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.

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.

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

UMI®

Université d'Ottawa · University of Ottawa

A Formal Specification Based Approach to Feature Interaction : Application with Prolog

by
Serge Colle

A thesis
presented to the School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of Masters in Computer Science*

School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada
Oct 23, 1998

* The Masters program in Computer Science is a joint program with Carleton University, administered by the Ottawa-Carleton Institute for Computer Science

Merci a Rolland et Rita
... que le seigneur soit avec vous.

Abstract

The rapid growth of new services for telecommunication and multimedia systems is being challenged by the *feature interaction problem*. A number of approaches for managing feature interaction have been proposed. A relational method for specifying software systems and detecting features interactions is presented. The method proposed allows features to be specified independently as well as detecting interactions between two or more features. This method is based on the lattice of relational specifications; a system is defined by the conjunction of the features and feature interaction is detected when such a conjunction does not exists. After reading this thesis a reader should be able to specify a simple system and detect interactions between it's features.

Contents

List of Figures

List of Tables

Acknowledgements

I wish to express my gratitude to all my friends, family member and colleagues. To Dr. Ali Mili, who supervised my thesis and introduce me to the world of formal languages. Without his guidance, advice and patience this paper would not have been possible. To Dr. Marc Frappier, who was extremely helpful through numerous discussions. To Elias Rafoul, who shared my frustration and my determination as a fellow researcher and for making this enjoyable as a friend. To my family members, Rita, Rolland, Alain and Patricia, who never stop believing in me and kept motivating me througout the years. To Jeff Campbell and Antonio Mirijello, for their valuable help. Last but not least to Gloria who reminded me that there is more to life than work and made me enjoy life.

Chapter 1

Introduction

1.1 Background

One of the main concerns of software engineers in the telecommunication field is software *extensibility* and *compatibility*. This is because some system which is going to be around for a very long period of time may require frequent changes and additions to its functionality. The feature interaction problem [10, 12] can be simply defined as an unwanted interference between two or more features. It represents an obstacle to the development and implementation of telecommunication, multimedia, Intelligent Network, or any system which features are added incrementally. This problem makes it hard to add a new feature rapidly.

The problem of resolving any specific interaction is not necessarily difficult but the number of features is so large that it is not practical to resolve them one by one as the features are being added to the system. At this stage, detecting feature interaction is very costly in manpower and time, and thus may cause a loss in revenue. Many interactions are not detected until the testing phase, and some are not detected until the system has been installed at the customer site. The latter will not only cost a loss in revenue, but may cost the company a future sale.

1.2 Existing Solution

Existing solutions to the problem are either implemented at the specification stage [13] of the features software lifecycle or while the features are actually running [15]. Two categories of formal specification exist: the model-oriented approach [9, 1], and the behavioral approach [3]. The former has a general tendency to be either too abstract or so complicated that it is virtually unmanageable. The latter is also generally abstract. This can be achieved by directly specifying Labeled Transition Systems (LTSs) [2] or Relation Specification [3]. The following representations are commonly used:

- *SDL* [5, 6, 7]
- *LOTOS* [13, 14]
- *state machine and rule based representations* [19, 21, 22, 8]
- *extended specifications* [26, 30]
- *temporal logic* [28, 27, 29]

1.3 Relation Algebra Approach

Detecting feature interaction at the specification stage has a great advantage. Interactions are detected before the design, implementation and integration of the feature to the system, therefore permitting the specifier to resolve this interaction in the initial stage of the software engineering life cycle. This reduces both the cost and the faults.

In this thesis, we analyze and detect feature interactions in PSTN (Public Switch Telephone Network) using relational specifications [16]. A method based on the refinement lattice of specifications is introduced. This method allows features to be specified independantly and the detection of FI at the validation phase. The combination of all the features can be given by the least upper bound of all features.

In our relational framework, two or more features interact when the refinement lattice has no least upper bound. In other words feature interaction exists if there exists an input sequence where no one output can satisfy all features, thus proving that the least upper bound does not exist. This is accomplished by writting inductive definitions of relations using axioms in predicate calculus.

In this thesis, we employ POTS (Plain Old Telephone System) as an illustrative example. Telecommunication features such as call waiting (CW), three-way calling (TWC), originating call screening (OCS), abbreviated dialing (ABD) also known as speed call, and call forward on busy (CFB), will also be specified. These telecommunication features are not inclusive but should be enough for the purpose of this thesis.. In the following we give an informal definition of the features used.

- POTS :- Is the basic telephone service, which allows the converstaion between two users and the billing for services.

- CW :- Is a feature, which generates a special tone, to indicate to the user already engaged in a conversation, that a second call is waiting for a reply.
- CFB :- Is a feature that allows a user to transfer the calls he receives to a specific location, when he is busy.
- TWC :- Is a feature which allows a user already engaged in a conversation to add a third user.
- OCS :- Is a feature which blocks any attempt to connect to the numbers that the user wishes to block.
- ABD :- Is a feature which allows the users to store a number in a memory location address with one of the digits 2-9.

1.4 Research Contribution

In this thesis we will generate specifications for the Plain Old Telephone System, and 5 independant features which consist of call waiting, call forward busy, three way calling, originating call screening and abreviated dialing. These specifications will then be validated manually, ensuring that the specifications are complete. It will be demonstrated that applying our proposed method, to our relational specifications will detect interaction between features. Some experiments using Prolog will be execute to show that interaction between feature can be detected automatically.

The research contribution in this thesis can be summarized as follows

- Relational specification of POTS, CW, CFB, TWC, OCS and ABD.
- Validation of these specification.
- Validating the specification using Prolog.
- Detection of feature interactions between the specified features.
- Detecting Feature Interaction automatically using Prolog.
- Categorization of the feature interaction found.

The relational specification and the properties used to validate these specification can be found in Chapter 5. Chapter 6 define and describes how feature interaction can be detected from our specification. In Chapter 7 a method was propose and used to validate the specification with the help of Prolog. In the same chapter some experiment were run to show that with certain modifications, feature interaction could be detected automatically.

Part I

Background

Chapter 2

Mathematical Background

Our objective in this chapter is to present the notions of sets and relations. Most of this chapter is taken directly from [16]. The understanding of these two notions is essential to comprehend the material discussed in this thesis. Relations will be presented with into two distinct points of view: first, as sets (of pairs); second an algebraic structure defined by operations *(union, intersection, inversion, complement, composition and ordering (inclusion)*. While most of this chapter is describing widely used notations it also introduces some notations which are specific to this thesis. We assume that the reader is familiar with predicate logic and discrete mathematics.

2.1 Sets

A *set* is a well-defined group of objects. It is described by some rule that makes it possible to tell whether or not a particular object is in the set. The set of all natural numbers less than 11 consists of {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. Sets can be defined by listing all their elements within braces, such as {New York, Los Angeles, Chicago}, or by giving a description that determines what is in the set and what is not: "An ellipse is the set of all points in a plane such that the sum of the distance to two fixed points in the plane is a constant." As we can see, sets do not always have to contain numbers. They can be comprised of many different types of objects, each of which is called an *element*.

2.1.1 Defining Sets

The sets we will be using throughout this thesis are:

- **natural**, the set of natural numbers,
- **integer**, the set of integers,

- **boolean**, the set {true, false}
- other sets specific to the system defined in Chapter 5.

2.1.2 Operations on Sets

Let S and S' be two distinct sets. Given a set S, the *power* set of S is the set of all subset of the set S. The *power set* of S is denoted by ρ(S). The *Cartesian product* of set S by set S', denoted by $S \times S'$, is the set of all ordered pairs (s, s') where $s \in S$ and $s' \in S'$ such that $S \times S' = \{(s, s') | s \in S \wedge s' \in S'\}$. *Ordered pairs* are an ordered collection containing two elements, also known as ordered n-tuples, where n = 2. The *union* of the sets S and S', denoted by $S \cup S'$, is the set that contains those elements that are either in S or in S' or in both. The *intersection* of the sets S and S' denoted $S \cap S'$, is the set containing those elements in both S and S'. The *difference* of S and S', denoted by $S - S'$, is the set containing those elements that are in S but not in S'. The difference of S and S' is also called the complement of S' with respect to S. What is the complement? Let us define L to be the universal set. The *complement* of the set S, denoted by $\overline{S}$, is the complement of S with respect to L. In other words, the complement of the set S is $L - S$. The *precedence* of these operations from highest to lowest is: the *power set*, the *complement*, the *Cartesian product*, the *intersection* followed by the *difference* and the *union*.

2.2 Relations

Relations have been used for specification and design in domains such as: *theory of data types, relational specification, program construction*, and *constructing program from specification.* In this paper we will study binary relations from two different aspects: *the elementary theory of relations* and the *calculus of relations*.

2.2.1 The Elementary Theory of Binary Relations

The elementary theory of binary relations defines the relations as set of pairs and involves the set theoretic nature of relations. A *binary relation* on a set S is a subset of the Cartesian product $S \times S$. Let (s, s') be an element of relation R where s is the antecedent in R and s' is the image in 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 *domain* of relation R consists of the set of all of its antecedents and is denoted by $dom(R)$; formally

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

The set of all the images of relation R is called the *range* of relation R, and is denoted by $rng(R)$; formally,

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

Among the constant relations on a set S are the *universal relation*(S x S) denoted by $L(S)$, the *empty relation* by $\phi(S)$, the *identity relation* ($\{(s, s') | s' = s\}$) denoted by $I(S)$, and the *diversity relation* ($L(S) - I(S)$) denoted by V(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 homegeneity, the symbol to represent the empty set is $\emptyset$ and the symbol to represent the empty relation is ϕ.

2.2.2 The Calculus of Relations

The *calculus of relations* is defined as a set $\mathbf{R}$ of relations provided with the binary operations of *union*, denoted by $\cup$, *intersection*, denoted by $\cap$, *difference*, denoted by $-$, and the unary operations of *complement* denoted by placing $\bar{x}$ over the operand. Elements of $\mathbf{R}$ are ordered by *inclusion*, that we denote by $\subseteq$. L, I and ϕ are special elements contained in set $\mathbf{R}$.

The *precedence* of these operation from highest to lowest is:*complement*, followed by *intersection*, the *difference* and the *union*.

