n_3)$	$\emptyset$
A-12	$k \notin I_{POTS} \wedge x \in I$	$x.k$	-

Table 6.2: POTS Axioms: Non-Cumulative Table For Unsuccessful Connection Attempts

The axiom associated with line n is given by:

$$\frac{p_n}{s_n \triangleleft POTS \triangleright s'_n} \ .$$

For instance, the axiom associated to line A-8 is:

$$\text{(POTS-A-8):} \ \frac{}{(\mathsf{Init}).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_1) \triangleleft POTS \triangleright \{(\mathsf{BusyTone}, n_1)\}} \ .$$

6.2.2 POTS Rules

POTS reduction and swap rules are presented in the two subsequent sections.

6.2.2.1 POTS Reduction Rules

Reduction rules are used to eliminate symbols from the input sequence. Two reduction tables are provided: Table 6.3 illustrates how complete connection cycles are removed from the input sequence, while Table 6.4 describes the removal of connections in progress.

No	Condition	Prefix	Discard	Postfix
R-1	$k \notin I_{POTS} \wedge x' \neq \varepsilon$	x	k	x'
R-2	$x' \neq \varepsilon$	x	$(\mathsf{OffHook}, n_1).(\mathsf{OnHook}, n_1)$	x'
R-3	$x' \neq \varepsilon$	x	$(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2).(\mathsf{OnHook}, n_1)$	x'
R-4	$x' \neq \varepsilon$	$x.(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{Dial}, n_1, n_3)$	x'
R-5	$x' \neq \varepsilon$	$x.(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{Dial}, n_3, n_2)$	x'

Table 6.3: POTS Reduction Rules: Non-Cumulative Table For Complete Connection Cycle

No	Condition	Prefix	Discard	Postfix
R-6		x	$(\mathsf{OffHook}, n_2)$	$(\mathsf{OffHook}, n_1)$
R-7	$n_2 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{OffHook}, n_3)$
R-8	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{Dial}, n_2, n_3)$
R-9	$n_1 \neq n_3 \wedge n_2 \neq n_4$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{Dial}, n_3, n_4)$
R-10	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	(OnHook, n_2)
R-11		x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{OffHook}, n_3)$

Table 6.4: POTS Reduction Rules: Non-Cumulative Table For Connection in Progress

Tables 6.3 and 6.4 are interpreted in terms of axioms in the same manner. Let p, *pre*, *discard*, *post* be respectively the condition, the prefix, the discard and the postfix of line n, and let y be a variable with respect to line n.

Then the associated axiom is:

$$\frac{p \ \wedge \ pre.post \lhd POTS \rhd y}{pre.discard.post \lhd POTS \rhd y} \ .$$

For instance, the rule associated with lines R-3 and R-10 are respectively:

$$\text{(POTS-R-3):} \ \frac{x \neq \varepsilon \ \wedge \ x.x' \lhd POTS \rhd y}{x.(\mathsf{OffHook}, n1).(\mathsf{Dial}, n_1, n_2)(\mathsf{OnHook}, n1).x' \lhd POTS \rhd y} \ .$$

$$\text{(POTS-R-10):} \ \frac{n_1 \neq n_2 \ \wedge \ x.(\mathsf{OffHook}, n_2) \lhd POTS \rhd y}{x.(\mathsf{OffHook}, n_1).(\mathsf{OffHook}, n_2) \lhd POTS \rhd y} \ .$$

To illustrate the application of reduction axioms, suppose we want to prove the following sequence:

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{FlashHook}, a_1)(\mathsf{OffHook}, a_2).$
$(\mathsf{Dial}, a_2, a_3).(\mathsf{Dial}, a_1, a_4) \lhd POTS \rhd \{(\mathsf{Ring}, a_1, a_4)\}$

PROOF.

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{FlashHook}, a_1).$
$(\mathsf{OffHook}, a_2).(\mathsf{Dial}, a_2, a_3).(\mathsf{Dial}, a_1, a_4) \lhd POTS \rhd \{(\mathsf{Ring}, a_1, a_4)\}$

$\Leftarrow$ (POTS-R-9)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{FlashHook}, a_1).(\mathsf{OffHook}, a_2).(\mathsf{Dial}, a_1, a_4) \lhd POTS \rhd \{(\mathsf{Ring}, a_1, a_4)\}$

$\Leftarrow$ (POTS-R-1)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_2).(\mathsf{Dial}, a_1, a_4) \lhd POTS \rhd \{(\mathsf{Ring}, a_1, a_4)\}$

$\Leftarrow$ (POTS-R-6)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_4) \lhd POTS \rhd \{(\mathsf{Ring}, a_1, a_4)\}$

$\Leftarrow$ (POTS-A-3)

true

6.2.2.2 Swap Rules

By applying swap rules, a complex sequence can be transformed into an equivalent sequence. Swap rules are used for swapping input elements with each other until a reduction rule may be applied. Table 6.5 provides the list of swap axioms for POTS.

No	Condition	Input 1	Input 2
S-1		$(\mathsf{OffHook}, n_1)$	$(\mathsf{OffHook}, n_2)$
S-2	$n_1 \neq n_3 \wedge n_2 \neq n_4$	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{Dial}, n_3, n_4)$
S-3		(OnHook, n_1)	(OnHook, n_2)
S-4	$n_1 \neq n_2 \wedge n_1 \neq n_3$	$(\mathsf{OffHook}, n_1)$	$(\mathsf{Dial}, n_2, n_3)$
S-5	$n_1 \neq n_2$	$(\mathsf{OffHook}, n_1)$	(OnHook, n_2)
S-6	$n_1 \neq n_2 \wedge n_1 \neq n_3$	(OnHook, n_1)	$(\mathsf{Dial}, n_2, n_3)$

Table 6.5: POTS Swap Rules : Non-Cumulative Table

Two rules are associated to a line n in a swap table in the following manner: let p, s_1, s_2 be respectively the condition, input element 1, and input element 2 of line n, and let x, x', y be three variables; then the associated rules are:

$$\frac{p \wedge x' \neq \varepsilon \wedge x.s_1.s_2.x' \triangleleft POTS \triangleright y}{x.s_2.s_1.x' \triangleleft POTS \triangleright y} \quad \frac{p \wedge x' \neq \varepsilon \wedge x.s_2.s_1.x' \triangleleft POTS \triangleright y}{x.s_1.s_2.x' \triangleleft POTS \triangleright y} .$$

$$(\text{POTS-S-6} - 1): \frac{n_1 \neq n_2 \wedge n_1 \neq n_3 \wedge x' \neq \varepsilon \wedge x.(\mathsf{OnHook}, n_1).(\mathsf{Dial}, n_2, n_3).x' \triangleleft POTS \triangleright y}{x.(\mathsf{Dial}, n_2, n_3).(\mathsf{OnHook}, n_1).x' \triangleleft POTS \triangleright y} .$$

$$(\text{POTS-S-6} - 2): \frac{n_1 \neq n_2 \wedge n_1 \neq n_3 \wedge x' \neq \varepsilon \wedge x.(\mathsf{Dial}, n_2, n_3)(\mathsf{OnHook}, n_1).x' \triangleleft POTS \triangleright y}{x.(\mathsf{OnHook}, n_1).(\mathsf{Dial}, n_2, n_3).x' \triangleleft POTS \triangleright y} .$$

The above rules state that OnHook and Dial may be swapped interchangeably as long as they do not occur at the end of the input sequence. To illustrate the application of the swap rules, suppose we want to prove the following sequence:

$$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_2).(\mathsf{Dial}, a_1, a_4).$$
$$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_2, a_6).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright (\mathsf{Ring}, a_3, a_5)$$

PROOF.

$$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_2).(\mathsf{Dial}, a_1, a_4).$$
$$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_2, a_6).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_5)\}$$

$\Leftarrow$ (POTS-S-4)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_2).(\mathsf{Dial}, a_1, a_4).(\mathsf{Dial}, a_2, a_6).$
$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_5)\}$

$\Leftarrow$ (POTS-R-11)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_2).(\mathsf{Dial}, a_1, a_4).$
$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_5)\}$

$\Leftarrow$ (POTS-S-4)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_4).(\mathsf{OffHook}, a_2).$
$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_5)\}$

$\Leftarrow$ (POTS-R-11)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_5)\}$

$\Leftarrow$ (POTS-R-6)

$(\mathsf{Init}).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_5)\}$

$\Leftarrow$ (POTS-R-6)

$(\mathsf{Init}).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_5) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_5)\}$

$\Leftarrow$ (POTS-A-3)

true

6.3 Three Way Calling

The Three Way Calling feature (TWC) allows a subscriber to place a call on hold, dial a second user and establish a second connection with another user. The subscriber is then capable of establishing a three way call between all active parties. The feature is activated when the subscriber n_1 presses FlashHook to which the black box replies by putting user n_2 on Hold and informs n_1 to dial the number of the third party n_3 by sending a DialTone to n_1. If the dialed party answers, it is joined into a two way call with n_1, while the other party is still on Hold. If n_1 presses FlashHook, a three way call is formed between n_1, n_2 and n_3.

$$
\begin{aligned}
I_{TWC} \triangleq\ & \{\mathsf{Init}\} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{FlashHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{TWCAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{TWCDeact}\} \times \mathbf{PhoneId}
\end{aligned}
$$

where $I_{TWC} \subseteq I$.

6.3.1 Three Way Calling Axioms

In this section, we present a definition of TWC axioms. Note that Axioms POTS-A-1 to POTS-A-12 also apply as part of TWC axioms. Therefore, there is no need to repeat these axioms. There is however one axiom that must be modified, namely axiom POTS-A-12. We must substitute I_{TWC} for I_{POTS}. Therefore the new A-12 for TWC is presented in table 6.6.

No	Premise	Input	Output
A-12	$k \notin I_{TWC} \wedge x \in I$	$x.k$	-

Table 6.6: TWC Axiom for Invalid Input

Let

$\mathsf{c2} \triangleq (\mathsf{Init}).(\mathsf{TWCAct}, n1).(\mathsf{OffHook}, n_1)$

$\mathsf{c3} \triangleq (\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_2)$

$\mathsf{c4} \triangleq (\mathsf{FlashHook}, n_1).(\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_2)$

No	Premise	Input	Output
A-13		(Init).(TWCAct, n_1)	$\emptyset$
A-14	$n_1 \neq n_2$	c3.(FlashHook, n_1)	{(DialTone, n_1), (Hold, n_1, n_2)}
A-15	$n_1 \neq n_3$	(Dial, n_1, n_3)	{(Ring, n_1, n_3)}
A-16		(OffHook, n_3)	{(Conn, n_1, n_3)}
A-17		(FlashHook, n_1)	{(Conn, n_1, n_2), (Conn, n_2, n_3)}
A-18	$z \in \{2,3\} \wedge w \in \{2,3\} \wedge z \neq w$	(OnHook, n_z)	{(Disc, n_1, n_z), (Disc, n_w, n_z)}
A-19		(OnHook, n_w)	{(Disc, n_1, n_w)}
A-20		(OnHook, n_1)	$\emptyset$

Table 6.7: TWC Axioms: Cumulative Table for a Successful Connection

Axiom TWC-A-14 describes when user n_1 is engaged in a conversation with n_2 and wants to initiate a call to a third party. User n_1 will receive a DialTone while user n_2 will be placed on Hold. Axiom TWC-A-18 looks more complex than it really is. It describes the case when n_1 has initiated a three way call with n_2 and n_3. If n_2 hangs up, then n_2 will be disconnected from n_1 and n_3. Similarly if n_3 hangs up, then n_3 will be disconnected from n_1 and n_2.

Table 6.8 describes the axioms for unsuccessful TWC connection attempts.

No	Premise	Input	Output
A-21	$unique(n_1, n_2, n_3)$	$c2$.(OffHook, n_2).(Dial, n_1, n_3).(Dial, n_2, n_3)	{(BusyTone, n_2)}
A-22	$unique(n_1, n_2, n_3)$	(Init).$c3$.(FlashHook, n_1).(OffHook, n_3).(Dial, n_1, n_3)	{(BusyTone, n_1)}
A-23		x.(TWCDeact, n_1)	$\emptyset$

Table 6.8: TWC Axioms: Non-Cumulative Table For Unsuccessful Connection Attempts

6.3.2 Three Way Calling Rules

In this section, we present the reduction and swap rules for the three way calling feature.

6.3.2.1 Three Way Calling Reduction Rules

The reduction rules for TWC include the reduction rules for POTS. Again, there is one rule that must be modified, namely POTS-R-1. The new R-1 for TWC is:

No	Condition	Prefix	Discard	Postfix
R-1	$k \notin I_{TWC}$	x	k	x'

Table 6.9: TWC Reduction Rule for Invalid Input

No	Condition	Prefix	Discard	Postfix
R-12	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	c2.(OnHook, n_2)	x'
R-13	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	c2.(FlashHook, $n1$).(OnHook, n_2)	x'
R-14	$x' \neq \varepsilon$	x.(OffHook, n_1)	(FlashHook, n_1)	x'
R-15	$x' \neq \varepsilon$	x	(TWCAct, n_1).(TWCDeact, n_1)	x'

Table 6.10: TWC Reduction Axioms: Non-Cumulative Table For Complete Connection Cycle

Rule TWC-R-14 describes when user n_1 lifts the handset and presses FlashHook, the state of the system will not change. Similarly rule TWC-R-15 simply states if the three way calling feature is activated, and then immediately afterwards deactivated, the state of the system remains unchanged.

Next, we present the TWC reduction rules for connections in progress.

No	Condition	Prefix	Discard	Postfix
R-16	$n_1 \neq n_2$	x	(TWCAct, n_1)	(OffHook, n_2)
R-17	$n_1 \neq n_2$	x	(TWCAct, n_1)	(Dial, n_2, n_3)
R-18	$n_1 \neq n_2$	x	(TWCAct, n_1)	(OnHook, n_2)
R-19	$n_1 \neq n_2$	x	(TWCAct, n_1)	(FlashHook, n_2)
R-20	$n_1 \neq n_2$	x	(TWCAct, n_1)	(TWCAct, n_2)
R-21	$n_1 \neq n_2$	x	(TWCAct, n_1)	(TWCDeact, n_2)
R-22	$n_1 \neq n_2$	x	(TWCDeact, n_1)	(OffHook, n_2)
R-23	$n_1 \neq n_2$	x	(OffHook, n_1)	(FlashHook, n_2)
R-24	$n_1 \neq n_2$	x	(OffHook, n_1)	(TWCAct, n_2)
R-25	$n_1 \neq n_2$	x	(OffHook, n_1)	(TWCDeact, n_2)
R-26	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(FlashHook, n_3)
R-27	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(TWCAct, n_3)
R-28	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(TWCDeact, n_3)
R-29	$n_1 \neq n_2$	x	(FlashHook, n_1)	(OffHook, n_2)
R-30	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Dial, n_2, n_3)
R-31	$n_1 \neq n_2$	x	(FlashHook, n_1)	(FlashHook, n_2)
R-32	$n_1 \neq n_2$	x	(FlashHook, n_1)	(TWCAct, n_2)
R-33	$n_1 \neq n_2$	x	(FlashHook, n_1)	(TWCDeact, n_2)

Table 6.11: TWC Reduction Rules: Non-Cumulative Table For Connection in Progress

6.3.2.2 Three Way Calling Swap Rules

The following table describes the swap rules for TWC. The swap rules for POTS are valid swap rules for TWC and therefore need not be repeated.

No	Condition	Input 1	Input 2
S-7	$n_1 \neq n_2$	(TWCAct, n_1)	(OffHook, n_2)
S-8	$n_1 \neq n_2$	(TWCAct, n_1)	(Dial, n_2, n_3)
S-9	$n_1 \neq n_2$	(TWCAct, n_1)	(OnHook, n_2)
S-10	$n_1 \neq n_2$	(TWCAct, n_1)	(FlashHook, n_2)
S-11	$n_1 \neq n_2$	(TWCAct, n_1)	(TWCAct, n_2)
S-12	$n_1 \neq n_2$	(TWCAct, n_1)	(TWCDeact, n_2)
S-13	$n_1 \neq n_2$	(TWCDeact, n_1)	(OffHook, n_2)
S-14	$n_1 \neq n_2$	(TWCDeact, n_1)	(Dial, n_2, n_3)
S-15	$n_1 \neq n_2$	(TWCDeact, n_1)	(OnHook, n_2)
S-16	$n_1 \neq n_2$	(TWCDeact, n_1)	(FlashHook, n_2)
S-17	$n_1 \neq n_2$	(TWCDeact, n_1)	(TWCDeact, n_2)
S-18	$n_1 \neq n_2$	(FlashHook, n_1)	(OffHook, n_2)
S-19	$n_1 \neq n_2$	(FlashHook, n_1)	(Dial, n_2, n_3)
S-20	$n_1 \neq n_2$	(FlashHook, n_1)	(FlashHook, n_2)

Table 6.12: TWC Swap Rules : Non-Cumulative Table

The following sequence is reduced and proven using the axioms and rules defined for TWC.

PROOF.

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).(\mathsf{TWCAct}, a_3).(\mathsf{OffHook}, a_3).$
$(\mathsf{FlashHook}, a_1).(\mathsf{Dial}, a_1, a_4).(\mathsf{OffHook}, a_4) \triangleleft TWC \triangleright \{(\mathsf{Conn}, a_1, a_4)\}$

$\Leftarrow$ (TWC-S-8)

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{TWCAct}, a_3).(\mathsf{Dial}, a_1, a_2).(\mathsf{OffHook}, a_3).$
$(\mathsf{FlashHook}, a_1).(\mathsf{Dial}, a_1, a_4).(\mathsf{OffHook}, a_4) \triangleleft TWC \triangleright \{(\mathsf{Conn}, a_1, a_4)\}$

$\Leftarrow$ (TWC-R-17)

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).(\mathsf{OffHook}, a_3).$
$(\mathsf{FlashHook}, a_1).(\mathsf{Dial}, a_1, a_4).(\mathsf{OffHook}, a_4) \triangleleft TWC \triangleright \{(\mathsf{Conn}, a_1, a_4)\}$

$\Leftarrow$ (TWC-R-23)

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).$
$(\mathsf{FlashHook}, a_1).(\mathsf{Dial}, a_1, a_4).(\mathsf{OffHook}, a_4) \triangleleft TWC \triangleright \{(\mathsf{Conn}, a_1, a_4)\}$

$\Leftarrow$ (TWC-A-16)

true

6.4 Call Waiting

The Call Waiting features allows a subscriber who is already engaged in a conversation with another party to receive a call from a third party. If the subscriber receives a tone from the third party, the incoming call can be enabled by pressing FlashHook. Note that the initial calling party is put on Hold allowing only one conversation to be active at a given time. The subscriber may toggle between the two parties by consecutively pressing FlashHook.

$$
\begin{aligned}
I_{CW} \triangleq\ & \{\mathsf{Init}\} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{FlashHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CWAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CWDeact}\} \times \mathbf{PhoneId}
\end{aligned}
$$

where $I_{CW} \subseteq I$.

6.4.1 Call Waiting Axioms

In this section, we present a definition of CW axioms. Again, axioms POTS-A-1 to POTS-A-12 also apply as part of CW axioms. Therefore, we won't repeat them. The new A-12 for CW is presented in table 6.13.

No	Premise	Input	Output
A-12	$k \notin I_{CW} \wedge x \in I$	$x.k$	-

Table 6.13: CW Axiom for Invalid Input

$$\mathsf{c5} \triangleq (\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_2).(\mathsf{OffHook}, n_3).(\mathsf{Dial}, n_3, n_1)$$

No	Premise	Input	Output
A-12		Init.(CWAct, n_1)	$\emptyset$
A-13	$unique(n_1, n_2, n_3)$	c5	{(RingTone, n_3), (WaitTone, n_1)}
A-14		(FlashHook, n_1)	{(Disc, n_1, n_2), (Conn, n_1, n_3)}
A-15		(FlashHook, n_1)	{(Conn, n_1, n_2), (Disc, n_1, n_3)}
A-16		(OnHook, n_3)	$\emptyset$

Table 6.14: CW Axioms: Cumulative Table for a Successful Connection

No	Input	Output
A-17	(Init).(CWAct, n_1).c5(FlashHook, n_1).(OffHook, n_4).((Dial, n_4, n_1)	{(BusyTone, n_4)}
A-18	x.(CWDeact, n_1)	$\emptyset$

Table 6.15: CW Axioms: Non-Cumulative Table For Unsuccessful Connection Attempts

Axiom CW-A-13 describes when user n_1 is engaged in a conversation with user n_2 and has an incoming call initiated by n_3 at the same time. User n_3 will receive RingTone as an output while user n_2 will receive the WaitTone output. To enable n_3's call n_1 must press FlashHook. An output of Disc is received by n_1 and n_2 while an output of Conn is received by n_1 and n_3 as described by axiom CW-A-14.

6.4.2 Call Waiting Rules

In this section, we present the reduction and swap rules for the call waiting feature.

6.4.2.1 Call Waiting Reduction Rules

The reduction rules for CW include the reduction rules for POTS. Again, there is one rule that must be modified, namely POTS-R-1. The new R-1 for TWC is described in table 6.16.

No	Condition	Prefix	Discard	Postfix
R-1	$k \notin I_{CW}$	x	k	x'

Table 6.16: CW Reduction Rule for Invalid Input

c6 $\triangleq$ (Dial, n_1, n_2).(FlashHook, n_1).(OffHook, n_2)

No	Condition	Prefix	Discard	Postfix
R-12	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	c6.(OnHook, n_2)	x'
R-13	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	c6.(FlashHook, $n1$).(OnHook, n_2)	x'
R-14	$x' \neq \varepsilon$	x.(OffHook, n_1)	(FlashHook, n_1)	x'
R-15	$x' \neq \varepsilon$	x	(FlashHook, n_1).(FlashHook, n_1)	x'
R-16	$x' \neq \varepsilon$	x	(CWAct, n_1).(CWDeact, n_1)	x'

Table 6.17: CW Reduction Axioms : Non-Cumulative Table For Complete Connection Cycle

Rule CW-R-15 describes when user n_1 presses FlashHook two consecutive times, the state of the system remains unchanged. Similarly, rule CW-R-16 describes when an input of CWAct followed immediately by an input of CWDeact will also keep the system unchanged.

Next, we present the CW reduction rules for connections in progress.

No	Condition	Prefix	Discard	Postfix
R-17	$n_1 \neq n_2$	x	(CWAct, n_1)	(OffHook, n_2)
R-18	$n_1 \neq n_2$	x	(CWAct, n_1)	(Dial, n_2, n_3)
R-19	$n_1 \neq n_2$	x	(CWAct, n_1)	(OnHook, n_2)
R-20	$n_1 \neq n_2$	x	(CWAct, n_1)	(FlashHook, n_2)
R-21	$n_1 \neq n_2$	x	(CWAct, n_1)	(CWAct, n_2)
R-22	$n_1 \neq n_2$	x	(CWAct, n_1)	(CWDeact, n_2)
R-23	$n_1 \neq n_2$	x	(OffHook, n_1)	(FlashHook, n_2)
R-24	$n_1 \neq n_2$	x	(OffHook, n_1)	(CWAct, n_2)
R-25	$n_1 \neq n_2$	x	(OffHook, n_1)	(CWDeact, n_2)
R-26	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(FlashHook, n_3,)
R-27	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CWAct, n_3)
R-28	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CWDeact, n_3)
R-29	$n_1 \neq n_2$	x	(FlashHook, n_1)	(OnHook, n_2)
R-30	$n_1 \neq n_2$	x	(FlashHook, n_1)	(OffHook, n_2)
R-31	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Dial, n_2, n_3)
R-32	$n_1 \neq n_2$	x	(FlashHook, n_1)	(FlashHook, n_2)
R-33	$n_1 \neq n_2$	x	(FlashHook, n_1)	(CWDeact, n_2)

Table 6.18: CW Reduction Rules: Non-Cumulative Table For Connection in Progress

6.4.2.2 Call Waiting Swap Rules

The following table describes the swap rules for CW. The swap rules for POTS are valid swap rules for CW and therefore need not be repeated.