2.2.3 Operations on Relations

In complement to the theoretic operations defined in the previous section, we will define other operations that can be used to combine relations. The *theory of relations* will be used to define these operations while we will rely on the *calculus of relations* in our discussion of these relational properties. 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.

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

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

The *product* of relation R by relation R' is the relation denoted by $R \circ R'$ (sometimes it is denoted simply as RR') and defined by

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

The i^{th} *power* of relation R, for a natural number i, is the relation denoted by R^i and defined by :

$$R^0 = I$$
$$\text{for } i \geq 1 : R^i = R \circ R^{i-1}.$$

The *transitive closure* of relation R is the relation denoted by R^+ and defined by

$$R^+ = \{(s, s') | \exists i \geq 1 : (s, s') \in R^i\}.$$

The *relextive transitive closure* of relation R is the relation denoted by R^* and defined by

$$R^* = I \cup R^+ \text{ which equals } R^i = \{(s, s') | \exists i \geq 0 : (s, s') \in R^i\}.$$

The *sum* of relation R with relation R' is the relation denoted by $R + R'$ and defined by

$$R + R' = \{(s, s') | \forall t : (s, t) \in R \vee (t, s') \in R'\} = \overline{\overline{R}\overline{R'}}.$$

A *transitive root* of relation R is a relation T whose reflexive transitive closure equals R. If no proper subset of T is a transitive root of R, we say that T is an *irreducible transitive root* of R. A relation may have more than one unique transitive or irreducible transitive root.

2.2.4 Properties of Relations

2.2.4.1 Totality Properties

Let R be a relation on a set S is said to be *total* if and only if $dom(R) = S$; which can be written algebraically as $R \circ L = L$. A relation R is said to be *surjective* if and only if $\hat{R}$ is total; which can be written algebraically as $L \circ R = L$.

2.2.4.2 Equivalence Properties

Let R be a relation on set S. Relation R is said to be *reflexive* if and only if $I \subseteq R$. Relation R is said to be *symmetric* if and only if $\hat{R} \subseteq R$. Relation R is said to be *transitive* if an only if $R^2 \subseteq R$. Relation R is said to be an *equivalence* if and only if it is reflexive, symmetric, and transitive. The *equivalence class* of element $s \in S$ *modulo* an equivalence relation R is the set $s \bullet R$.

2.2.4.3 Ordering Properties

Let R be a relation on set S. Relation R is said to be *antisymmetric* if and only if $R \cap \hat{R} \subseteq I$. Relation R is said to be *asymmetric* if and only if $R \cap \hat{R} = \phi$. Relations R is said to be *connected* if and only if $V \subseteq R \cup \hat{R}$. Relation R is said to be *strongly connected* if and only if $L \subseteq R \cup \hat{R}$.

Relation R is said to be a *partial ordering* if and only if it is reflexive, antisymmetric and transitive. Relation R is said to be a *total ordering* if and only if it is a partial ordering and is strongly connected.

2.2.4.4 Lattice Structuring

Let R be a partial ordering on set S and let A be a subset of S.
An element a in A is said to be minimal in A with respect to R if and only if $R \bullet a \cap A = \phi$. Given two elements a and b of S, an upper bound of a and b is an element of $a \bullet R \cap b \bullet R$. A *least upper bound* of a and b is a minimal element of the set $a \bullet R \cap b \bullet R$. Similarly, we can define the motions of *maximal* elements, *lower bounds* and *greatest lower bounds*.

Definition 2.1 *Given a set S and a partial ordering R, we say that (S, R) is a lattice if and only if for any pair of elements a and b there exists a unique least upper bound and a unique greatest lower bound.*

The least upper bound of a and b (sometimes call the *join*) is denoted by $a \sqcup b$ and the greatest lower bound of a and b (sometimes call the *meet*) is denoted by $a \sqcap b$.

Chapter 3

Relations: A tool for specifying

Specifying a program has been an important issue to researchers in the last decades. Reducing the economic side of constructing a program, and increasing its reliability are among the reasons why specifications have been an important issue in the scientific world.

3.1 The Nature of Specifications

The Webster's New Collegiate Dictionary gives two definitions to the word specification:

1. The act of specifying.

2. A detailed and precise presentation of something.

The former refers to the process while the latter refers to the product. The distinction between process and product is crucial, as it is a key feature of several software engineering goal structures.

3.1.1 The Product

The *product* is the final specification. This should be a detailed description of the requirement imposed by the client. From this point on we will refer to the person wishing to acquire the software product as the client. For the purpose of this thesis, we will restrict ourselves to *functional requirements*.

The purposes of the specifications as a product are [32, 16]:

- It is a contract between the client and the programmer. This contract will be used to validate the delivered software program.

- It is the designer's reference and plan to design the program with the correct functionality.

- It is also part of the user's manual since it describes the functionality and the operating convention.

3.1.2 The Process

Traditionally [32, 16], 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. The former can be informally defined as what gets done when and the latter can be informally define as who does what. We apply the same premise to the specification process, and submit that this process includes two phases and two activities. In our view, the specification process involves three partners, which we consider in turn below.

- *The user group.* This group is familiar with the application domain (of the projected software product), but is not assumed to be familiar with the art of writing formal specifications, nor is assumed to have a clear idea of all the requirements wished to impose.
- *The specifier group.* This group has some general knowledge of the application domain, and is quite familiar with the art of eliciting requirements from the user group and capturing them faithfully in some formal notation.
- *The verification and validation group.* This group has some general knowledge of the application domain, and is familiar with the art of eliciting (arbitrarily partial, arbitrarily weak) requirements from the user group and capturing them in some formal notation.

3.1.2.1 The Phases

Concerning the meaning of the terms *phase* (the subject of this section) and *activity* (the subject of the next section), we adopt the definitions of B. Boehm [25]. The lifecycle of the specification process includes two phases which we discuss in turn below.

1. *The specification generation phase.* During this phase, two activities proceed concurrently:
   - By interaction with the user group, the specifier group elicits requirements information and pins it down in the formal specification document.
   - By interaction with the user group, the verification and validation group elicits redundant requirements information and pins it down independently (from the specifier group) in some formal document. We refer to each piece of such redundant information as a *property.*

The redundant information elicited by the verification and validation group from the user group deals with two dual aspects of the specification, namely its completeness with respect to the user requirements, and its minimality.

2. *The specification validation phase.* During this phase the verification and validation group confront the redundant information generated in the previous phase against the specification generated by the specifier group to establish diagnostics on the specification at hand.

3.1.2.2 The Activities

The activity of generating the specification in the first phase of the lifecycle given above is essentially of the same nature as the activity of generating redundant properties: in both cases the task is to map informal user requirements into formal notation. There is, however, a key difference: while specifications must be faithful to the user's intention, properties may be arbitrarily partial with respect to it. Two kinds of properties are derived during this phase; we discuss them in turn below.

- *Completeness properties.* To generate a completeness property, the verification and validation group asks the following question: *what feature does the user want, that the specifier is likely to have overlooked?* Once such a property is derived, the verification and validation group poses the following premise: *In order to be complete, the specification must capture all the information represented in the property.*

- *Minimality properties.* To generate a minimality property, the verification and validation group asks the following question: *what feature is the specifier likely to have captured in his specification, when in fact the user does not want it?* Once such a property is derived, the verification and validation group poses the following premise: *In order to be minimal, the specification must not capture the information represented in the property.*

The lifecycle of the specification is summarized in the figure 3.1.

3.2 Properties of Specifications

3.2.1 Properties of the Product

In order to fulfill its function in an effective manner, a specification must satisfy a number of properties. We distinguish in this thesis between properties of the product and properties of the process. Below we identify three properties of the product:

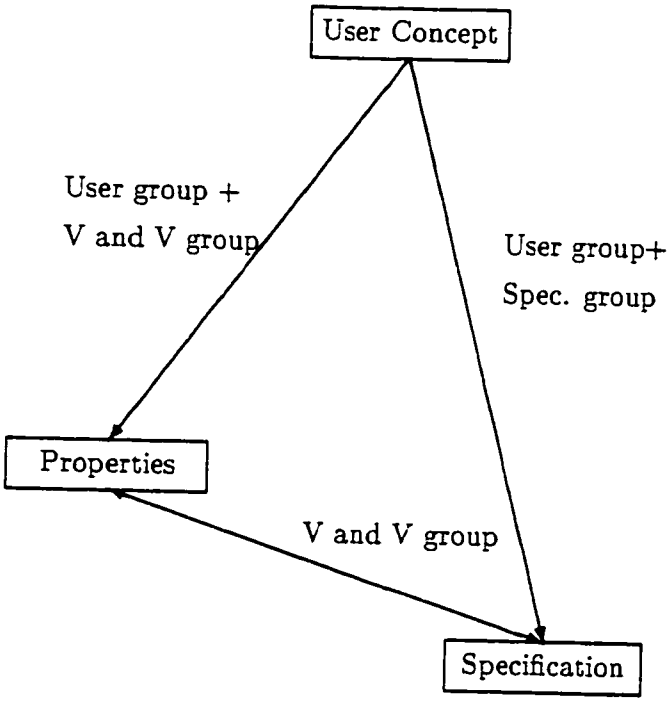

Figure 3.1: The Specification Process

- *Formality.* A specification is formal if and only if the language used is formally defined. This property is crucial if the specification product is to be used as a contract. It is also essential for proving and validating purposes.

- *Clarity.* A specification must be easy to read, analyze and comprehend. While programs are processed by computers, specification are read by human being, hence they are useless if they are unreadable.

- *Abstraction.* A specification should never contain any implementation information. In reality it should be abstract in that aspect. Specification should contain only the information of the user's problem and not the information about the solution to the problem. This property is well knowed in the software engineering world. Moreover all design decision should be postponed as late as possible.