No	Condition	Input 1	Input 2
S-7	$n_1 \neq n_2$	(CWAct, n_1)	(OffHook, n_2)
S-8	$n_1 \neq n_2$	(CWAct, n_1)	(Dial, n_2, n_3)
S-9	$n_1 \neq n_2$	(CWAct, n_1)	(OnHook, n_2)
S-10	$n_1 \neq n_2$	(CWAct, n_1)	(FlashHook, n_2)
S-11	$n_1 \neq n_2$	(CWAct, n_1)	(CWAct, n_2)
S-12	$n_1 \neq n_2$	(CWAct, n_1)	(CWDeact, n_2)
S-13	$n_1 \neq n_2$	(CWDeact, n_1)	(OffHook, n_2)
S-14	$n_1 \neq n_2$	(CWDeact, n_1)	(Dial, n_2, n_3)
S-15	$n_1 \neq n_2$	(CWDeact, n_1)	(OnHook, n_2)
S-16	$n_1 \neq n_2$	(CWDeact, n_1)	(FlashHook, n_2)
S-17	$n_1 \neq n_2$	(CWDeact, n_1)	(CWDeact, n_2)
S-18	$n_1 \neq n_2$	(FlashHook, n_1)	(OnHook, n_2)
S-19	$n_1 \neq n_2$	(FlashHook, n_1)	(OffHook, n_2)
S-20	$n_1 \neq n_2$	(FlashHook, n_1)	(Dial, n_2, n_3)
S-21	$n_1 \neq n_2$	(FlashHook, n_1)	(FlashHook, n_2)

Table 6.19: CW Swap Rules : Non-Cumulative Table

The following sequence is reduced and proven using the axioms and rules defined for TWC.

PROOF.

$$(\mathsf{Init}).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).(\mathsf{OffHook}, a_2).(\mathsf{CWAct}, a_4).(\mathsf{OffHook}, a_4).$$
$$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft CW \triangleright \{(\mathsf{Disc}, a_1, a_2), (\mathsf{Conn}, a_1, a_3)\}$$

$\Leftarrow$ (CW-S-7)

$$(\mathsf{Init}).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).(\mathsf{CWAct}, a_4).(\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_4).$$
$$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft CW \triangleright \{(\mathsf{Disc}, a_1, a_2), (\mathsf{Conn}, a_1, a_3)\}$$

$\Leftarrow$ (CW-R-17)

$$(\mathsf{Init}).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).(\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_4).$$
$$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft CW \triangleright \{(\mathsf{Disc}, a_1, a_2), (\mathsf{Conn}, a_1, a_3)\}$$

$\Leftarrow$ (POTS-R-6)

$$(\mathsf{Init}).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).(\mathsf{OffHook}, a_2).$$
$$(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft CW \triangleright \{(\mathsf{Disc}, a_1, a_2), (\mathsf{Conn}, a_1, a_3)\}$$

$\Leftarrow$ (CW-A-13)

true

6.5 Originating Call Screening

The Originating Call Screening feature (OCS) prevents a specified number from being dialed by a subscriber. The disallowed number is added to the database of the black box by initiating a OCSAdd. Similarly, the number is removed from the database by the subscriber when a OCSDel is initiated.

$$
\begin{aligned}
I_{OCS} \triangleq\ & \{\mathsf{Init}\} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSAdd}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSDel}\} \times \mathbf{PhoneId}
\end{aligned}
$$

where $I_{OCS} \subseteq I$.

6.5.1 Originating Call Screening Axioms

In this section, we present a definition of OCS axioms. Again, axioms POTS-A-1 to POTS-A-12 also apply as part of OCS axioms. Therefore, we won't repeat them. The new A-12 for OCS is:

No	Premise	Input	Output
A-12	$k \notin I_{OCS} \wedge x \in I$	$x.k$	-

Table 6.20: OCS Axiom for Invalid Input

No	Premise	Input	Output
A-13		$(\mathsf{Init}).(\mathsf{OCSAct}, n_1)$	$\emptyset$
A-14	$n_1 \neq n_2$	$(\mathsf{OCSAdd}, n_1, n_2)$	$\emptyset$
A-15		$(\mathsf{OffHook}, n_1)$	$\{(\mathsf{DialTone}, n_1)\}$
A-16		$(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{ErrorTone}, n_1)\}$
A-17		(OnHook, n_1)	$\emptyset$

Table 6.21: OCS Axioms: Cumulative Table for a Successful Connection

Axiom OCS-A-16 describes when user n_1 has number n_2 as one of the numbers to be screened. When n_1 attempts to dial n_2, an output of ErrorTone is received.

$$\mathsf{c7} \triangleq (\mathsf{Init}).(\mathsf{OCSAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{OCSAdd}, n_1, n_2)$$

No	Premise	Input	Output
A-18	$unique(n_1, n_2, n_3)$	$\mathsf{c7}.(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{ErrorTone}, n_1)\}$
A-19		$(\mathsf{Init}).(\mathsf{OCSAct}, n_1).(\mathsf{OCSAdd}, n_1, n_1)$	$\{(\mathsf{ErrorTone}, n_1)\}$
A-20	$(\mathsf{OCSAdd}, n_1, n_2) \in x$	$x.(\mathsf{OCSDel}, n_1, n_2)$	$\emptyset$
A-21	$(\mathsf{OCSAct}, n_1) \in x$	$x.(\mathsf{OCSDeact}, n_1)$	$\emptyset$

Table 6.22: OCS Axioms: Non-Cumulative Table For Unsuccessful Connection Attempts

6.5.2 Originating Call Screening Rules

In this section, we present the reduction and swap rules for the originating call screening feature.

6.5.2.1 Originating Call Screening Reduction Rules

The reduction rules for OCS include the reduction rules for POTS. Again, there is one rule that must be modified, namely POTS-R-1. The new R-1 for OCS is described in table 6.23.

No	Condition	Prefix	Discard	Postfix
R-1	$k \notin I_{OCS}$	x	k	x'

Table 6.23: OCS Reduction Rule for Invalid Input

No	Condition	Prefix	Discard	Postfix
R-12	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	$(\mathsf{OCSAdd}, n_1, n_2).(\mathsf{OCSDel}, n_1, n_2)$	x'
R-13	$x' \neq \varepsilon \wedge w \geq 0$	x	$(\mathsf{OCSAct}, n_1).(\mathsf{OCSAdd}, n_1, ?)^w.(\mathsf{OCSDeact}, n_1)$	x'

Table 6.24: OCS Reduction Axioms: Non-Cumulative Table For Complete Connection Cycle

Rule OCS-R-12 describes when user n_1 adds n_2 to the screening list and immediately removes it. The state of the system does not change when this sequence is encountered, therefore OCSAdd and OCSDel can be discarded. Similarly, when n_1 adds several numbers to the screening list (the "?" means any valid number) and OCS is deactivated immediately after the last OCSAdd, the system remains unchanged as described by OCS-R-13.

Next, we present the OCS reduction rules for connections in progress.

No	Condition	Prefix	Discard	Postfix
R-14	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OffHook, n_2)
R-15	$n_1 \neq n_2$	x	(OCSAct, n_1)	(Dial, n_2, n_3)
R-16	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OnHook, n_2)
R-17	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSAdd, n_2, n_3)
R-18	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSAct, n_2)
R-19	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSDeact, n_2)
R-20	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSAdd, n_2, n_3)
R-21	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSDel, n_2, n_3)
R-22	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSAct, n_2)
R-23	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSDeact, n_2)
R-24	$n_1 \neq n_2$	x	(OnHook, n_1)	(OCSAdd, n_2, n_3)
R-25	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OffHook, n_3)
R-26	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OCSAdd, n_3, n_4)
R-27	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OCSDel, n_3, n_4)
R-28	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OCSAct, n_3)
R-29	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OCSDeact, n_3)
R-30	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OffHook, n_3)
R-31	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(Dial, n_3, n_4)
R-32	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OnHook, n_3)
R-33	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSAdd, n_3, n_4)
R-34	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSDel, n_3, n_4)
R-35	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSAct, n_3)
R-36	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSDeact, n_3)

Table 6.25: OCS Reduction Rules: Non-Cumulative Table For Connection in Progress

6.5.2.2 Originating Call Screening Swap Rules

The following table describes the swap rules for OCS. The swap rules for POTS are valid swap rules for OCS and therefore need not be repeated.

No	Condition	Input 1	Input 2
S-7	$n_1 \neq n_2$	(OCSAct, n_1)	(OffHook, n_2)
S-8	$n_1 \neq n_2$	(OCSAct, n_1)	(Dial, n_2, n_3)
S-9	$n_1 \neq n_2$	(OCSAct, n_1)	(OnHook, n_2)
S-10	$n_1 \neq n_2$	(OCSAct, n_1)	(OCSAdd, n_2, n_3)
S-11	$n_1 \neq n_2$	(OCSAct, n_1)	(OCSAct, n_2)
S-12	$n_1 \neq n_2$	(OCSAct, n_1)	(OCSDeact, n_2)
S-13	$n_1 \neq n_2$	(OCSDeact, n_1)	(OffHook, n_2)
S-14	$n_1 \neq n_2$	(OCSDeact, n_1)	(Dial, n_2, n_3)
S-15	$n_1 \neq n_2$	(OCSDeact, n_1)	(OnHook, n_2)
S-16	$n_1 \neq n_2$	(OCSDeact, n_1)	(OCSAdd, n_2, n_3)
S-17	$n_1 \neq n_2$	(OCSDeact, n_1)	(OCSDel, n_2, n_3)
S-18	$n_1 \neq n_2$	(OCSDeact, n_1)	(OCSDeact, n_2)
S-19	$n_1 \neq n_3$	(OCSAdd, n_1, n_2)	(OffHook, n_3)
S-20	$n_1 \neq n_3$	(OCSAdd, n_1, n_2)	(Dial, n_3, n_4)
S-21	$n_1 \neq n_3$	(OCSAdd, n_1, n_2)	(OnHook, n_3)
S-22	$n_1 \neq n_3$	(OCSAdd, n_1, n_2)	(OCSAdd, n_3, n_4)
S-23	$n_1 \neq n_3$	(OCSAdd, n_1, n_2)	(OCSDel, n_3, n_4)
S-24	$n_1 \neq n_3$	(OCSDel, n_1, n_2)	(OffHook, n_3)
S-25	$n_1 \neq n_3$	(OCSDel, n_1, n_2)	(Dial, n_3, n_4)
S-26	$n_1 \neq n_3$	(OCSDel, n_1, n_2)	(OnHook, n_3)
S-27	$n_1 \neq n_3$	(OCSDel, n_1, n_2)	(OCSDel, n_3, n_4)

Table 6.26: OCS Swap Rules : Non-Cumulative Table

The following sequence is reduced and proven using the axioms and rules defined for OCS.

PROOF.

$$(\mathsf{Init}).(\mathsf{OCSAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{OCSAct}, a_3).(\mathsf{OffHook}, a_3).$$
$$(\mathsf{Dial}, a_1, a_2).\triangleleft POTS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$$

$\Leftarrow$ (OCS-S-10)

$$(\mathsf{Init}).(\mathsf{OCSAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{OCSAct}, a_3).(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{OffHook}, a_3).$$
$$(\mathsf{Dial}, a_1, a_2).\triangleleft POTS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$$

$\Leftarrow$ (OCS-R-17)

$$(\mathsf{Init}).(\mathsf{OCSAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{OffHook}, a_3).$$
$$(\mathsf{Dial}, a_1, a_2).\triangleleft POTS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$$

$\Leftarrow$ (OCS-S-19)

$$(\mathsf{Init}).(\mathsf{OCSAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{OCSAdd}, a_1, a_2).$$
$$(\mathsf{Dial}, a_1, a_2).\triangleleft POTS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$$

$\Leftarrow$ (OCS-R-20)

$(\mathsf{Init}).(\mathsf{OCSAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{OCSAdd}, a_1, a_2).$
$(\mathsf{Dial}, a_1, a_2).\triangleleft POTS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$

$\Leftarrow$ (OCS-A-18)

true

6.6 Abbreviated Dialing

The Abbreviated Dialing feature (ABD) allows a subscriber to create an alias for a number which is oftenly used. The alias is typically an abbreviation of the actual dialed number. The alias is added to the database by the subscriber by initiating a ABDAdd. Similarly, the alias is removed from the database by the subscriber when a ABDDel is initiated.

$$
\begin{aligned}
I_{ABD} \triangleq\ & \{\mathsf{Init}\} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDAdd}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDDel}\} \times \mathbf{PhoneId}
\end{aligned}
$$

where $I_{ABD} \subseteq I$.

6.6.1 Abbreviated Dialing Axioms

In this section, we present a definition of ABD axioms. Again, axioms POTS-A-1 to POTS-A-12 also apply as part of ABD axioms. Therefore, we won't repeat them. The new A-12 for ABD is presented in table 6.27.

No	Premise	Input	Output
A-12	$k \notin I_{ABD} \wedge x \in I$	$x.k$	-

Table 6.27: ABD Axiom for Invalid Input

No	Premise	Input	Output
A-13		(Init).(ABDAct, n_1)	$\emptyset$
A-14		(OffHook, n_1)	{(DialTone, n_1)}
A-15	$unique(n_1, n_2, n_3)$	(ABDAdd, n_1, n_2, n_3)	$\emptyset$
A-16		(Dial, n_1, n_2)	{(Ring, n_1, n_3)}
A-17		(OffHook, n_3)	{(Conn, n_1, n_3)}
A-18	$n \in \{n_1, n_3\}$	(OnHook, n)	{(Disc, n_1, n_3)}
A-19	$n' \in \{n_1, n_3\} \wedge n' \neq n$	(OnHook, n')	$\emptyset$

Table 6.28: ABD Axioms: Cumulative Table for a Successful Connection

$$c8 \triangleq (\text{Init}).(\text{ABDAct}, n_1).(\text{OffHook}, n_1).(\text{ABDAdd}, n_1, n_2, n_3)$$

No	Premise	Input	Output
A-20	$unique(n_1, n_2, n_3)$	c8.(OffHook, n_3).(Dial, n_1, n_2)	{(BusyTone, n_1)}
A-21	$unique(n_1, n_2, n_3, n_4)$	c8.(OffHook, n_4).(Dial, n_4, n_3).(Dial, n_1, n_2)	{(BusyTone, n_1)}
A-22	(ABDAdd, n_1, n_2, n_3) $\in x \wedge n_2 \neq n_3$	x.(ABDDel, n_1, n_2)	$\emptyset$
A-23	(ABDAct, n_1) $\in x$	x.(ABDDeact, n_1)	$\emptyset$

Table 6.29: ABD Axioms: Non-Cumulative Table For Unsuccessful Connection Attempts

Axiom ABD-A-16 describes when user n_1 has created an alias for n_3 and called it n_2. When n_1 dials n_2, telephone number n_3 will be dialed and an output of {(Ring, n_1, n_3)} will be received.

6.6.2 Abbreviated Dialing Rules

In this section, we present the reduction and swap rules for the abbreviated dialing feature.

6.6.2.1 Abbreviated Dialing Reduction Rules

The reduction rules for ABD include the reduction rules for POTS. Again, there is one rule that must be modified, namely POTS-R-1. The new R-1 for ABD is described in table 6.30.

No	Condition	Prefix	Discard	Postfix
R-1	$k \notin I_{ABD}$	x	k	x'

Table 6.30: ABD Reduction Rule for Invalid Input

No	Condition	Prefix	Discard	Postfix
R-12	$x' \neq \varepsilon \wedge n_1 \neq n_3$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3).(\mathsf{ABDDel}, n_1, n_2)$	x'
R-13	$x' \neq \varepsilon \wedge r \neq s \wedge w \geq 0$	x	$(\mathsf{ABDAct}, n_1).(\mathsf{ABDAdd}, n_1, r, s)^w.(\mathsf{ABDDeact}, n_1)$	x'

Table 6.31: ABD Reduction Axioms: Non-Cumulative Table For Complete Connection Cycle

Rule ABD-R-12 describes when user n_1 creates an alias for n_3 and deletes it immediately after. The state of the system does not change when this sequence is encountered, therefore ABDAdd and ABDDel can be discarded. Similarly, when n_1 creates several aliases and ABD is deactivated immediately after the last ABDAdd, the system remains unchanged as described by ABD-R-13.

Next, we present the ABD reduction rules for connections in progress.

No	Condition	Prefix	Discard	Postfix
R-14	$n_1 \neq n_2$	x	(ABDAct, n_1)	$(\mathsf{OffHook}, n_2)$
R-15	$n_1 \neq n_2$	x	(ABDAct, n_1)	$(\mathsf{Dial}, n_2, n_3)$
R-16	$n_1 \neq n_2$	x	(ABDAct, n_1)	(OnHook, n_2)
R-17	$n_1 \neq n_2$	x	(ABDAct, n_1)	$(\mathsf{ABDAdd}, n_2, n_3, n_4)$
R-18	$n_1 \neq n_2$	x	(ABDAct, n_1)	(ABDAct, n_2)
R-19	$n_1 \neq n_2$	x	(ABDAct, n_1)	$(\mathsf{ABDDeact}, n_2)$
R-20	$n_1 \neq n_2$	x	$(\mathsf{ABDDeact}, n_1)$	$(\mathsf{OffHook}, n_2)$
R-21	$n_1 \neq n_2$	x	$(\mathsf{ABDDeact}, n_1)$	$(\mathsf{Dial}, n_2, n_3)$
R-22	$n_1 \neq n_2$		$(\mathsf{ABDDeact}, n_1)$	(OnHook, n_2)
R-23	$n_1 \neq n_2$	x	$(\mathsf{ABDDeact}, n_1)$	$(\mathsf{ABDAdd}, n_2, n_3, n_4)$
R-24	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{ABDDel}, n_2, n_3)$
R-25	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	(ABDAct, n_2)
R-26	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{ABDDeact}, n_2)$
R-27	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{ABDAdd}, n_3, n_4)$
R-28	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{ABDDel}, n_3, n_4)$
R-29	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	(ABDAct, n_3)
R-30	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{ABDDeact}, n_3)$
R-31	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{OffHook}, n_4)$
R-32	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{Dial}, n_4, n_5)$
R-33	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	(OnHook, n_4)
R-34	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{ABDAdd}, n_4, n_5, n_6)$
R-35	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{ABDDel}, n_4, n_5)$
R-36	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	(ABDAct, n_4)
R-37	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{ABDDeact}, n_4)$

Table 6.32: ABD Reduction Rules: Non-Cumulative Table For Connection in Progress

6.6.2.2 Abbreviated Dialing Swap Rules

The following table describes the swap rules for ABD. The swap rules for POTS are valid swap rules for ABD and therefore need not be repeated.

No	Condition	Input 1	Input 2
S-7	$n_1 \neq n_2$	(ABDAct, n_1)	(OffHook, n_2)
S-8	$n_1 \neq n_2$	(ABDAct, n_1)	(Dial, n_2, n_3)
S-9	$n_1 \neq n_2$	(ABDAct, n_1)	(OnHook, n_2)
S-10	$n_1 \neq n_2$	(ABDAct, n_1)	(ABDAdd, n_2, n_3, n_4)
S-11	$n_1 \neq n_2$	(ABDAct, n_1)	(ABDAct, n_2)
S-12	$n_1 \neq n_2$	(ABDAct, n_1)	(ABDDeact, n_2)
S-13	$n_1 \neq n_2$	(ABDDeact, n_1)	(OffHook, n_2)
S-14	$n_1 \neq n_2$	(ABDDeact, n_1)	(Dial, n_2, n_3)
S-15	$n_1 \neq n_2$	(ABDDeact, n_1)	(OnHook, n_2)
S-16	$n_1 \neq n_2$	(ABDDeact, n_1)	(ABDAdd, n_2, n_3, n_4)
S-17	$n_1 \neq n_2$	(ABDDeact, n_1)	(ABDDel, n_2, n_3)
S-18	$n_1 \neq n_2$	(ABDDeact, n_1)	(ABDDeact, n_2)
S-19	$n_1 \neq n_4$	(ABDAdd, n_1, n_2, n_3)	(OffHook, n_4)
S-20	$n_1 \neq n_4$	(ABDAdd, n_1, n_2, n_3)	(Dial, n_4, n_5)
S-21	$n_1 \neq n_4$	(ABDAdd, n_1, n_2, n_3)	(OnHook, n_4)
S-22	$n_1 \neq n_4$	(ABDAdd, n_1, n_2, n_3)	(ABDAdd, n_4, n_5, n_6)
S-23	$n_1 \neq n_4$	(ABDAdd, n_1, n_2, n_3)	(ABDDel, n_4, n_5)
S-24	$n_1 \neq n_3$	(ABDDel, n_1, n_2)	(OffHook, n_3)
S-25	$n_1 \neq n_3$	(ABDDel, n_1, n_2)	(Dial, n_3, n_4)
S-26	$n_1 \neq n_3$	(ABDDel, n_1, n_2)	(OnHook, n_3)
S-27	$n_1 \neq n_3$	(ABDDel, n_1, n_2)	(ABDDel, n_3, n_4)

Table 6.33: ABD Swap Rules: Non-Cumulative Table

The following sequence is reduced and proven using the axioms and rules defined for ABD.

PROOF.

(Init).(ABDAct, a_1).(OffHook, a_1).(ABDAdd, a_1, a_2, a_3).(ABDDeact, a_4).(OffHook, a_5).
(Dial, a_5, a_6).(ABDDeact, a_3).(Dial, a_1, a_2)$\triangleleft POTS \triangleright$\{(Ring, a_1, a_3)\}

$\Leftarrow$ (ABD-S-13)

(Init).(ABDAct, a_1).(OffHook, a_1).(ABDAdd, a_1, a_2, a_3).(OffHook, a_5).(ABDeact, a_4).
(Dial, a_5, a_6).(ABDDeact, a_3).(Dial, a_1, a_2)$\triangleleft POTS \triangleright$\{(Ring, a_1, a_3)\}

$\Leftarrow$ (ABD-R-25)

(Init).(ABDAct, a_1).(OffHook, a_1).(ABDAdd, a_1, a_2, a_3).(ABDeact, a_4).
(Dial, a_5, a_6).(ABDDeact, a_3).(Dial, a_1, a_2)$\triangleleft POTS \triangleright$\{(Ring, a_1, a_3)\}

$\Leftarrow$ (ABD-R-30)

(Init).(ABDAct, a_1).(OffHook, a_1).(ABDAdd, a_1, a_2, a_3).(ABDeact, a_4).
(ABDDeact, a_3).(Dial, a_1, a_2)$\triangleleft POTS \triangleright$\{(Ring, a_1, a_3)\}

$\Leftarrow$ (ABD-R-21)

$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).(\mathsf{ABDeact}, a_4).$
$(\mathsf{Dial}, a_1, a_2) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_1, a_3)\}$

$\Leftarrow$ (ABD-S-16)

$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{ABDeact}, a_4).(\mathsf{ABDAdd}, a_1, a_2, a_3)$
$(\mathsf{Dial}, a_1, a_2) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_1, a_3)\}$

$\Leftarrow$ (ABD-R-23)

$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3)$
$(\mathsf{Dial}, a_1, a_2) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_1, a_3)\}$

$\Leftarrow$ (ABD-A-16)

true

6.7 Call Forward

The Call Forward feature (CF) allows the subscriber to redirect incoming calls to another directory number. The redirection is enabled by initiating an input sequence of CFEna. Similarly, the redirection is disabled by initiating an input sequence of CFDis.

$$
\begin{aligned}
I_{CF} \triangleq\ & \{\mathsf{Init}\} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFEna}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFDis}\} \times \mathbf{PhoneId}
\end{aligned}
$$

where $I_{CF} \subseteq I$.