3.2.2 Properties of the Process

While properties of the product are intrinsic to the product, and can be established by looking at the product alone, properties of the process deal with the relationship of the specification product with the original intention of the user. In an effort to synthesize the earlier lists of

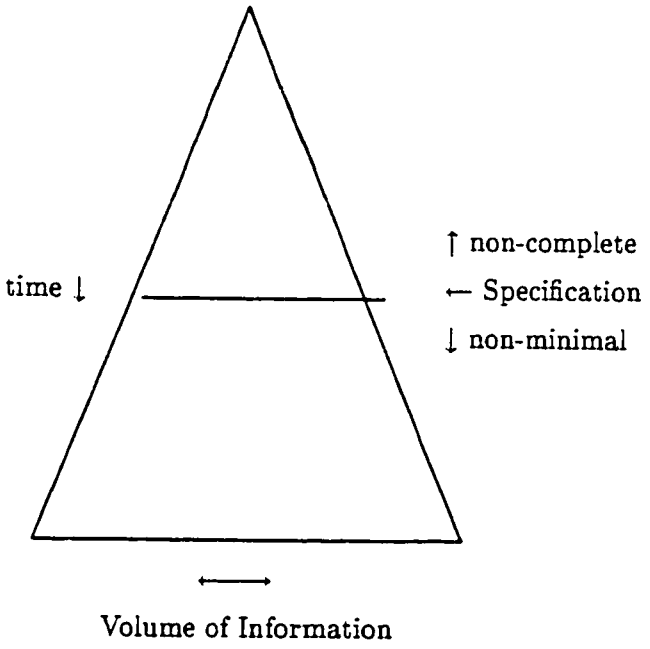

Figure 3.2: The Volume of Information: Specification to Design

properties [36], we have identified two properties of the specification as a process. We present them briefly below:

- *Completeness.* A specification is complete if and only if it captures *all* the user requirements. Completeness cannot be proven. It can only be progressively and painstakingly established using redundancy between requirements elicited (from the user group) and the specifier group and requirements information elicited (from the user group) by the verification group.

- *Minimality.* A specification is minimal if and only if it captures ***nothing but*** the user requirements. A specification is said to be minimal if it contains only those requirements that have been proposed by the user.

Figure 3.2 shows where a non-complete specification lies with respect to good specifications, which are both complete and minimal.

3.3 A Specification Model : Specifying with Relations

We let the *input space* of a program be the set of inputs which may be processed, the *output space* be the set of all possible results the program may return, and the *internal space* be the set of intermediate values which may be needed while the program is executing.

A program performs a series of mappings, first from the input space to its internal space and then from its internal space to the output space. 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 pairs (s, s') such that some property holds between s and s'.

Such a specification, that we represent by the pair (S, R), prescribes requirements on a program whose space is S. The pair (s, s') is in R if and only if the user considers that s is a possible initial state, and s' is a correct final state for initial state s. There may be more than one distinct s' for any initial state. A specification that only has one s' for any s is called a deterministic specification while if more than one s' is possible it is called a non-deterministic specification. As an example, we consider the space S defined by

$$S = \mathbf{Natural} x \mathbf{Natural} x \mathbf{Natural}$$

and we let the user requirement be to preserve the sum of a and b while decreasing c. Then a complete and minimal formal specification for this requirement is:

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

Examples of complete but non-minimal specifications include the following:

$$R_1 = \{(s, s') | a' = a \wedge b' = b \wedge c' < c\}.$$
$$R_2 = \{(s, s') | a' + b' = a + b \wedge c' < c - 1\}.$$

Relation R_1 is not minimal because it is $a + b$ which must stay constant and not a and b individually. The second relation is not minimal since c does not have to be decremented by 1 but may be decremented by any natural value.

Examples of incomplete specifications include the following:

$R_3 = \{(s, s') | a' + b' = a + b \wedge c' \leq c\}$.
$R_4 = \{(s, s') | a \geq b \wedge a' + b' = a + b \wedge c < c'\}$.

Relation R_3 is not complete because it allows more outputs than the user authorized ($c' = c$). Relation R_4 is also not complete since it does not cover all the input cases (such as $a < b$).

3.4 The Semi-lattice of Specification

3.4.1 The Refinement Ordering

It was established that a specification can be represented by relations. In this section we will see that specifications can be compared according to the amount of their input-output information.

Definition 3.1 *Relation R is said to be* refines *than relation R', denoted $R \sqsubseteq R'$, iff $R'L \subseteq RL \wedge R'L \cap R \subseteq R'$.*

If and only if R is more define than of R', we may also say that R *refines* R', or that it is a *refinement* of R'. This definition can be interpreted 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 deemed *more-defined* than another if it knows more input and is more precise in the input-output mapping. This ordering relation is crucial in the study of program specification, verification, validation, design and construction. In this thesis, we will only be interested in the program specification, verification and validation aspect.

There exist two instances where the *more-defined ordering* is of a simple form:

- if $dom(R) = dom(R')$ then $R \sqsubseteq R' \Leftrightarrow R \subseteq R'$,
- if R and R' are deterministic then $R \sqsubseteq R' \Leftrightarrow R' \subseteq R$.

3.4.2 Ordering Properties

It is trivial to show that relation R is more-defined than R. The relation *more-defined* is reflexive. Using the antisymmetry and transitivity of the inclusion relation ($\subseteq$), it can be deduced that the *more-defined* relation is also antisymmetric and transitive. Because it is reflexive, antisymmetric and transitive, relation *more-defined* is a partial ordering. We may refer to this as the *refinement ordering*.

3.4.3 Lattice Properties

A *lattice* is an ordered set X such that the greatest lower bound and the least upper bound of any non-empty finite subset Y of X exist. Given two relation R and R' on S, we are interested in whether they have a *least upper bound*, $R \sqcup R'$ and a *greatest lower bound*, $R \sqcap R'$. We have the following proposition which we present without proof(due to [17].

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 $(R \cap R')L = RL \cap R'L$, we find that two relations R and R' have a greatest lower bound if and only if, for those elements that belong to both $dom(R) \wedge dom(R')$, 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 relation R and R' contradict each other. Condition (cs) means in effect that R and R' are not mutually inconsistent; we call this the *consistency condition*, whence its abbreviation.

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

Proposition 3.2 provides that the least upper bound of two relations R and R' represents the total input-output information contained in them; these can be summed only if they are mutually consistent, hence the consistency condition must hold for the least upper bound to exist.

Corollary 3.1 *If R and R' have a least upper bound,they must satisfy the following condition:*

1 $\quad \forall s : s \in dom(R) \wedge s \in dom(R') \Rightarrow \exists s' : (s, s') \in R \wedge (s, s') \in R'$.

We may generalize this definition to an arbitrary set $A \triangleq \{R_j : j \in J\}$ of relations, where J is a non-empty index set.

$$cs(A) \triangleq \forall s : (\forall j : j \in J : s \in dom(R_j)) \Rightarrow \exists s' : (\forall j : j \in J : (s, s') \in R_j)$$

Proposition 3.3 *Any pair of relations R and R' have a greatest lower bound, which is given by the expression $R \sqcap R' = RL \cap R'L \cap (R \cup R')$.*

The greatest lower bound of two relation R and R' shows that there is information duplication, i.e. the information that is common to both R and R'. We have found in effect that the more defined the greatest lower bound of two relations, the more information they have in common. The least upper bound and greatest lower bound take a simple form under two conditions, that we discuss in turn below(under the form of corollaries).

Corollary 3.2 *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*

$$R \sqcup R' = R \cap R',$$

also under the same condition

$$R \sqcap R' = R \cup R'.$$

Example. Let S be the space defined by:

$$S = \{x, y, z\} \text{ where } x, y, z \text{ are } \textbf{Natural}$$

and let R and R' be the following relations

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

Then

$$\begin{aligned} & R \sqcup R' \\ & = R \cup R' \\ & = \{(s, s') | x \geq y \wedge z' = x\} \cup \{(s, s') | x \leq y \wedge z' = y\} \\ & = \{(s, s') | z' = max(x, y)\}. \end{aligned}$$

And

$$\begin{aligned} & R \sqcap R' \\ & = R \cap R' \\ & = \{(s, s') | x = y \wedge z' = x\}. \end{aligned}$$

Both relations R and R' provide the case when $x = y$ ($x \geq y$ and $x \leq y$ provided respectively by relation R and R' and the condition where the same output is $z' = x = y$.

Corollary 3.3 *If R and R' satisfy the condition $RL = R'L = (R \cap R')L$ then R and R' has a least upper bound which is*

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

Also under the same condition,

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

Corollary 3.3 shows that the least upper bound of R and R' is their intersection.

Corollary 3.4 *Let R and R' be two specification. Then a program satisfying $R \sqcup R'$ satisfies both R and R'.*

Example.
Let S be the set of integer arrays of size n, for some $n > 0$ and let R and R' be the following relations

$$R = \{(s, s') | perm(s, s')\},$$

where $perm(s, s')$ means that s' is a permutation of s.

$$R' = \{(s, s') | ascending(s')\},$$

where $ascending(s')$ means that s' is arranged in ascending order. Then we can easily verify that R and R' satisfy the condition , and

$$\begin{aligned} & R \sqcup R' \\ & = R \cap R' \\ & \{(s, s') | perm(s, s') \wedge ascending(s')\} \\ & \{(s, s') | s' = sort(s)\}, \end{aligned}$$

where $sort(s)$ is the ascending permutation of array s. This relation does indeed capture the total information carried by R and R'. Now let's compute the greatest lower bound.

$$\begin{aligned} & R \sqcap R' \\ & = R \cup R' \\ & = \{s, s') | perm(s, s') \vee ascending(s')\} \end{aligned}$$

Definition 3.2 *Program p is said to be* totally correct *with respect to (or satisfies) a specification R, if for any input sequence $s \in dom(R)$, program p will terminates and produce an output s' such that $(s, s') \in R$; for any input sequence $s \notin dom(R)$, program p may deliver any input sequence, or it may fail to terminate.*

This notion of total correctness is captured by the refinement relation.

Part II

Specifying Telecommunication Services

Chapter 4

Specifying Dynamic Software Systems

4.1 A Specification Model

A program processes input data and returns the result as output data. For the sake of *formality*, we use the language of mathematics to represent program specification. For the sake of *abstraction*, we concentrate on relations between inputs and outputs. For the sake of generality, we require that our specification model captures not only those software products whose output is dependent on their input, but also those whose output is dependent on some internal state; the internal state typically represents a summary of past invocations of the software product. In order to represent internal states while satisfying the criterion of abstraction, we propose to let specifications be defined as relations from the set of *input sequences* to outputs. The specification of a software system may be descibed by a relation from the sequences of inputs to outputs. We can specify a software system by using three components:

- **Input Space** I, defined as the set of all possible inputs of the system.
- **Output Space** O, defined as the set of all possible outputs of the system.
- **Input-Output Relation** R, defined as 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.