6.7.1 Call Forward Axioms

In this section, we present a definition of CF axioms. Again, axioms POTS-A-1 to POTS-A-12 also apply as part of CF axioms. Therefore, we won't repeat them. The new A-12 for CF is:

No	Premise	Input	Output
A-12	$k \notin I_{CF} \wedge x \in I$	$x.k$	-

Table 6.34: CF Axiom for Invalid Input

No	Premise	Input	Output
A-13		(Init).(CFAct, n_1)	$\emptyset$
A-14		(OffHook, n_1)	{(DialTone, n_1)}
A-15	$n_1 \neq n_2$	(CFEna, n_1, n_2)	$\emptyset$
A-16		(OnHook, n_1)	$\emptyset$
A-17		(OffHook, n_3)	{(DialTone, n_3)}
A-18		(Dial, n_3, n_1)	{(Ring, n_3, n_2)}
A-19		(OffHook, n_2)	{(Conn, n_3, n_2)}
A-20	$n \in \{n_2, n_3\}$	(OffHook, n)	{(Disc, n_3, n_2)}
A-21	$n' \in \{n_2, n_3\} \wedge n' \neq n$	(OnHook, n')	$\emptyset$

Table 6.35: CF Axioms: Cumulative Table for a Successful Connection

Axiom CF-A-18 describes when user n_1 has forwarded all incoming calls to n_2. When n_3 attempts to call n_1, an output of {(Ring, n_3, n_2)} is received.

$$c9 \triangleq (\mathsf{Init}).(\mathsf{CFAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{CFEna}, n_1, n_2).(\mathsf{OnHook}, n_1).(\mathsf{OffHook}, n_3)$$

No	Premise	Input	Output
A-22	$unique(n_1, n_2, n_3)$	c9.(OffHook, n_2).(Dial, n_3, n_1)	{(BusyTone, n_3)}
A-23	$n_1 \neq n_3$	c9.(OffHook, n_1).(Dial, n_3, n_1)	{(BusyTone, n_3)}
A-24	$unique(n_1, n_2, n_3, n_4)$	c9.(OffHook, n_4).(Dial, n_4, n_1).(Dial, n_3, n_1)	{(BusyTone, n_3)}
A-25	$unique(n_1, n_2, n_3, n_4)$	c9.(OffHook, n_4).(Dial, n_4, n_2).(Dial, n_3, n_1)	{(BusyTone, n_3)}
A-26	(CFEna, n_1, n_2) $\in x$	x.(CFDis, n_1)	$\emptyset$
A-27	(CFAct, n_1) $\in x$	x.(CFDeact, n_1)	$\emptyset$

Table 6.36: CF Axioms: Non-Cumulative Table For Unsuccessful Connection Attempts

6.7.2 Call Forward Rules

In this section, we present the reduction and swap rules for the call forward feature.

6.7.2.1 Call Forward Reduction Rules

The reduction rules for CF include the reduction rules for POTS. Again, there is one rule that must be modified, namely POTS-R-1. The new R-1 for CF is described in table 6.37.

No	Condition	Prefix	Discard	Postfix
R-1	$k \notin I_{CF}$	x	k	x'

Table 6.37: CF Reduction Rule for Invalid Input

No	*Condition*	*Prefix*	*Discard*	*Postfix*
R-12	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	$(\mathsf{CFEna}, n_1, n_2).(\mathsf{CFDis}, n_1)$	x'
R-13	$x' \neq \varepsilon \wedge w \geq 0$	x	$(\mathsf{CFAct}, n_1).(\mathsf{CFEna}, n_1, ?)^w.(\mathsf{CFDeact}, n_1)$	x'

Table 6.38: CF Reduction Axioms: Non-Cumulative Table For Complete Connection Cycle

Rule CF-R-12 describes when user n_1 redirects all incoming calls to n_2 and immediately after disables the redirection. The state of the system does not change when this sequence is encountered, therefore CFEna and CFDis can be discarded. Similarly, when n_1 redirects several incoming calls (the "?" means any valid number) and CF is deactivated immediately after the last CFEna, the system remains unchanged as described by CF-R-13.

Next, we present the CF reduction rules for connections in progress.

No	Condition	Prefix	Discard	Postfix
R-14	$n_1 \neq n_2$	x	(CFAct, n_1)	(OffHook, n_2)
R-15	$n_1 \neq n_2$	x	(CFAct, n_1)	(Dial, n_2, n_3)
R-16	$n_1 \neq n_2$	x	(CFAct, n_1)	(OnHook, n_2)
R-17	$n_1 \neq n_2$	x	(CFAct, n_1)	(CFEna, n_2, n_3)
R-18	$n_1 \neq n_2$	x	(CFAct, n_1)	(CFAct, n_2)
R-19	$n_1 \neq n_2$	x	(CFAct, n_1)	(CFDeact, n_2)
R-20	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFEna, n_2, n_3)
R-21	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFDis, n_2)
R-22	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFAct, n_2)
R-23	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFDeact, n_2)
R-24	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CFEna, n_3, n_4)
R-25	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CFDis, n_3)
R-26	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CFAct, n_3)
R-27	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CFDeact, n_3)
R-28	$n_1 \neq n_3$	x	(CFEna, n_1, n_2)	(OffHook, n_3)
R-29	$n_1 \neq n_3$	x	(CFEna, n_1, n_2)	(Dial, n_3, n_4)
R-30	$n_1 \neq n_3$	x	(CFEna, n_1, n_2)	(OnHook, n_3)
R-31	$n_1 \neq n_3$	x	(CFEna, n_1, n_2)	(CFEna, n_3, n_4)
R-32	$n_1 \neq n_3$	x	(CFEna, n_1, n_2)	(CFDis, n_3)
R-33	$n_1 \neq n_3$	x	(CFEna, n_1, n_2)	(CFAct, n_3)
R-34	$n_1 \neq n_3$	x	(CFEna, n_1, n_2)	(CFDeact, n_3)

Table 6.39: CF Reduction Rules: Non-Cumulative Table For Connection in Progress

6.7.2.2 Call Forward Swap Rules

The following table describes the swap rules for CF The swap rules for POTS are valid swap rules for CF and therefore need not be repeated.

No	Condition	Input 1	Input 2
S-7	$n_1 \neq n_2$	(CFAct, n_1)	(OffHook, n_2)
S-8	$n_1 \neq n_2$	(CFAct, n_1)	(Dial, n_2, n_3)
S-9	$n_1 \neq n_2$	(CFAct, n_1)	(OnHook, n_2)
S-10	$n_1 \neq n_2$	(CFAct, n_1)	(CFEna, n_2, n_3)
S-11	$n_1 \neq n_2$	(CFAct, n_1)	(CFAct, n_2)
S-12	$n_1 \neq n_2$	(CFAct, n_1)	(CFDeact, n_2)
S-13	$n_1 \neq n_2$	(CFDeact, n_1)	(OffHook, n_2)
S-14	$n_1 \neq n_2$	(CFDeact, n_1)	(Dial, n_2, n_3)
S-15	$n_1 \neq n_2$	(CFDeact, n_1)	(OnHook, n_2)
S-16	$n_1 \neq n_2$	(CFDeact, n_1)	(CFEna, n_2, n_3)
S-17	$n_1 \neq n_2$	(CFDeact, n_1)	(CFDis, n_2)
S-18	$n_1 \neq n_2$	(CFDeact, n_1)	(CFDeact, n_2)
S-19	$n_1 \neq n_3$	(CFEna, n_1, n_2)	(OffHook, n_3)
S-20	$n_1 \neq n_3$	(CFEna, n_1, n_2)	(Dial, n_3, n_4)
S-21	$n_1 \neq n_3$	(CFEna, n_1, n_2)	(OnHook, n_3)
S-22	$n_1 \neq n_3$	(CFEna, n_1, n_2)	(CFEna, n_3, n_4)
S-23	$n_1 \neq n_3$	(CFEna, n_1, n_2)	(CFDis, n_3)
S-24	$n_1 \neq n_2$	(CFDis, n_1)	(OffHook, n_2)
S-25	$n_1 \neq n_3$	(CFDis, n_1)	(Dial, n_2, n_3)
S-26	$n_1 \neq n_2$	(CFDis, n_1)	(OnHook, n_2)
S-27	$n_1 \neq n_2$	(CFDis, n_1)	(CFDis, n_2)

Table 6.40: CF Swap Rules: Non-Cumulative Table

The following sequence is reduced and proven using the axioms and rules defined for CF.

PROOF.

$$(\mathsf{Init}).(\mathsf{CFAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{CFAct}, a_4)(\mathsf{OffHook}, a_4).(\mathsf{CFEna}, a_1, a_2).$$
$$(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft POTS \triangleright (\mathsf{Ring}, a_3, a_2)$$

$\Leftarrow$ (CF-S-7)

$$(\mathsf{Init}).(\mathsf{CFAct}, a_1).(\mathsf{CFAct}, a_4).(\mathsf{OffHook}, a_1).(\mathsf{OffHook}, a_4).(\mathsf{CFEna}, a_1, a_2).$$
$$(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_2)\}$$

$\Leftarrow$ (CF-R-20)

$$(\mathsf{Init}).(\mathsf{CFAct}, a_1).(\mathsf{CFAct}, a_4).(\mathsf{OffHook}, a_1).(\mathsf{CFEna}, a_1, a_2).$$
$$(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_2)\}$$

$\Leftarrow$ (CF-R-14)

$$(\mathsf{Init}).(\mathsf{CFAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{CFEna}, a_1, a_2).$$
$$(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft POTS \triangleright \{(\mathsf{Ring}, a_3, a_2)\}$$

$\Leftarrow$ (CF-A-18)

true

Chapter 7

Detecting Feature Interaction

In this chapter, we present a description of feature interaction between features.

7.1 Defining Feature Interaction

In order to deal with the feature interaction problem, it is essential to define each feature independently, and to define the combination of all features. Using the proposed relational approach in this thesis, each feature is given by a relation and the combination of all features is given by the greatest lower bound of the features.

From the specifications presented in chapter 6, the axioms and rules for POTS are included as axioms and rules for the specified telecommunication features. If a subscriber has service $POTS$ and feature F, then the subscriber expects that as long as feature F is not invoked, the telephone system reacts as prescribed by $POTS$. In addition, whenever the feature F is invoked (by submitting an input sequence that includes feature-specific inputs and can be interpreted as a request for service F) then the user expects the reaction of the system to be dictated by the specification F. The relation that represents the overall system made up of feature F on top of $POTS$ is then $F' = F \cup POTS$. This implies that a feature F is actually a service represented by the relation $F' = F \cup POTS$. When two features, A and B are combined the resulting service is represented by $S = A \sqcup B$. By virtue of the discussions of chapter 3, we know that the join of two specifications is not always defined; specifically, we know that two specifications have a join if and only if they satisfy the consistency condition, which is denoted by $cs(A, B)$ and defined by:

$$AL \cap BL = (A \cap B)L.$$

In set theoretic terms, this condition can also be written as:

$$dom(A) \cap dom(B) = dom(A \cap B).$$

Whence the definition.

5 Definition. Given two features F and G that are offered on top of POTS, and given that $F' = F \cup POTS$ and $G' = G \cup POTS$, we say that F and G cause a *feature interaction* if and only if F' and G' do not satisfy the consistency condition $cs(F', G')$. Also, we say that an input sequence h *sentisizes* the feature interaction between F and G if and only if

$$h \in dom(F') \cap dom(G') \wedge h \notin dom(F' \cap G').$$

Feature interaction occurs when a collection of features yield a different output for some input s where $s \in I_{both_features}$ where $I_{both_features}$ is the input space ranging over both features.

Our definition is capable of covering cases of interaction between n features even if none can be detected for $n-1$ features. As an example, consider the 3 features R_1, R_2, R_3 with the following behavior:

$$\begin{array}{ll} s \triangleleft R_1 \triangleright s_a & s \triangleleft R_1 \triangleright s_b \\ s \triangleleft R_2 \triangleright s_b & s \triangleleft R_2 \triangleright s_c \\ s \triangleleft R_3 \triangleright s_a & s \triangleleft R_3 \triangleright s_c \end{array}$$

If considered by pairs, these features are consistent. Otherwise, when taken altogether, they do not satisfy the consistency condition because for input s, there is no output common to each of them.

7.2 Call Waiting and Three Way Calling

The Call Waiting feature interacts with The Three Way Calling feature for the following input sequence, denoted by s_1:

$$\begin{array}{l} (\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2). \\ (\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \end{array}$$

For Three-Way Calling, the only output corresponding to the input sequence s_1 is:

$$\{(\mathsf{DialTone}, a_1), (\mathsf{Hold}, a_1, a_2)\} \ .$$

PROOF.

$$\begin{array}{l} (\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2). \\ (\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft TWC \triangleright \{(\mathsf{DialTone}, a_1), (\mathsf{Hold}, a_1, a_2)\} \end{array}$$

$\Leftarrow$ (TWC-R-1)

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).$
$(\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft TWC \triangleright \{(\mathsf{DialTone}, a_1), (\mathsf{Hold}, a_1, a_2)\}$

$\Leftarrow$ (TWC-R-26)

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).$
$(\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_3).(\mathsf{FlashHook}, a_1) \triangleleft TWC \triangleright \{(\mathsf{DialTone}, a_1), (\mathsf{Hold}, a_1, a_2)\}$

$\Leftarrow$ (TWC-R-23)

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).$
$(\mathsf{OffHook}, a_2).(\mathsf{FlashHook}, a_1) \triangleleft TWC \triangleright \{(\mathsf{DialTone}, a_1), (\mathsf{Hold}, a_1, a_2)\}$

$\Leftarrow$ (TWC-A-14)

true

On the other hand, for Call Waiting, the only output corresponding to the input sequence s_1 is:

$\{(\mathsf{Disc}, a_1, a_2), (\mathsf{Conn}, a_1, a_3)\}$.

PROOF.

$(\mathsf{Init}).(\mathsf{TWCAct}, a_1).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).$
$(\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft TWC \triangleright \{(\mathsf{DialTone}, a_1), (\mathsf{Hold}, a_1, a_2)\}$

$\Leftarrow$ (CW-R-1)

$(\mathsf{Init}).(\mathsf{CWAct}, a_1).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).$
$(\mathsf{OffHook}, a_2).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1).(\mathsf{FlashHook}, a_1) \triangleleft TWC \triangleright \{(\mathsf{Disc}, a_1, a_2), (\mathsf{Conn}, a_1, a_3)\}$

$\Leftarrow$ (CW-A-14)

true

Obviously, the *consistency condition* for Call Waiting and Three Way Calling is not satisfied for input sequence s_1. Therefore there does not exist a greatest lower bound for these two features, nor does there exist a program that satisfies both features simultaneously.

7.3 Originating Call Screening and Abbreviated Dialing

The Abbreviated Dialing feature and the Originating Call Screening interact for the following input sequence:

$$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{OCSAct}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).$$
$$(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2)$$

In this sequence, a_2 is an alias for a_3 as indicated by the $(\mathsf{ABDAdd}, a_1, a_2, a_3)$ input. However, OCS prevents calls being made between a_2 and a_3. The interaction occurs when a_1 dials a_2. For the ABD feature, the corresponding output is $\{(\mathsf{Ring}, a_1, a_3)\}$.

PROOF.

$$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{OCSAct}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).$$
$$(\mathsf{OffHook}, a_1).(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{Dial}, a_1, a_2) \triangleleft ABD \triangleright \{(\mathsf{Ring}, a_1, a_3)\}$$

$\Leftarrow$ (ABD-R-1)

$$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).$$
$$(\mathsf{OffHook}, a_1).(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{Dial}, a_1, a_2) \triangleleft ABD \triangleright \{(\mathsf{Ring}, a_1, a_3)\}$$

$\Leftarrow$ (ABD-R-1)

$$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).$$
$$(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2).\triangleleft ABD \triangleright \{(\mathsf{Ring}, a_1, a_3)\}$$

$\Leftarrow$ (ABD-A-16)

true

On the other hand, the OCS feature, yields the corresponding output $\{(\mathsf{ErrorTone}, a_1)\}$.

PROOF.

$$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{OCSAct}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).$$
$$(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$$

$\Leftarrow$ (OCS-R-1)

$(\mathsf{Init}).(\mathsf{OCSAct}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).$
$(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$

$\Leftarrow$ (OCS-R-1)

$(\mathsf{Init}).(\mathsf{OCSAct}, a_1).(\mathsf{OCSAdd}, a_1, a_2).$
$(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, a_1)\}$

$\Leftarrow$ (OCS-A-16)

true

Hence these two features do not satisfy the consistency condition.

7.4 Originating Call Screening and Call Forwarding

Call Forwarding and Originating Call Screening interact for the following input sequence:

$(\mathsf{Init}).(\mathsf{OCSAct}, a_3).(\mathsf{CFAct}, a_1)).(\mathsf{OCSAdd}, a_3, a_1)(\mathsf{OffHook}, a_1).$
$(\mathsf{CFEna}, a_1, a_2).(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1)$

In the above sequence, a_1 is a subscriber to CF while a_3 is a subscriber OCS. Any calls from a_3 to a_1 are blocked, but a_1 has forwarded all incoming calls to a_2.

The output generated by CF for this sequence is $\{(\mathsf{Ring}, a_3, a_2)\}$

PROOF.

$(\mathsf{Init}).(\mathsf{OCSAct}, a_3).(\mathsf{CFAct}, a_1)).(\mathsf{OCSAdd}, a_3, a_1)(\mathsf{OffHook}, a_1).$
$(\mathsf{CFEna}, a_1, a_2).(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft CF \triangleright \{(\mathsf{Ring}, a_3, a_3)\}$

$\Leftarrow$ (CF-R-1)

$(\mathsf{Init}).(\mathsf{CFAct}, a_1)).(\mathsf{OCSAdd}, a_3, a_1)(\mathsf{OffHook}, a_1).$
$(\mathsf{CFEna}, a_1, a_2).(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft CF \triangleright \{(\mathsf{Ring}, a_3, a_2)\}$

$\Leftarrow$ (CF-R-1)

$(\mathsf{Init}).(\mathsf{CFAct}, a_1).(\mathsf{OffHook}, a_1).$
$(\mathsf{CFEna}, a_1, a_2).(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft CF \triangleright \{(\mathsf{Ring}, a_3, a_2)\}$

$\Leftarrow$ (CF-A-18)

true

The corresponding output for OCS is $\{(\mathsf{ErrorTone}, a_3)\}$.

PROOF.

$(\mathsf{Init}).(\mathsf{OCSAct}, a_3).(\mathsf{CFAct}, a_1)).(\mathsf{OCSAdd}, a_3, a_1)(\mathsf{OffHook}, a_1).$
$(\mathsf{CFEna}, a_1, a_2).(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, a_3)\}$

$\Leftarrow$ (OCS-R-1)

$(\mathsf{Init}).(\mathsf{OCSAct}, a_3).(\mathsf{OCSAdd}, a_3, a_1)(\mathsf{OffHook}, a_1).$
$(\mathsf{CFEna}, a_1, a_2).(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, a_3)\}$

$\Leftarrow$ (OCS-R-1)

$(\mathsf{Init}).(\mathsf{OCSAct}, a_3).(\mathsf{OCSAdd}, a_3, a_1)(\mathsf{OffHook}, a_1).$
$(\mathsf{OnHook}, a_1).(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, a_3)\}$

$\Leftarrow$ (OCS-R-2)

$(\mathsf{Init}).(\mathsf{OCSAct}, a_3).(\mathsf{OCSAdd}, a_3, a_1)(\mathsf{OffHook}, a_3).(\mathsf{Dial}, a_3, a_1) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, a_3)\}$

$\Leftarrow$ (OCS-A-16)

true

Again, these two features do not produce a common output for the same input. Hence the consistency condition is not satisfied.

7.5 Categorization of the Interactions Revisited

In section 1.5, we described a technique that allows for the categorization of the feature interaction problem. The problem can be categorized by the *nature* and by the *cause* of the interaction. With the availability of this technique, we can categorize the feature interactions that were described in this chapter.

- **CW and TWC.** The cause of the interaction is limitation on network support, specifically CPE signalling. The nature of interaction is SUSC (Single-User-Single-Component). CW and TWC interpret the flashhook signal differently. CW uses the flashhook signal to toggle between two connected callers while TWC uses it to initiate a third party call.

- **OCS and ABD.** The cause of the interaction is problems in distributed systems, specifically personalized instantiation. The nature of interaction is MUSC (Multiple-User-Single-Component). OCS assigns a number to be screened if ABD has an alias for the screened number, then the number can be called.

- **OCS and CF.** The cause of the interaction is problems in distributed systems, specifically personalized instantiation. The nature of interaction is MUSC (Multiple-User-Single-Component). Again this interaction is similar to the one between ABD and OCS. OCS assigns a number to be screened but if the user can potentially forward all calls to the screened calls, it is possible to call a screened number.

Chapter 8

Automation of Feature Interaction Detection

8.1 Using a Theorem Prover to Detect Feature Interaction

In this secton, we present an automated theorem prover which will automatically detect interactions between features. The theorem prover used is called OTTER which stands for Organized Techniques for Theorem-proving and Effective Research. OTTER is a fourth-generation Argonne National Laboratory deduction system whose ancestors date back to the early 1960s. These ancestors include the TP series, AURA and LMA/ITP [16, 40].

8.1.1 OTTER Overview

We present a brief overview of the features, syntax and commands of OTTER in this section which are sufficient for our purposes of detecting feature interaction.

8.1.1.1 Introduction to OTTER

OTTER is a resolution-style program for first order logic with equality [32]. Some of the features of OTTER include the inference rules binary resolution, hyperresolution, UR-resolution and binary paramodulation. Inference rules take a small set of clauses and in turn, infer a clause. If the inferred clause is a new and useful one, it is stored in memory and may become available for succeeding inferences.

OTTER is not automatic. The user must choose inference rules, set options to control the processing of inferred clauses, and decide which input formulas or clauses are to be included in the initial set of support and which equalities are to be demodulators after the user has encoded a problem into first-order logic or into clauses. If a proof is not provided by OTTER, the user is able to try again with different initial conditions. The detection of feature interaction can be automated.

We chose OTTER as our tool because OTTER is based on first order logic. As a result, our specifications can be easily mapped to OTTER once written.

8.2 Using OTTER

In this section, we briefly discuss the syntax, semantics and commands of OTTER.

8.2.1 Syntax

Comments can be placed in the input file using the "%" symbol. Any sequence of characters appearing on the right of % are simply ignored. Comments may appear within terms.

8.2.2 Names

Names are alphanumeric strings that may contain other characters such as $ and _. Names may contain up to 50 characters and can be used for:

- constant symbols
- function symbols
- predicate symbols
- propositional variables
- regular variables

The *type* of a name may be its predicate symbol, function symbol, constant or variable and it is determined by its context. Since variables in clauses are not explicitly bound by universal quantifiers, a convention for distinguishing constants from variables are used. For clauses, variables begin with a lower-case letter. For formulas, any name may be used as a variable since variables are explicitly quantified.

Certain names are reserved by OTTER. Any binary symbol beginning with **EQ**, **Eq** or **eq** is interpreted by demodulation and paramodulation as an equality predicate. The = symbol may be used to write infix equality axioms. All symbols beginning with $ are reserved for special purposes such as evaluable functions or predicates.

8.2.3 Terms and Atoms

To determine whether a simple term is a constant or a variable depends on the context of the term. If it appears in a clause, the name determines the type. If it appears in a formula, it is a variable if it is bound by a quantifier. Generally, complex terms are written in prefix form, for

instance **f(a,b,c).**

Atoms are similar to complex rules, except that a name is also an atom (a propositional variable), and equalities may be written in infix form as $(t_1 = t_2)$ and $(t_1 != t_2)$.

8.2.4 Literals and Clauses

If a is an atom, then a and $-a$ are literals. No white spaces are permitted the negation sign and the atom. A clause is a sequence of literals separated by "|". Clauses are always terminated by a period.

8.2.5 Formulas

The following is a list of valid formulas:

1. Atoms are formulas
2. If F and G are formulas, then $(F <->G)$

   and $(F->G)$ are formulas.
3. If $F_1, ..., F_n$ are formulas, then $(F_1|...|F_n)$ and $(F_1...F_n)$ are formulas.
4. The symbols **all** and **exists** are quantifiers. If $Q_1...Q_n$ are quantifiers,

   $x_1...x_n$ are names and F is a formula, then $(Q_1x_1...Q_nx_nF)$ is a formula.
5. If F is a nonnegated formula, then $-F$ is a formula.

Table 8.1 summarizes the OTTER connectives.

Connective	OTTER Symbol
Not	-
And	&
Or	|
If and only if	<->
Implies	->

Table 8.1: OTTER Connectives

Clauses differ both syntactically and the way they are treated by OTTER. For example, the string "**x | y | z**" is a clause whereas the string "**(x | y | z)**" is a formula. When inputed,

formulas are translated into clauses.

8.2.6 Commands and the Input File

As input, OTTER takes a small set of commands, some of which indicate that a list of objects (clauses, formulas, or weight templates) follows the command. All lists of objects are terminated by **end_of_list**. The following is a partial list of the commands:

set(*flag_name*). % set a flag

clear(*flag_name*). % clear a flag

assign(*parameter_name,integer*). % assign an integer to a parameter

list(*axioms*). % read axioms in clause form

formula_list(*axioms*). % read axioms in formula form

weight_list(*weight_list_name*). % read weight templates

8.3 Analyzing Feature Interaction and OTTER

In this section, we revisit the mathematical definition of detecting feature interaction and map it into OTTER.