4.2 Axiomatic Representation of Specifications

It is helpful to have an axiomatic approach [31, 32] to the generation of a specification R. You can specify a deductive system using either of the two following formulas

1. formulas of predicate logic

2. formulas of the form $(Q, y) \in R$, where $Q \in \mathsf{I}^*, y \in \mathsf{O}$, and $R \subseteq \mathsf{I}^* \mathrm{x} \mathsf{O}$, where I^* is all possible inputs sequence and O is all possible output.

Given a deductive system , let R be the relation defined by:

The pair (Q, y) is in R if and only if the formula '$(Q, y) \in R$' is a theorem of system D

Before starting to define axioms and rules lets consider the language of first order logic with the usual connectives in the following binding order from highest to lowest: $\neg, \wedge, \vee, \{\Leftarrow, \Rightarrow\}, \Leftrightarrow, \{\forall, \exists, \wedge_j, \vee_j\}$ The binding order can be modified by using parenthesis. Tuples are denoted with $(, ...,)$ and sets are denoted with $\{, ..., \}$.

Using this language, an inductive definition of a relation is given by a set of axioms of the form $(\wedge^n{}_{i=1} A_i) \Leftrightarrow B$, where n is natural number, and A_i and B are meta-variables for predicates. For readability we will use the notation followed below:

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

Axioms

For each trivial input sequence, say Q_i, construct an axiom, say a_i, which expresses that the input sequence Q_i yields output y_i if the premises P_i holds. Namely:

$$a_i \frac{P_i}{Q \triangleleft R \triangleright y}$$

where $Q \triangleleft R \triangleright y$ is an abbreviation of $(Q, y) \in R$.

Rules

Typically, more that one input sequence Q_i will yield output y_i. Other elements of I^* share the same image. This is precisely the role of the rules, which define other sequences for each y_i.

$$r_i \frac{P_i \wedge Q_i \triangleleft R \triangleright y}{Q' \triangleleft R \triangleright y}$$

In other words, the rules define the abstract level of relation R, while the axioms map each level set into its image.

Illustrative Example

The axiomatic representation given above can be illustrated by means of the specification of a queue data type.

What is a queue?

1. A queue will have 4 inputs which are procedures init, push, pop and print and the output will be either a set of values of some type itemtype or an error message.
2. The main purpose of this data type is to store values of type itemtype, given as parameters to the procedure push(), and delete them by means of procedure pop in the same order as they arrive, if procedure pop is invoked while the queue is empty it is merely ignored.
3. The function print() prints the oldest stored value which was not yet deleted. If the queue is empty this function will return an error message
4. Whenever procedure init is invoked, the queue is initialized again and all past inputs are ignored; it is assumed that all input sequences include at least one occurrence of init.

It is posed:

$$\mathsf{I} = \{\mathsf{push}(value), \mathsf{pop}, \mathsf{init}, \mathsf{print}\}$$

$$\mathsf{O} = itemtype \cup \{\phi\} \cup \{error\}$$

where itemtype represents the set (type) of items to be stored in the queue.

Axioms

A_1 Push Print Axiom:

$$\frac{\{\mathsf{init}, \mathsf{pop}\} \notin Q'}{\mathsf{init}.\mathsf{push}(t).Q'.\mathsf{print} \lhd R \rhd t} \ ,$$

A_2 Init Print Axiom:

$$\overline{Q.\mathsf{init}.\mathsf{print} \lhd R \rhd error} \ ,$$

A_3 Init Axiom:

$$\overline{Q.\mathsf{init} \lhd R \rhd \phi} \ ,$$

A_4 Init Pop Axiom:

$$\overline{Q.\mathsf{init}.\mathsf{pop} \lhd R \rhd \phi} \ ,$$

A_5 Init Push Axiom:

$$\frac{n \geq 1}{Q.\mathsf{init}.\mathsf{push}(-)^n \lhd R \rhd \phi} \ .$$

The A_1 axiom provides that the function *print* will print the oldest value in the queue. The A_2 axiom provides that the function *print* will return an error if submitted while the queue is empty. Axioms A_3, A_4 and A_5 merely inform us that *init*, *push* and *pop* are procedures with no visible output. We have define ϕ to represent that there is no visible output and that we don't care.

Axioms

R_1 Init Rule:

$$\frac{Q'.\mathsf{init}.Q \lhd R \rhd y}{Q''.\mathsf{init}.Q \lhd R \rhd y} ,$$

R_2 Init Pop Rule

$$\frac{Q.\mathsf{init}.Q^+ \lhd R \rhd y \wedge Q^+ \in I^+ \wedge \mathsf{init} \notin Q^+}{Q.\mathsf{init}.\mathsf{pop}.Q^+ \lhd R \rhd y} ,$$

R_3 Push Pop:

$$\frac{Q.\mathsf{push}(-)^n.Q^+ \lhd R \rhd y \wedge Q^+ \in I^+ \wedge \mathsf{init} \notin Q^+}{Q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.Q^+ \lhd R \rhd y} ,$$

R_4 Null Top Rule:

$$\frac{Q.Q^+ \lhd R \rhd y \wedge Q^+ \in I^+ \wedge init \notin Q^+}{Q.\mathsf{print}.Q^+ \lhd R \rhd y} .$$

The first rule provides that everything that has occurred before the *init* is forgotten. The second rule provides that when it is called on an empty queue, that procedure pop is merely ignored; the input Q^+ is assumed to be non-empty, since the empty case is handled by the init pop axiom (A_4). It is also required that Q^+ does not contain the input *init* since it is handled by applying different rules and axioms. The push pop rules express the behavior of a queue which follows the first-in-first-out principle, or in other words, that the pop procedure deletes the oldest value entered in the queue. The null print rule provides that function print is ignored as soon as it is serviced.

4.3 Mathematics of Specification Validation

How can the specifier verify whether the specification meets all of the user's requirements? Actually it is impossible to be 100 percent certain that all of the requirement has been captured (completeness); nor that we did not add more than what the user had in mind(minimality). This section discusses how to check both the completeness and the minimality of a specification

once generated.

The validation process is accomplished in two steps [16]:

- some property, which captures some aspect of completeness or minimality is derived.
- this property is matched against the generated specification to ensure/check completeness or minimality.

4.3.1 Checking Completeness

To generate properties for this steps , the verification and validation group asks the question: what aspects of the user requirement is the specifier likely to have overlooked? Another question which could be asked is what properties should be expected from the specification if it is complete? Let V be the relation that captures some such aspect. Then the verification and validation group must ensure that all information found in V is found in the generated specification R. In other words we must verify that relation R is more-defined than relation V.

Definition 4.1 *Specification R is complete with respect to property V if and only if R is more-defined than V.*

Axiomatic specification example

Consider the specification of the queue data type given in the previous section and the following completeness property: the specification must indicate that when the same element is pushed twice, there are indeed two copies of it stored. The relation generated to reflect this property is,

$$V = \{(Q,t) | \exists Q' : Q = Q'.init.push(t).push(t).pop.print\}$$

If the relation defined by specification queue can be established to be more-defined than V, then it can be claimed that queue is complete with respect to V. Assume that after inspecting the user requirements, the following completeness property was generated:

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

$$Q.init.pop.push(a).push(T)^n.print.print$$

is the value of a, where T is any dynamic arbitrary variable. This property would find errors such as:

- if the specifier did not consider that pop has no visible output following the init,
- if the specifier made the pop cancel a push that follows it rather than precede it,

- if the specifier did not understand that the print operation has no effect on the state of the system,
- if the specifier did not understand the property and behavior of the queue.

This property can be written under the form of the following relation:

$$V = \{Q, y) | Q = Q'.init.pop.push(a).push(T)^n.print.print\}$$

The completeness of the queue specification with respect to this property can be established by proving that the relation R is more-defined than V. This states that the specification is complete with respect to V. However as stated earlier it does not imply that the specification is complete with respect to the user requirements but instead state that the specification covers most input-output possibilities.

4.3.2 Checking Minimality

To generate properties for this step, the verification group asks the question: what features of the user requirement is the specifier likely to over specify. Sometimes the specifier will specify something he thinks the user wanted but in reality the user really does not.

Let W, be a relation generated by the verification and validation group, where W contains information that R is not supposed to carry, hence R should not be more-defined than W..

Definition 4.2 *A specification R is said to be minimal with respect to property W if and only if R is not more-defined than W.*

Axiomatic specification validation example

Lets consider the specification of the queue and the following minimality property: the specifier may have though that an error message should be returned when the pop operation is performed on an empty queue. However the user just wanted the pop operation to be ignored. The relation generated to reflect this property is,

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

If R is more-defined that W, the queue is not minimal with respect to W. Hence to check minimality it is sufficient to show that R is not more-defined than W.

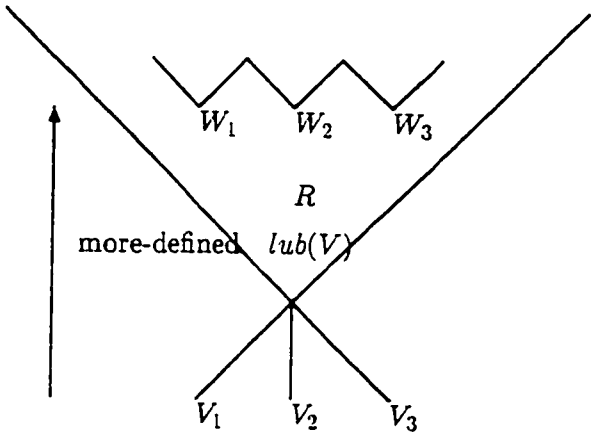

Figure 4.1: The Range of Complete and Minimal Specifications

4.3.3 Complete and Minimal Specification

We may think of the completeness properties (V) and the minimality properties (W) as boundary of a band within which the specification may lie. As the properties become more restrictive the band will grow narrower. The band is also inversely proportional to the amount of properties generated.

The specification must be higher than $\mathbf{V}$ where $\mathbf{V} = lub(V_1, ..., V_3)$ to be complete and must be lower than the W_i to be minimal. Therefore to be both complete and minimal, the specification R must be within the the lines delimited with W_i and V_i.

4.4 Semantic Definition

As seen in the previous sections, deriving the axioms and rules of a data type from its requirements text is possible. It can also be observed that the axioms and rules represent the desired properties of the data type at hand. The axiomatic representation is a tool for the stepwise construction of data type specifications.

The question that arises then is: given an axiomatic specification defined by axioms and rules, what relation does this specification define from I^* to O? Also , how do we extract this relation from the axioms and rules? These questions can be answered by following two step [31, 18, 32], which are:

- we give a relational interpretation of the axioms and rules,

- we present a least fixpoint definition of them.

4.4.1 Relational Interpretation

This section will discuss relational interpretation with the help of mainly examples. We consider the Push – Print axiom, presented in Section 4.2,

A_1 Push Print Axiom:

$$\frac{\mathsf{init} \in Q \wedge \{\mathsf{init}, \mathsf{pop}\} \notin Q'}{\mathsf{init}.\mathsf{push}(t).Q'.\mathsf{print} \lhd R \rhd t} ,$$

and we let A be the following relation:

$$A = \{(Q, y) | \exists q : Q = q.init.print \wedge y = error\}.$$

Informally, the axiom provides that all pairs whose first element ends with Init.Print and whose second element is *error* are in relation R; now, A is precisely the set of all those pairs. Hence the axiom can be written as

$$A \subseteq R.$$

Now, let consider the Push – Pop rule, presented in Section 4.2,

R_3 Push Pop:

$$\frac{(Q.\mathsf{push}(-)^n.Q^+ \lhd R \rhd y \wedge Q^+ \in \mathsf{I}^+ \wedge \mathsf{init} \notin Q^+}{Q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.Q^+ \lhd R \rhd y} ,$$

and we let B be the following relation on I^*

$$B = \{(Q, Q') | \exists q, q^+ : Q = q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.q^+ \wedge Q' = q.\mathsf{push}(-)^n.q^+ \wedge \mathsf{init} \notin q + \wedge q + \in \mathsf{I}^+\}$$

The rule can be interpreted logically as:

$$q.\mathsf{push}(-)^n.q^+ \wedge \mathsf{init} \notin q^+ \wedge q^+ \in \mathsf{I}^+ \Rightarrow (q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.q^+, y) \in R$$

Using the definition of relation B, we can rewrite this as:

$$(q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.q^+, q.q^+) \in B \wedge (q.q^+, y) \in R \Rightarrow (q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.q^+, y) \in R.$$

Using the definition of relative product, we can rewrite this as

$$(q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.q^+, q.q^+) \in B \circ R \Rightarrow (q.\mathsf{push}(t).\mathsf{push}(-)^n.\mathsf{pop}.q^+, y) \in R.$$

Hence:

$$(\forall Q \forall y, Q \in dom(B) \wedge (Q, y) \in B \circ R \Rightarrow (Q, y) \in R.$$

Because $Q \in dom(B)$ is equivalent to $(Q, Q) \in I(dom(B)))$, this can be easily rewritten as :

$$I(dom(B)) \circ B \circ R \subseteq R.$$

Because $I(dom(B)) \circ B = B$, we get

$$B \circ R \subseteq R$$

From the experience of [40], it has been found that most axioms and rules have such relations.

To summarize then, the relation R we are trying to define must fulfill the following inclusions:

- $A \subseteq R$
- $B \circ B \subseteq R.$

It is also found that these two inequalities could be reduce into one :

$$A \cup B \circ R \subseteq R.$$

If we denote by $\mathcal{F}$ the relation

$$\mathcal{F} = \{(R, R') | R' = A \cup B \circ R\},$$

then the equation above can be rewritten as:

$$\mathcal{F}(R) \subseteq R.,$$

4.4.2 Fixpoint Semantics

The definition of the axiomatic system that we introduced (in Section 4.2 to represent relation R is:

"The pair (Q, y) is in R iff the formula $(Q, y) \in R$ is a theorem of system D"

Because of the iff clause in this definition, R contains every pairs (Q, y) such that the formula $(Q, y) \in R)$ is a theorem of D, and nothing else. Therefore R must be chosen as small as possible with respect to inclusion, subject to the inequality:

$$\mathcal{F}(R) \subseteq R.,$$

Proposition 4.1 *Given an axiomatic specification d, let $\mathcal{F}(R)$ be the functional $\mathcal{F}(R) = A \cup B \circ R$, where A and B are the relations defined by the axioms and the rules of d. Then the relation defined by the axiomatic system d is: the* smallest relation R *such that $\mathcal{F}(R) \subseteq R$.*

Now, a theorem by Tarski, entitled the lattice-theoretical fixpoint theorem provides:

Theorem 4.1 *Let $U = (A, \leq)$ be a complete lattice and $\mathcal{F}$ be a monotonic function on A. Then the least element x such that $\mathcal{F}(x) \leq x$ is the same as the least element x such that $\mathcal{F}(x) = x$*

The set used is a set of relations, ordered with the relational inclusion which is in fact a lattice. Also it is a complete lattice since every subset has a least upper bound and a greatest lower bound. Since both the union and the relative product are increasing, therefore $\mathcal{F}$ is also increasing. Using the theorem 4.1 we can reformulate proposition 4.1 and get:

Proposition 4.2 *Given an axiomatic specification d, let $\mathcal{F}$ be the functional $\mathcal{F}(R) = A \cup B \circ R$, where A and B are the relations defined by the axioms and rules of d. Then the relation defined by the axiomatic system d is: the* smallest relation *such that $\mathcal{F}(R) = R$.*

Since we know that functional $\mathcal{F}$ defined as it is above is ($\mathcal{F}(R) = A \cup B \circ R$) is indeed continuous, we can apply Kleen's theorem

Theorem 4.2 *Every continuous functional $\mathcal{F}$ has a least fixpoint, which is the least upper bound ($\{\mathcal{F}^i(\phi)\}$).*

and deduce the following proposition

Proposition 4.3 *Given an axiomatic specification d, let $\mathcal{F}$ be the functional $\mathcal{F}(R) = A \cup B \circ R$, where A and B are the relations defined by the axioms and rules of d. Then the relation defined by the axiomatic system d exists, is unique, and is defined by: $lub(\{\mathcal{F}^i(\phi)\})$.*

This is the relation that we consider defined by the axiom that generates A and the rule that generates B. **Illustrative Examples**

Below is a partial expression for $lub(\{\mathcal{F}^i(\phi)\})$:

$lub(\{\mathcal{F}^0(\phi)\}) = \phi$

$lub(\{\mathcal{F}^1(\phi)\}) = A.$

$lub(\{\mathcal{F}^2(\phi)\}) = A \cup B \circ A.$

$lub(\{\mathcal{F}^3(\phi)\}) = A \cup B \circ A \cup B^2 \circ A.$

... ...

$lub(\{\mathcal{F}^k(\phi)\}) = A \cup B \circ A \cup B^2 \circ A ... \cup B^k \circ A.$

... ...

Also, we give some elements of the consecutive elements of this sequence. Because R is the lub of all these relations, it contains all the pairs we will show below:

Element(s) of $\mathcal{F}^0(\phi)$: none

Element(s) of $\mathcal{F}^1(\phi)$: (q.Init.Print, $error$).

Element(s) of $\mathcal{F}^2(\phi)$: (q.Init.Print, $error$),
(q.Init.Push.Pop.Print, $error$).

Element(s) of $\mathcal{F}^3(\phi)$: $(q.\mathsf{Init.Print}, error)$.
$(q.\mathsf{Init.Push}(t).\mathsf{Pop.Print}, error)$.
$(q.\mathsf{Init.Push}(t).\mathsf{Pop.Push}(t).\mathsf{Pop.Print}, error)$.
$(q.\mathsf{Init.Push}(t).\mathsf{Push}(t).\mathsf{Pop.Pop.Print}, error)$.

Chapter 5

A Telephone System Specification

The traditional case study for feature interaction detection is a telephone system augmented with a certain number of features. In this section, we provide a relational specification of a Plain Old Telephone System (POTS). This system offers the basic services to its customers. In other words a user may be connected to one other user at most. In our framework, we define one input space I, that ranges over the entire system, including the features. To simplify our specification we also define an input space $I_{FEATURE}$ for each feature. This approach helps us discard unrelated input sequences. In this Chapter, we also provide specification for the three way calling (TWC), call waiting (CW), originating call screening (OCS), abbreviated dialing (ABD) and call forward busy (CFB). The system is comprised of communication switches and communication network. The input is the data transferred from the telephone to the system and the output is the data transferred from the system to the telephone. We impose no limit on the number of users and the number of connections. We assume that the system produces an output immediately after receiving an input, and before receiving the next input.

5.1 The Space of the Specification

An input $(\mathsf{OffHook}, n)$, where n represents the telephone number which is used to identify the calling party is sent to the network whenever the user picks up the handset. For simplicity, we assume that the identification number, which uniquely identifies a telephone, is the same as the telephone number unless otherwise mentioned. Input (OnHook, n) is sent when user n hangs up the telephone, typically to terminate a call.