Let $H = I^* \wedge h \in H$ and F_1 and F_2 represent two features.

Absence of Feature Interaction:

$dom(F_1 \cap F_2) \supseteq dom(F_1) \cap dom(F_2)$

Presence of Feature Interaction:

$\exists h \in H : h \in dom(F_1) \cap dom(F_2) \wedge h \notin dom(F_1 \cap F_2)$

$\forall h : (\exists y F_1(h, y)) \Leftrightarrow dom(F_1(h))$

$\forall h : (\exists y F_2(h, y)) \Leftrightarrow dom(F_2(h))$

$\forall h, y : (F_1F_2(h, y) \Leftrightarrow F_1(h, y) \wedge F_2(h, y))$

$\forall h : (\exists y F_1F_2(h, y) \Leftrightarrow dom(F_1F_2(h))$

FeatureInteraction $\Leftrightarrow (\exists h : dom(F_1(h) \wedge dom(F_2(h)) \wedge \neg dom(F_1F_2(h))$

The above set provides a complete mathematical definition for the detection of feature interaction. Next, we map this set into OTTER and obtain the following formulas:

(all h (exists y dom($F_1(h)$) $<->$ $F_1(h,y)$)).

(all h (exists y dom($F_2(h)$) $<->$ $F_2(h,y)$)).

(all h all y ($F_1F_2(h,y)$ $<->$ $F_1(h,y)$ & $F_2(h,y)$)).

(all h (exists y dom($F_1F_2(h)$) $<->$ $F_1F_2(h,y)$)).

(exists h (FeatureInteraction(F_1,F_2) $<->$ dom($F_1(h)$) & dom($F_2(h)$) & - dom($F_1F_2(h)$))).

8.4 Mapping Specifications to OTTER

Our specification of each feature can be mapped into OTTER without much difficulty. For each feature, we define the input space and the axioms, reduction and swap rules. Next, we present the OTTER representation of the input space for POTS:

potsInput(Init).

(all X

(potsInput(OffHook(X)))).

(all X

(potsInput(OnHook(X)))).

(all X all Y

(X != Y - >

potsInput(Dial(X,Y)))).

Axioms are the easiest to transform from relational calculus to OTTER. Consider the axiom POTS-A-3.

$$\text{(POTS-A-3):}\ \frac{n_1 \neq n_2}{(\mathsf{Init}).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2) \triangleleft POTS \triangleright \{(\mathsf{Ring}, n_1, n_2)\}}$$

In OTTER this axiom is represented as:

(all N1 all N2

(N1 != N2 - >

pots([[Init],[OffHook,N1],[Dial,N1,N2]],[[Ring,N1,N2]])).

Consider the POTS reduction rule POTS-R-10.

$$\text{(POTS-R-10):}\ \frac{n_1 \neq n_2 \ \wedge\ x.(\mathsf{OffHook}, n_2) \lhd POTS \rhd y}{x.(\mathsf{OffHook}, n_1).(\mathsf{OffHook}, n_2) \lhd POTS \rhd y}$$

In OTTER, this rule is represented as:

(all N1 all N2 all X all Y

(N1 != N2 - > ([[X],pots([OffHook,N2]],[y]) - >

pots([[X],[OffHook,N1],[OffHook,N2]],[Y])))).

Finally, we transform the swap rules into OTTER. Consider the swap rule POTS-S-6. This rule has two definitions in relational calculus:

$$\text{(POTS-S-6} - 1\text{):}\ \frac{n_1 \neq n_2 \ \wedge\ n_1 \neq n_3 \ \wedge\ x' \neq \varepsilon \ \wedge\ x.(\mathsf{OnHook}, n_1).(\mathsf{Dial}, n_2, n_3).x' \lhd POTS \rhd y}{x.(\mathsf{Dial}, n_2, n_3).(\mathsf{OnHook}, n_1).x' \lhd POTS \rhd y}.$$

$$\text{(POTS-S-6} - 2\text{):}\ \frac{n_1 \neq n_2 \ \wedge\ n_1 \neq n_3 \ \wedge\ x' \neq \varepsilon \ \wedge\ x.(\mathsf{Dial}, n_2, n_3)(\mathsf{OnHook}, n_1).x' \lhd POTS \rhd y}{x.(\mathsf{OnHook}, n_1).(\mathsf{Dial}, n_2, n_3).x' \lhd POTS \rhd y}.$$

In OTTER, this rule is represented as:

(all X all Y all Xprime all N1 all N2 all N3

(Xprime != $nil & N1 != N2 & N1 != N3 - >

(pots([[X],[OnHook,N1],[Dial,N2,N3],[Xprime]],[Y]) < - >

pots([[X],[Dial,N2,N3],[OnHook,N1],[Xprime]],[Y])))).

Since the POTS axioms and rules are included in the Three Way Calling and Call Waiting axioms and rules, there is no need to redefine them for each feature. Instead, all POTS rules and axioms are automatically included for these features as specified in the following rules:

(all X all Y

(twcPrime(X,Y) < - > pots(X,Y) — twc(X,Y))).

(all X all Y

(cwPrime(X,Y) < - > pots(X,Y) — cs(X,Y))).

8.5 Validation of OTTER Specifications

Once the specifications described in chapter 6 are mapped to OTTER, we can automatically validate our specifications. In our experiment, we translated POTS relational specifications into OTTER. For simplicity, we did not include POTS swap rules. Only POTS axioms and reduction rules were coded in OTTER. Note that a slightly different implementation from the one described in the previous subsection was used. The following sequence was submitted to OTTER:

$$pots([Init(x), OffHook(y), OnHook(y), OffHook(x)], [DialTone(x)]).$$

OTTER was able to establish this sequence confirming that it is in fact valid according to POTS axioms and rules. OTTER used reduction rule POTS-R-2 to reduce this rule to:

$$pots([Init(x), OffHook(x)], [DialTone(x)]).$$

By using axiom POTS-A-2 this axiom is true. OTTER yielded the following output:

$$pots([Init(\$c1), OffHook(\$c5), OnHook(\$c5), OffHook(\$c1)], [DialTone(\$c1)]).$$

$$pots([Init(\$c1), OffHook(\$c1)], [DialTone(\$c1)]).$$

Another sequence that was submitted to OTTER is:

$$(pots([Init(z), OffHook(x), Dial(x, y), OnHook(x), OffHook(z), Dial(z, w)], [Ring(z, w)])).$$

OTTER was able to establish the sequence by using the reduction rule POTS-R-3. The following output was produced:

$$pots([Init(\$c3), OffHook(\$c5), Dial(\$c5, \$c6), OnHook(\$c5),$$

$$OffHook(\$c3), Dial(\$c3, \$c12)], [Ring(\$c3, \$c12)]).$$

By using axiom POTS-A-3 this axiom is true. OTTER yielded the following output:

$$pots([Init(\$c3), OffHook(\$c3), Dial(\$c3, \$c12)], [Ring(\$c3, \$c12)]).$$

Another interesting experiment that we attempted involved generating all valid sequences in POTS. There are an infinite number of valid sequences. This suggests that the OTTER program would never terminate. Instead, we set a flag that generated a finite number of valid sequences.

8.6 Confirming Feature Interactions

In another experiment, we automated the detection of input sequences that cause feature interaction. Two features (ABD and OCS) were specified using OTTER. Then OTTER was given input sequences known to cause feature interaction. OTTER was able to establish that these sequences cause feature interaction based on the axioms and rules of the features.

One input sequence that OTTER was given is:

$$(\mathsf{Init}).(\mathsf{ABDAct}, a_1).(\mathsf{OCSAct}, a_1).(\mathsf{ABDAdd}, a_1, a_2, a_3).$$
$$(\mathsf{OCSAdd}, a_1, a_2).(\mathsf{OffHook}, a_1).(\mathsf{Dial}, a_1, a_2)$$

OTTER was able to reduce sequence using the rules and axioms as described in section 7.3. As output, OTTER yielded the following:

$$---------PROOF---------54[]\$answer(interaction).$$

To keep the experiment simple, only the axioms and rules needed to prove the interaction were implemented in the OTTER program. In essence, this experiment was very similar to the validation experiment. The theorem prover was simply given an input sequence the causes feature interaction, and asked to prove it using axioms and rules. Any input elements that did not belong to the input space of the specified feature were discarded. After all invalid input elements were removed from the input sequence, OTTER proceeded to reduce the input sequence to an axiom for its respective feature. A different output element was established for the same input sequence hence producing feature interaction.

When experimenting with OTTER, we attempted to generate all sequences that cause feature interactions for two given features. This task was a difficult one because OTTER was unable to produce these sequences in an intelligent manner. Sequences that were generated by OTTER were feature specific. This means that OTTER would never produce input sequences that are valid for more than one feature. Even if we modified the specifications to produce valid sequences for more than one feature, the task of generating input sequences that cause a feature interaction is very difficult. This is due to the fact that OTTER, or any general-purpose theorem prover will attempt to exhaust all possibilities. In this case, computation time and memory present problems. As the number of rules and axioms increase, the number of valid sequences grow exponentially. Even if the number of rules and axioms is relatively small, the number of users in the system are theoretically infinite.

Heuristics have to be added to the theorem prover in order to achieve some kind of result. Limiting the search space or the number of users in a system may be one approach. Another approach involves making the theorem prover perform more intelligent searches by adding more rules. All in all, this problem remains unsolved for our approach.

8.7 The Swap Rule Problem

A problem arises when we try to find all interactions between any two features. OTTER attempts to find all sequences that cause an interaction by employing the axioms and rules described in chapter 5. When swap rules are applied to a given sequence this problem may present itself.

To illustrate the problem at hand, we examine the next example. Consider the input sequence :

$$(\mathsf{Init}).s_1.s_2.s_3 \triangleleft POTS \triangleright y$$

and consider the applicable swap rules:

$$\frac{p \wedge x' \neq \varepsilon \wedge x.s_1.s_2.x' \triangleleft POTS \triangleright y}{x.s_2.s_1.x' \triangleleft POTS \triangleright y} \quad \frac{p \wedge x' \neq \varepsilon \wedge x.s_2.s_1.x' \triangleleft POTS \triangleright y}{x.s_1.s_2.x' \triangleleft POTS \triangleright y}$$

If we apply the first swap rule to the sequence, we obtain

$$(\mathsf{Init}).s_2.s_1.s_3 \triangleleft POTS \triangleright y$$

Immediately after, we can apply the second swap rule to the sequence giving us

$$(\mathsf{Init}).s_1.s_2.s_3 \triangleleft POTS \triangleright y$$

This is the sequence that we initially began with. If OTTER selects these sequences one after another interchangeably, we can potentially obtain an infinite search space.

One way around this problem is to select sequences that do not depend on any swap rules and remove the swap rules entirely. In this case, feature interaction may be proven to exist using the theorem prover.

Chapter 9

Conclusion

9.1 Summary

In this thesis, we presented a method for the detection of feature interactions. This method is based on relational algebra and the refinement lattice. Each feature was specified independently of other features and then the specification was validated. Specifications as a relationship between input and output pairs was deemed inappropriate for the detection of feature interaction since this technique does not allow us to describe the external behavior of a software system. Instead, we used a technique which allowed us to describe the behavior of a system by examining the relationship between the history of inputs and the corresponding output set.

By employing an inductive style for writing the specification and presenting a tabular representation, the specifications are more readable and accurate. This technique helped to keep the set of axioms and rules minimal relative to other methodologies.

9.2 Assessment

Using relational algebra has shown to be extremely powerful for specifying features and capturing the behavior of systems [2]. In our model, writing specifications can be difficult and time consuming at first, however the specifications produced represent the concise behavior of each feature. These specifications allow us to detect any undesired interactions that may occur between features. Detecting the interactions manually is a difficult task. Since our method employs first order logic, our specifications can be easily mapped to Prolog [18] or OTTER once written. A relation can be represented by a binary predicate whose first argument is a list of inputs and second argument is a list of outputs.