Every user (telephone) must be initialized. The first (OnHook, n) informs the telephone system that user n is connected to the network and is ready and able to receive a call. Hence an input (OnHook, n_i) should be entered for all i users. We will refer to this as the initializing step and for simplicity and readability we will define (Init, n) to be the first onhook performed

by user n. Furthermore, adding a multitude of (Init, n) at the beginning of each rule may add complexity and make it hard to read. For this purpose we let

$$(\mathsf{Init}, n_1, ..., n_i) \quad \triangleq \quad (\mathsf{Init}, n_1).....(\mathsf{Init}, n_i).$$

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 will be described in more detail later on, since the outcome is dependant on the specific features.
Therefore we consider the following input space for our telephone system.

$$\begin{aligned}
\mathsf{I} \quad \triangleq \quad & \{\mathsf{Init}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OffHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OnHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{Dial}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{FlashHook}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{TWCAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{TWCDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CWAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CWDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDAdd}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \times \mathbf{PhoneId} \cup \\
& \{\mathsf{ABDDel}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSAdd}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{OCSDel}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFBAct}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFBDeact}\} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFBEn}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup \\
& \{\mathsf{CFBDis}\} \times \mathbf{PhoneId}
\end{aligned}$$

The output space of the telephone system is defined as follows.

$$\mathsf{O} \quad \triangleq \quad \wp(\{\mathsf{DialTone}\} \times \mathbf{PhoneId} \cup$$

$$\{\mathsf{Ring}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup$$
$$\{\mathsf{BusyTone}\} \times \mathbf{PhoneId} \cup$$
$$\{\mathsf{RingTone}\} \times \mathbf{PhoneId} \cup$$
$$\{\mathsf{ErrorTone}\} \times \mathbf{PhoneId} \cup$$
$$\{\mathsf{WaitTone}\} \times \mathbf{PhoneId} \cup$$
$$\{\mathsf{Conn}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup$$
$$\{\mathsf{Disc}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId} \cup$$
$$\{\mathsf{Hold}\} \times \mathbf{PhoneId} \times \mathbf{PhoneId})$$

Hence, an output is a *set* of commands. The visible outputs are singletons, however for some features such as TWC and CW, outputs may be sets of several elements. Notice the $\wp$ at the beginning of the $\triangleq$ of the output space. This indicates to us that the output may be a set of any combination of the commands including the empty set which indicates that the system produces no visible output and is denoted by ϕ.

Output element $(\mathsf{DialTone}, n)$ corresponds to the dial tone that user n hears after the PSTN has processed the input $(\mathsf{OffHook}, n)$ which it receives after the user has picked up the handset and places 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 unit is busy. 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 (OnHook, n_i) has been sent and where i is 1 or 2. The output $(\mathsf{Hold}, n_1, n_2)$ will suspend the connection between n_1 and n_2, without terminating down the connection. Then n_1 can connect to another user. The inputs of the form XXXAct and XXXDeact where XXX can be TWC,CW,ABD,OCS,CFB are the inputs to activate and deactivate repsectively the feature XXX. Some features need special input such as $(\mathsf{ABDAdd}, n_1, n_2, n_3)$, $(\mathsf{ABDDel}, n_1, n_2)$, $(\mathsf{OCSAdd}, n_1, n_2)$, $(\mathsf{OCSDel}, n_1, n_2)$, $(\mathsf{CFBEn}, n_1, n_2)$ and (CFBDis, n_1) which will be describe later. The input $(\mathsf{FlashHook}, n_1)$ is also used by some features, which will also be describe later.

5.2 POTS Specification

When writing the specification of POTS, we have found that there are two types of rules: basic rules and complex rules, which we will refer as *axioms* and *rules* respectively. The latter one can be divide into two classes: reduction rules and swaps rule. Moreover, each class can be

written using a *tabular* format which is a lot more readable than the usual plain mathematical notation of premise over conclusion. We will also be able to reuse the same table for different features.

Even though the input space of POTS is I, POTS only uses a subset of these inputs. To be concise we define

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

to be the input used by POTS, where $I_{POTS} \subseteq I$.
In the next sections, we provide our POTS specification under a tabular format.

5.2.1 POTS Axioms

We will present two tables for the axioms. Table 5.1 provides the axioms when a successful connection between two users is established , while Table 5.2 describes the cases when the connection is unsuccessful.

No	Premise	Input	Output
A-1		(Init, n_1)	ϕ
A-2		$(\mathsf{OffHook}, n_1)$	$\{(\mathsf{DialTone}, n_1)\}$
A-3	$n_1 \neq n_2$	$(\mathsf{Init}, n_2).(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{Ring}, n_1, n_2)\}$
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')	ϕ

Table 5.1: Cumulative Table for a Successful Connection

Each table has a different interpretation in terms of axioms. We say that Table 5.1 is *cumulative*. Each line in the table represents one axiom. Let p_n, s_n and s'_n be respectively the premise, the input and the output of line n. Let p_n be the concatenation of all premises above and including line n and let s_n the concatenation of all input above and including line n. Then the axiom associated to 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 to lines A-3 and A-5 are, respectively:

$$\text{(POTS-A-3):}\ \frac{n_1 \neq n_2}{(\mathsf{Init}, n1, n2).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2) \triangleleft POTS \triangleright \{(\mathsf{Ring}, n_1, n_2)\}}\ ,$$

$$\text{(POTS-A-5):}\ \frac{n_1 \neq n_2 \ \wedge\ n \in \{n_1, n_2\}}{(\mathsf{Init}, n1, n2).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_2).(\mathsf{OnHook}, n)}$$

$$\triangleleft POTS \triangleright \{(\mathsf{Disc}, n_1, n_2)\}\ .$$

Our tabular format avoids the unnecessary repetition of the premises and the input sequences from one axiom to another. Also, it provides a quick and clear visualization of the behavior for prototypical input sequences, i.e. the establishment of a successful connection.

$$\mathsf{cl}(n_1, ..., n_t) \ \hat{=}\ (\mathsf{Init}(n1, ..., n_t).(\mathsf{OffHook}, n_1)$$

No	Premise	Input	Output
A-7		$\mathsf{Init}, n_1).(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{Error}, n_1)\}$
A-8	$n_1 \neq n_2$	$\mathsf{cl}(n_1, n_2).(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{BusyTone}, n_1)\}$
A-9		$\mathsf{cl}(n_1, n_2).(\mathsf{Dial}, n_1, n_1)$	$\{(\mathsf{BusyTone}, n_1)\}$
A-10	$\mathit{distinct}(n_1, n_2, n_3)$	$\mathsf{cl}(n_1, n_2, n_3).(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_1, n_3).(\mathsf{Dial}, n_2, n_3)$	$\{(\mathsf{BusyTone}, n_2)\}$
A-11		$\mathsf{cl}(n_1).(\mathsf{OnHook}, n_1)$	ϕ
A-12	$n_1 \neq n_2$	$\mathsf{cl}(n_1, n_2, n_3).(\mathsf{Dial}, n_1, n_2).(\mathsf{Dial}, n_1, n_3)$	ϕ
A-13	$K \notin I_R$	$x.K$	U

Table 5.2: Unsuccessful Connection Attempts

Table 5.2 is not cumulative. The axiom associated with line n is simply given by:

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

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

$$\text{(POTS-A-8):}\ \frac{n_1 \neq n_2}{(\mathsf{Init}, n_1, n_2).(\mathsf{OffHook}, n_1).(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_1, n_2) \triangleleft POTS \triangleright \{(\mathsf{BusyTone}, n_1)\}}\ .$$

As previously mentioned, we will specify POTS as a normal feature, and add all of its functionality to every feature defined in this thesis. Thus POTS will be a subset of all other features. Even though all features will have an axiom and a rule for inputs which are not defined, an invalid input for one feature may be valid for another. This is why the condition for axiom 13 is $K \notin I_R$ which informs the reader of the specification that input K is not an element of the domain of the relation. The variable R must be replaced by the relation in question which in this case is $POTS$. Therefore, the axiom associated with line A-13 is:

$$\text{(POTS-A-13):}\ \frac{K \notin I_{POTS}}{x.K \triangleleft POTS \triangleright \phi}\ .$$

5.2.2 POTS Rules

The axioms provide the output for basic input sequences. As mentioned previously, for more complex input sequences, we use rules. The next two section will discuss the *reduction* rules and the *swap* rules.

5.2.2.1 POTS Reduction Rules

Reduction rules are rules that are used to eliminate symbols from the input sequence. By recursively applying reduction axioms, a complex input sequence is reduced to a base case input sequence, and then the corresponding output may be derived.

We provide two reduction tables: Table 5.3 illustrates how completed cycles are removed from the input sequence, while Table 5.4 deals with the removal of incomplete cycle.

No	Condition	Prefix	Discard	Postfix
R-1	$K \notin I_R \wedge x' \neq \varepsilon$	x	K	x'
R-2	$x' \neq \varepsilon$	x	(OffHook, n_1).(OnHook, n_1)	x'
R-3	$x' \neq \varepsilon$	x	(OffHook, n_1).(Dial, n_1, n_2).(OnHook , n_1)	x'
R-4	$x' \neq \varepsilon$	x.(Dial, n_1, n_2)	(Dial, n_1, n_3)	x'
R-5	$x' \neq \varepsilon$	x.(Dial, n_1, n_2)	(Dial, n_3, n_2)	x'

Table 5.3: Reduction Axioms - Complete Cycle

No	Condition	Prefix	Discard	Postfix
R-6		x	(OffHook, n_1)	(Init, n_2)
R-7		x	(OffHook, n_1)	(OffHook, n_2)
R-8	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	(OffHook, n_1)	(Dial, n_2, n_3)
R-9	$n_1 \neq n_2$	x	(OffHook, n_1)	(OnHook, n_2)
R-10	$n_2 \neq n_3$	x	(Dial, n_1, n_2)	(Init, n_3)
R-11	$n_2 \neq n_3$	x	(Dial, n_1, n_2)	(OffHook, n_3)
R-12	$n_1 \neq n_3 \wedge n_2 \neq n_4$	x	(Dial, n_1, n_2)	(Dial, n_3, n_4)
R-13		x	(Dial, n_1, n_2)	(OnHook, n_3)
R-14	$n_1 \neq n_2$	x	(Init, n_1)	(OffHook, n_2)
R-15	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	(Init, n_1)	(Dial, n_2, n_3)
R-16	$n_1 \neq n_2$	x	(Init, n_1)	(OnHook, n_2)

Table 5.4: Reduction Rules - Connection in Progress

These two tables are interpreted in terms of rules 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 fresh variable with respect to line n. Then the associated rule is:

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

For instance, the rule associated to lines R-2 and R-11 are, respectively:

$$\text{(POTS-R-2):}\ \frac{x.x' \lhd POTS \rhd y}{x.(\mathsf{OffHook}, n1).(\mathsf{OnHook}, n1).x' \lhd POTS \rhd y}\ .$$

$$\text{(POTS-R-11):}\ \frac{n_2 \neq n_3\ \wedge\ x.(\mathsf{OffHook}, n_3) \lhd POTS \rhd y}{x.(\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_3) \lhd POTS \rhd y}\ .$$

This rule provides that the behavior of POTS when input element $(\mathsf{OffHook}, n_3)$ is received, does not depend on input element $(\mathsf{Dial}, n_1, n_2)$ if the latter is the immediate predecessor of the former, and if $n_2 \neq n_3$. Also note that the latter may not be the last part of the input sequence. To illustrate the application of reduction rules, assume we want to prove that:

$$(\mathsf{OffHook}, u_1).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_2, u_3).(\mathsf{Dial}, u_1, u_4) \lhd POTS \rhd (\mathsf{Ring}, u_1, u_4)\ ,$$

where the u_i are distinct constants and also note that for simplicity sake we did not initialize any of the user. We will need to remove all the initializing part of the axioms.