9.3 Comparison to Related Work

As mentioned in the introduction, there have been a number of approaches to detect feature interactions. One approach employs the LOTOS specification language [17]. Structurally, this approach captures the underlying concepts of structuring LOTOS specifications using three constraint types: local, end-to-end and global constraints. By using this structure, new feature specifications can be integrated into existing ones by classifying their roles as caller or called and expressing their global constraint as a conjunction. Analytically, this approach is based on a reasoning mechanism. The specifier can analyze features based on their local views (ie. local constraints) where each view is an element of the global view of the system. Hence possible feature interactions are the conflicts between the feature views.

Another approach presented by [1] is to use state transition machines (STMs) to model feature behavior, where each transition is triggered by a single input event. In this approach, each feature is specified independently of other features. Multiple STMs can be composed together to form another STM representing the reachability graph. During the composition, each reachable state is tested to determine if an interaction can occur at that state.

Both of these strategies are very powerful. Since they both use reachability analysis to generate execution sequences to check for possible interactions, the state explosion problem may be encountered. Therefore search strategies must be given to the available tools. Interactions are detected by manual inspection of the traces produced from the tools.

The building block approach [30] is similar to the STM approach. Both adopt the idea of specifying features independently and using individual features as building blocks to build different scenarios. In the building block approach, procedural-level specifications capture the behavior of a feature, and behavior-level specifications describe what a feature must exibit using temporal logic formulas. Model checking is used both to verify desired behavior in the procedural specification and to check that temporal logic properties still hold after composition.

Our method has the same problems in terms of exponential growth of states as the other approaches discussed [1, 17, 30]. As the number of axioms and rules increase, the number of valid sequences grows exponentially. Each of these sequences must be analyzed to determine if the sequence causes a feature interaction. Clearly, this approach to finding interaction-causing sequences is very expensive and complex. Even when using an automated theorem prover such as OTTER, the problem is not simplified. Part of this is due to the fact that the theorem prover attempts to reduce a sequence by exhausting all possibilities.

9.4 Further Work

One obvious extension of our work is to refine or change the proposed specifications so we can obtain sequences that cause feature interactions without running into the sequence-generation combinatorial explosion. Another extension of this work may be to propose a solution once a feature interaction has been detected. Perhaps constructing a priority hierarchy of features can be an approach . When two features interact with one another, only one feature will take precedence over the other. Using this approach may be tricky when interactions occur between instances of the same feature.

One important issue to keep in mind is that POTS was treated as a feature itself. It is interesting to treat POTS as a base service and add features on to it. One problem that arises is that POTS itself may interact with some features (eg. POTS and OCS). In this situation, it is proposed to introduce a new operator which serves as an "on top of" operator denoted by $\Uparrow$. The relationship between POTS and any other feature POTS interacts with is $Feature_j \Uparrow POTS$. This implies that if a feature X interacts with POTS for some input sequence s such that $(s, s') \in X \wedge (s, s'') \in POTS$ where $s' \neq s''$, then only the output corresponding to feature X will be considered.

To illustrate this notion we provide the following example. Consider the sequence:

$$(\mathsf{Init}).(\mathsf{OCSAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{OCSAdd}, n_1, n_2).(\mathsf{Dial}, n_1, n_2)$$

The OCS feature will yield:

$$(\mathsf{Init}).(\mathsf{OCSAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{OCSAdd}, n_1, n_2).(\mathsf{Dial}, n_1, n_2) \triangleleft OCS \triangleright \{(\mathsf{ErrorTone}, n_1)\}$$

while POTS yields:

$$(\mathsf{Init}).(\mathsf{OCSAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{OCSAdd}, n_1, n_2).(\mathsf{Dial}, n_1, n_2) \triangleleft POTS \triangleright \{(\mathsf{Ring}, n_1, n_2)\}$$

Clearly there is an interaction between POTS and OCS since POTS provided $\{(\mathsf{Ring}, n_1, n_2)\}$ as output and OCS provided $\{(\mathsf{ErrorTone}, n_1)\}$ for the same input sequence. If use the $\Uparrow$ operator and specify that OCS $\Uparrow$ POTS, the output generated by OCS will be executed. This implies that when an interaction between POTS and OCS is encountered, OCS has priority.

Bibliography

[1] Au, P., Atlee, J.: *Evaluation of a State-Based Model of Feature Interactions*, Feature Interactions in Telecommunication Networks IV, IOS Press, 1997.

[2] Biebow, B., Hagelstein, J.: *Algebraic Specification of Synchronization and Errors: A Telephone Example*, Laboratoires de Marcoussis, France, Phillips Research Laboratory.

[3] Blom, J., Jonsson, B., Kempe, L.: *Using Temporal Logic for Modular Specification of Telephone Services*, Feature Interactions in Telecommunication Systems, IOS Press, 1994, pp.167-177.

[4] Blom, J., Bol, R., Kempe, L.: *Automatic Detection of Feature Interactions in Temporal Logic*, Proceedings of Third International Workshop on Feature Interactions in Telecommunication Systems, IOS Press, 1995, pp.1-19.

[5] Bostrum, M., Engstedt, M.: *Feature Interaction Detection and Resolution in the Delphi Framework*, Proceedings of Third International Workshop on Feature Interactions in Telecommunication Systems, IOS Press, 1995, pp.157-172.

[6] Boudriga, N., F. Elloumi and A. Mili. *On the Lattice of Specifications.* Submitted, 1991.

[7] Boudriga, N., Mili, A., Zalila, R.: *An Automated Tool For Specification Validation: Design and Preliminary Implementation.* Submitted, 1992

[8] Boumezbeur, R., Logrippo, L.: Specifying Telephone System in LOTOS. *IEEE Communication* (August 93) 38-45.

[9] Butler, M.J.: Feature Interaction Analysis Using Z, in *Report on Methods and Tools for Service Creation (First Version) Part II: Methods and Tools for Service Interaction Analysis*, CEC RACE project 2017 SCORE, R2017/SCO/WP2/DS/L/018/b2, January 1994. `ftp://ftp.abo.fi/pub/cs/papers/michael/fiz.ps.Z`.

[10] Cameron, E.J., Velthuijsen H.: Feature Interaction in Telecommunications System. *IEEE Communications* (August 93) 18-23.

[11] Cameron, E.J., N. Griffeth, Y.-J. Linand, M.E. Nilson, W.K. Schnure, H. Velthuijsen: A Feature Interaction Benchmark for IN and Beyond, in *Feature Interactions in Telecommunications Software Systems*, W. Bouma and H. Velthuijsen, eds, IOS Press, 1994, 1–23.

[12] Cheng, K.E., T. Ohta: *Feature Interactions in Telecommunications III.* IOS Press, 1995.

[13] Combes P., Pickin S.: *Formalization of a user View of Network and Services for Interaction Detection*, Feature Interactions in Telecommunication Systems, IOS Press, 1994, pp.120-135.

[14] Davey, B.A., H.A. Priestley: *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[15] Dworak, F.S.: *Approaches to Detecting Feature Interactions*, 1991 IEEE Globecom, pp.1371-1377.

[16] Ewing, L., Overbeek, R.: *The Automated Reasoning System ITP*, Tech. Report ANL-84/27, Argonne National Laboratory, Argonne, Ill, 1984.

[17] Faci, M., Logrippo, L.: Specifying Features and Analyzing their Interactions in a LOTOS Environment, in *Feature Interactions in Telecommunications Software Systems*, W. Bouma and H. Velthuijsen, eds, IOS Press, 1994, 136-151.

[18] Frappier M., Mili A., Desharnais J.: *Detecting Feature Interactions on Relational Specifications*, Feature Interactions in Telecommunication Networks IV, IOS Press, 1997.

[19] Gammelgaard, A., Kristensen, J.E.: *interaction Detection, a Logical Approach*, Feature Interactions in Telecommunication Systems, IOS Press, 1994, pp.178-196.

[20] Griffeth, N.D., Lin, Yow-Jian.: The Feature Interaction Problem. *Computer* (August 93) 14-18.

[21] Harada, Y., Hirakawa, T., Takenaka, T.: *A Design Support Method for Telecommunications Service Interactions*, 1991 IEEE Globecom, pp.1661-1666.

[22] Harada, Y., Hirakawa, T., Takenaka, T., Terashima N.: *A Conflict Detection Support Method For Telecommunication Service Descriptions*, IEICE Trans. Commun., Vol. E75-B No. 10, October, 1992

[23] Kelly, B., Crowther, M., King, J.: *Feature Interaction Detection Using SDL Models*, Proceedings of IEEE Globecom'94, 1994, Vol.3, pp.1857-1861.

[24] Kelly, B., Crowther, M., King, J., Masson, R., Delapeyre, J.: *Service Validation and Testing*, Proceedings of Third International Workshop on Feature Interactions in Telecommunication Systems, IOS Press, 1995, pp.173-184.

[25] Kimbler, K., Velthuijsen, H.: Feature Interaction Benchmark, KPN Research, PO Box 421, 2260 AK Leidschendam, The Netherlands, 1995, 1–13. {http://www.docs.uu.se/docs/fi/benchmark.ps.gz}

[26] Kimbler, K., Sobirk, D.: *Use Case Driven Analysis of Feature Interactions*, Feature Interactions in Telecommunication Systems, IOS Press, 1994, pp.167-177.

[27] Kimbler K.: *Towards a More Efficient Feature Interaction Analysis*, Proceedings of Third International Workshop on Feature Interactions in Telecommunication Systems, IOS Press, 1995, pp.201-211.

[28] Kuisch, E., Janmaat, R., Mulder, H., Keesmaat, I.: *A Practical Approach to Service Interactions*, IEEE Communications, 1993, pp.231-242.

[29] Lee, A.: *Formal Specification-A Key to Service Interaction Analysis*, Proceedings of SETSS, 1992, pp.62-66.

[30] Lin, F.J., Lin Y.J.: *A Building Block Approach to Detecting and Resolving Feature Interactions*, In Feature Interactions in Telecommunication Systems, 1995

[31] Makarevitch B.: *Resolving Service Interactions by Service Components*, Proceedings of Third International Workshop on Feature Interactions in Telecommunication Systems, IOS Press, 1995, pp.213-221.

[32] McCune,W.: *OTTER 2.O Users Guide*, Tech. Report ANL-90/9, Argonne National Laboratory, Argonne, IL, March 1990

[33] Mili, A., Desharnais, J., Mili, F.: *Computer Program Construction*, Oxford University Press, 1994.

[34] Mili, A., Noureddine, B., Mili, F.: *On the Lattice of Specifications: Applications to a Specification Methodology*, Formal Aspects of Computing(1992), 544-571.

[35] Mili A., Boudriga, N., Zalila, R., *Didon: System For Specification Validation*, Butterworth-Heinemann, 1991.

[36] Möller, B.: Relations as a Program Development Language. In *Constructing Programs from Specifications*, B. Möller, ed., Proc. IFIP TC 2/WG 2.1 Pacific Grove, CA, USA, May 13–16, North-Holland (1991) 373–397.

[37] Nakamura M., Kakuda Y., Kikuno T.: *Petri-Net Based Detection Method for Non-Deterministic Feature Interactions and its Experimental Evaluation*, Feature Interactions in Telecommunication Networks IV, IOS Press, 1997.

[38] Nakamura M., Kakuda Y., Kikuno T.: *Analyzing Non-determinism in Telecommunication Services Using P-invariant of Petri-Net Model*, Proc. of IEEE INFOCOM'97, April, 1997

[39] Tsang, S., Magill, E.H., Kelly, B.: Feature Interactions in Multimedia Systems. *The Feature Interaction Problem in Networked Multimedia Services*, EPSRC No. GR/K 72995, October, 1996

[40] Smith, B.: *An Incarnation of AURA*, Tech. Report ANL-88-2, Argonne National Laboratory, Argonne, Ill, 1988.

[41] Tsang, S., Magill, E.H.: Detecting Feature Interactions in the Intelligent Network. *University of Strthclyde, UK* (November 93)

[42] Verstraete, R.A., Decuypere, H.J.M.: *A Strategy for Studying Services and Service Interactions in Intelligent Networks*, 1990 IEEE Globecom, pp.1650-1654.

[43] Wakahara, Y., Fujioka, M., Kikuta, H., Yagi, H., Sakai, S.I.: *A Method for Detecting Service Interactions*, IEEE Communications, 1993, pp.32-37.