PROOF.

$$\begin{array}{ll} & (\mathsf{OffHook}, u_1).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_2, u_3).(\mathsf{Dial}, u_1, u_4) \lhd POTS \rhd (\mathsf{Ring}, u_1, u_4) \\ \Leftarrow & \qquad \langle\ \text{POTS-R-12}\ \rangle \\ & (\mathsf{OffHook}, u_1).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_1, u_4) \lhd POTS \rhd (\mathsf{Ring}, u_1, u_4) \\ \Leftarrow & \qquad \langle\ \text{POTS-R-8}\ \rangle \\ & (\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_4) \lhd POTS \rhd (\mathsf{Ring}, u_1, u_4) \\ \Leftarrow & \qquad \langle\ \text{POTS-A-3}\ \rangle \\ & \mathbf{true} \end{array}$$

5.2.2.2 Swap Rules

In the previous section we used reduction rules to reduce complex rules into axioms. However not all complex sequences can be reduced by applying our reduction rules. By applying swap rules a complex sequence can be transformed into an equivalent sequence. Basically swap rules are swapping input elements with each other, and are applied until a reduction axiom may be applied. Table 5.5 provides the list of swap axioms for POTS.

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 axioms are:

$$\frac{p\ \wedge\ x' \neq \varepsilon\ \wedge\ x.s_1.s_2.x' \lhd POTS \rhd y}{x.s_2.s_1.x' \lhd POTS \rhd y} \qquad \frac{p\ \wedge\ x' \neq \varepsilon\ \wedge\ x.s_2.s_1.x' \lhd POTS \rhd y}{x.s_1.s_2.x' \lhd POTS \rhd y}\ .$$

No	Condition	Prefix	Input 1	Input 2	Postfix
S-1		x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{OffHook}, n_2)$	x'
S-2	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{Dial}, n_2, n_3)$	x'
S-3	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	(OnHook, n_2)	x'
S-4	$n_1 \neq n_3 \wedge n_2 \neq n_4$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{Dial}, n_3, n_4)$	x'
S-5		x	(OnHook, n_1)	(OnHook, n_2)	x'
S-6	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	(OnHook, n_1)	$(\mathsf{Dial}, n_2, n_3)$	x'
S-7	$n_1 \neq n_2$	x	(Init, n_1)	$(\mathsf{OffHook}, n_2)$	x'
S-8	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	(Init, n_1)	$(\mathsf{Dial}, n_2, n_3)$	x'
S-9	$n_1 \neq n_2$	x	(Init, n_1)	(OnHook, n_2)	x
S-10	$n_1 \neq n_2$	x	(Init, n_1)	(Init, n_2)	x'

Table 5.5: Swap Axioms

However, since we are using the least fixpoint semantic, which states that the conclusion can only be true if and only if the premise holds. Since the least upper bound is used it can be deduced that the premise will hold if and only if the conclusion is true, therefore only one rule needs to be generated per line.

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

This rule provides that OnHook and Dial may be swapped interchangeably as long as they do not occur at the end of the input sequence. We will use the same input sequence that was used to prove the reduction rule. However, this time will initialize the user, therefore assume that we want to prove:

$$(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{OffHook}, u_1).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_2, u_3).(\mathsf{Dial}, u_1, u_4) \triangleleft \mathit{POTS} \triangleright (\mathsf{Ring}, u_1, u_4)\ ,$$

PROOF.

$$\begin{aligned}
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{OffHook}, u_1).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_2, u_3).(\mathsf{Dial}, u_1, u_4) \\
&\quad \triangleleft \mathit{POTS} \triangleright (\mathsf{Ring}, u_1, u_4) \\
\Leftarrow\ &\quad \langle\ \text{POTS-R-12}\ \rangle \\
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{OffHook}, u_1).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_1, u_4) \triangleleft \mathit{POTS} \triangleright (\mathsf{Ring}, u_1, u_4) \\
\Leftarrow\ &\quad \langle\ \text{POTS-R-8}\ \rangle \\
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_4) \triangleleft \mathit{POTS} \triangleright (\mathsf{Ring}, u_1, u_4) \\
\Leftarrow\ &\quad \langle\ \text{POTS-S-3}\ \rangle \\
&(\mathsf{Init}, u_1, u_2, u_4).(\mathsf{OffHook}, u_1).(\mathsf{Init}, u_3).(\mathsf{Dial}, u_1, u_4) \triangleleft \mathit{POTS} \triangleright (\mathsf{Ring}, u_1, u_4) \\
\Leftarrow\ &\quad \langle\ \text{POTS-R-15}\ \rangle \\
&(\mathsf{Init}, u_1, u_2, u_4).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_4) \triangleleft \mathit{POTS} \triangleright (\mathsf{Ring}, u_1, u_4) \\
\Leftarrow\ &\quad \langle\ \text{POTS-S-3}\ \rangle \\
&(\mathsf{Init}, u_1, u_4).(\mathsf{OffHook}, u_1).(\mathsf{Init}, u_2).(\mathsf{Dial}, u_1, u_4) \triangleleft \mathit{POTS} \triangleright (\mathsf{Ring}, u_1, u_4)
\end{aligned}$$

$$\Leftarrow \quad \langle \text{ POTS-R-15 } \rangle$$

$$\Leftarrow \quad (\mathsf{Init}, u_1, u_4).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_4) \triangleleft POTS \triangleright (\mathsf{Ring}, u_1, u_4) \quad \langle \text{ POTS-A-3 } \rangle$$

$$\mathbf{true}$$

□

Because of the initializing phase it is impossible to reduce a complex input sequence to an axiom by using only reduction rule.

5.3 Three-Way Calling

The feature three-way calling (TWC) allows a subscriber to place a call on Hold, Dial a second user and establish a second two way connection. Then the subscriber is allowed to establish a Three-way call between all active parties. This feature is enaole when the subscriber n_1 presses FlashHook which the PSTN replies to by putting user n_2 on Hold and inform 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.

The input space used by the Three-Way Calling feature is

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

where $I_{TWC} \in \mathsf{I}$.

5.3.1 Three-Way Calling Axioms

As mentioned previously, in this thesis, POTS is defined as a base feature and will be a subset of all other features. Hence TWC is a superset of POTS, meaning that all functionality of POTS is included in the specification of all other features. In reality POTS is the system and its functionality would never be added to the features since features are added to the system to add functionality. Therefore axioms TWC-A-1 through TWC-A-13 would be the same as

No	Premise	Input	Output
A-14		$(\mathsf{Init}, n_1, n_2).(\mathsf{TWCAct}, n1)$	ϕ
A-15	$n_1 \neq n_2$	$\mathsf{c3}.(\mathsf{FlashHook}, n_1)$	$\{(\mathsf{DialTone}, n_1), (\mathsf{Hold}, n_1, n_2)\}$
A-16	$distinct(n_1, n_2, n_3)$	$(\mathsf{Init}, n_3).(\mathsf{Dial}, n_1, n_3)$	$\{(\mathsf{Ring}, n_1, n_3)\}$
A-17		$(\mathsf{OffHook}, n_3)$	$\{(\mathsf{Conn}, n_1, n_3)\}$
A-18		$(\mathsf{FlashHook}, n_1)$	$\{(\mathsf{Conn}, n_1, n_2), (\mathsf{Conn}, n_2, n_3)\}$
A-19	$z \in (2,3) \wedge w \in (2,3) \wedge z \neq w$	(OnHook, n_z)	$\{(\mathsf{Disc}, n_1, n_z), \mathsf{Disc}, n_w, n_z)$
A-20		(OnHook, n_w)	$\{(\mathsf{Disc}, n_1, n_w)\}$
A-21		(OnHook, n_1)	ϕ

Table 5.6: Cumulative Table for a Successful TWC Three Way Connection

POTS (except that now I_R is equal to I_{TWC}) and can be found in Table 5.1.

Where

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

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

Note any occurrence of an abbreviation may be replaced by its defining expression.

Axiom TWC-A-14 states that if a user n_1 is engaged in a two way connection with user n_2 and he presses the FlashHook, the TWC feature will place user n_2 on Hold and inform user n_1 that he may proceed and dial the phone id of user n_3 with a dial tone. Axiom TWC-A-17 states that after user n_1 has dialed the third party and connected a second two-way connection, that he may now create a three-way calling by pressing FlashHook . Even though axiom TWC-A-19 looks complex, it simply states that the if either user n_2 or n_3 hangs up then the connections between that user and the other two users is broken.

Let

$$\mathsf{c4}(n_1, ..., n_t) \triangleq c2(n_1, ..., n_t).\mathsf{c3}.(\mathsf{FlashHook}, n_1).(\mathsf{Dial}, n_1, n_3).(\mathsf{OffHook}, n_3).(\mathsf{FlashHook}, n_1)$$

Table 5.7 describes the axioms which could not be represented in the cumulative table. Axiom TWC-A–23 complements rule TWC-A–19 by stating that if user n_1 hang-up that all connections between all three users will be terminated. The rest of the axioms are self explanatory.

5.3.2 Three-Way Calling Rules

In this section, we present the rules for the Three-Way Calling feature.

No	Premise	Input	Output
A-22	$distinct(n_1, n_2, n_3)$	$c4(n_1, n_2, n_3).(\mathsf{FlashHook}, n_1)$	$\{(\mathsf{Disc}, n_1, n_3) \wedge (\mathsf{Disc}, n_2, n_3)$
A-23	$distinct(n_1, n_2, n_3)$	$c4(n_1, n_2, n_3).(\mathsf{OnHook}, n_1)$	$\{(\mathsf{Disc}, n_1, n_2) \wedge (\mathsf{Disc}, n_1, n_3) \wedge (\mathsf{Disc}, n_2, n_3)\}$
A-24	$distinct(n_1, n_2, n_3)$	$c2(n_1, n_2, n_3).(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_1, n_3).(\mathsf{Dial}, n_2, n_3)$	$\{(\mathsf{BusyTone}, n_2)\}$
A-25	$distint(n_1, n_2, n_3)$	$c3(n_1, n_2, n_3).(\mathsf{FlashHook}, n_1).(\mathsf{OffHook}, n_3).(\mathsf{Dial}, n_1, n_3)$	$\{(\mathsf{BusyTone}, n_1)\}$
A-26		$x.TWCDeact$	φ

Table 5.7: Unsuccessful TWC Connection Attempts

5.3.2.1 Three-Way Calling Reduction Rules

TWC-R–1 to TWC-R–16 are exactly the same as POTS (tables 5.3 and 5.4). Table 5.8 and 5.9 are rules dealing with the TWCAct, TWCDeact, FlashHook inputs. These rules, should be interpreted as in section 5.2.2.1

No	Condition	Prefix	Discard	Postfix
R-17	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	$c2.(\mathsf{OnHook}, n_2)$	x'
R-18	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	$(\mathsf{Dial}, n_1, n_2).\mathsf{OffHook}, n_2.(\mathsf{FlashHook}, n_1).(\mathsf{OnHook}, n_2)$	x'
R-19	$x' \neq \varepsilon$	$x.(\mathsf{OffHook}, n_1)$	$(\mathsf{FlashHook}, n_1)$	x'
R-20	$x' \neq \varepsilon$	x	$(\mathsf{TWCAct}, n_1).(\mathsf{TWCDeact}, n_1)$	x'
R-21	$x' \neq \varepsilon$	x	$(\mathsf{FlashHook}, n_1).(\mathsf{FlashHook}, n_1)$	x'

Table 5.8: TWC Reduction Axioms - Complete Connection Cycle

5.3.2.2 Three-Way Calling Swap Rules

The concatenation of the tables 5.5 and 5.10 represents the swap rules for the Three-Way calling. However note that some of the swap rules when dealing with the input FlashHook must ensure that the user n_1 is not connected with user n_2.

Example

Let

$$TWC1 \triangleq (\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{TWCAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2).(\mathsf{FlashHook}, u_1).(\mathsf{Dial}, u_1, u_3).(\mathsf{OffHook}, u_3).(\mathsf{FlashHook}, u_1).(\mathsf{OnHook}, u_2).(\mathsf{Dial}, u_1, u_4)$$

$$TWC2 \triangleq (\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{TWCAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2).(\mathsf{FlashHook}, u_1).(\mathsf{Dial}, u_1, u_3).(\mathsf{OffHook}, u_3).(\mathsf{OnHook}, u_2).(\mathsf{FlashHook}, u_1).(\mathsf{Dial}, u_1, u_4)$$

These two sequences are exactly the same except that the inputs (OnHook, u_2) and $(\mathsf{FlashHook}, u_1)$ have been swapped. From rule TWC-S–27, without the prefix $x.(\mathsf{Init}, u_2)$, these

No	Condition	Prefix	Discard	Postfix
R-22	$n_1 \neq n_2$	x	(TWCAct, n_1)	(Init, n_2)
R-23		x	(TWCAct, n_1)	(OffHook, n_2)
R-24		x	(TWCAct, n_1)	(Dial, n_2, n_3)
R-25		x	(TWCAct, n_1)	(OnHook, n_2)
R-26	$n_1 \neq n_2$	x	(TWCAct, n_1)	(FlashHook, n_2)
R-27	$n_1 \neq n_2$	x	(TWCAct, n_1)	(TWCAct, n_2)
R-28	$n_1 \neq n_2$	x	(TWCAct, n_1)	(TWCDeact, n_2)
R-29	$n_1 \neq n_2$	x	(Init, n_1)	(FlashHook, n_2)
R-30	$n_1 \neq n_2$	x	(Init, n_1)	(TWCAct, n_2)
R-31	$n_1 \neq n_2$	x	(Init, n_1)	(TWCDeact, n_2)
R-32	$n_1 \neq n_2$	x	(OffHook, n_1)	(FlashHook, n_2)
R-33		x	(OffHook, n_1)	(TWCAct, n_2)
R-34		x	(OffHook, n_1)	(TWCDeact, n_2)
R-35	$n_1 \neq n_2$	x	(Dial, n_1, n_2)	(FlashHook, n_3,)
R-36	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(TWCAct, n_3)
R-37	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(TWCDeact, n_3)
R-38	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Init, n_2)
R-39	$n_1 \neq n_2$	x	(FlashHook, n_1)	(OffHook, n_2)
R-40	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Dial, n_2, n_3)
R-41	$n_1 \neq n_2$	x	(FlashHook, n_1)	(FlashHook, n_2)
R-42	$n_1 \neq n_2$	x	(FlashHook, n_1)	(TWCAct, n_2)
R-43	$n_1 \neq n_2$	x	(FlashHook, n_1)	(TWCDeact, n_2)

Table 5.9: TWC Reduction Rules - Connection in Progress

two sequences should be equivalent. However $TWC1$ generates no visible output while $TWC2$ generates a $\{(\text{Ring}, u_1, u_2)\}$. Adding the prefix $x.(\text{Init}, u_2)$ ensures these two inputs are not swapped when this case arises.

PROOF.

$$(\text{Init}, u_1, u_2, u_3, u_4).(\text{TWCAct}, n_1).(\text{OffHook}, u_1).(\text{Dial}, u_1, u_2).(\text{OffHook}, u_2).$$
$$(\text{FlashHook}, u_1).(\text{Dial}, u_1, u_3).(\text{OffHook}, u_3).(\text{OnHook}, u_2).(\text{FlashHook}, u_1).$$
$$(\text{Dial}, u_1, u_4) \triangleleft TWC \triangleright \{(\text{Ring}, u_1, u_4)\}$$
$$\Leftarrow \quad \langle \text{ TWC-S–3 } \rangle$$

$$(\text{Init}, u_1, u_2, u_3, u_4).(\text{TWCAct}, n_1).(\text{OffHook}, u_1).(\text{Dial}, u_1, u_2).(\text{OffHook}, u_2).$$
$$(\text{FlashHook}, u_1).(\text{Dial}, u_1, u_3).(\text{OnHook}, u_2).(\text{OffHook}, u_2).(\text{FlashHook}, u_1).$$
$$(\text{Dial}, u_1, u_4) \triangleleft TWC \triangleright \{(\text{Ring}, u_1, u_4)\}$$
$$\Leftarrow \quad \langle \text{ TWC-S–6 } \rangle$$

$$(\text{Init}, u_1, u_2, u_3, u_4).(\text{TWCAct}, n_1).(\text{OffHook}, u_1).(\text{Dial}, u_1, u_2).(\text{OffHook}, u_2).$$
$$(\text{FlashHook}, u_1).(\text{OnHook}, u_2).(\text{Dial}, u_1, u_3).(\text{OffHook}, u_3).(\text{FlashHook}, u_1).$$
$$(\text{Dial}, u_1, u_4) \triangleleft TWC \triangleright \{(\text{Ring}, u_1, u_4)\}$$
$$\Leftarrow \quad \langle \text{ TWC-R–18 } \rangle$$

$$(\text{Init}, u_1, u_2, u_3, u_4).(\text{TWCAct}, n_1).(\text{OffHook}, u_1).(\text{Dial}, u_1, u_3).(\text{OffHook}, u_3).$$
$$(\text{FlashHook}, u_1).(\text{Dial}, u_1, u_4) \triangleleft TWC \triangleright \{(\text{Ring}, u_1, u_4)\}$$
$$\Leftarrow \quad \langle \text{ TWC-A–16 } \rangle$$

$$\textbf{true}$$

□

No	Condition	Prefix	Input 1	Input 2	Postfix
S-11	$n_1 \neq n_2$	x	(TWCAct, n_1)	(Init, n_2)	x'
S-12		x	(TWCAct, n_1)	(OffHook, n_2)	x'
S-13		x	(TWCAct, n_1)	(Dial, n_2, n_3)	x'
S-14		x	(TWCAct, n_1)	(OnHook, n_2)	x'
S-15	$n_1 \neq n_2$	x	(TWCAct, n_1)	(FlashHook, n_2)	x'
S-16	$n_1 \neq n_2$	x	(TWCAct, n_1)	(TWCAct, n_2)	x'
S-17	$n_1 \neq n_2$	x	(TWCAct, n_1)	(TWCDeact, n_2)	x'
S-18	$n_1 \neq n_2$	x	(TWCDeact, n_1)	(Init, n_2)	x'
S-19	$n_1 \neq n_2$	x	(TWCDeact, n_1)	(OffHook, n_2)	x'
S-20	$n_1 \neq n_2$	x	(TWCDeact, n_1)	(Dial, n_2, n_3)	x'
S-21	$n_1 \neq n_2$	x	(TWCDeact, n_1)	(OnHook, n_2)	x'
S-22	$n_1 \neq n_2$	x	(TWCDeact, n_1)	(FlashHook, n_2)	x'
S-23	$n_1 \neq n_2$	x	(TWCDeact, n_1)	(TWCDeact, n_2)	x'
S-24	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Init, n_2)	x'
S-25	$n_1 \neq n_2$	x.(Init, n_2)	(FlashHook, n_1)	(OffHook, n_2)	x'
S-26	$n_1 \neq n_2$	x.(Init, n_2).(OffHook, n_2	(FlashHook, n_1)	(Dial, n_2, n_3)	x'
S-27	$n_1 \neq n_2$	x.(Init, n_2)	(FlashHook, n_1)	(OnHook, n_2)	x'
S-28	$n_1 \neq n_2$	x	(FlashHook, n_1)	(FlashHook, n_2)	x'

Table 5.10: TWC Swap Rules

It is left to the reader to prove that sequence $TWC1$ does in fact yield no visible output.

5.4 Call Waiting

The call-waiting features allows a subscriber who is already busy in an active conversation to receive a call from a third party. When a third party dials the subscriber hears a CWtone (WaitTone) and can join the third party into the call by pressing FlashHook. Note that the active call is put on Hold therefore the call waiting feature does not create a three-way call. The subscriber may toggle between the two two-way call by pressing FlashHook.

The CW input space is defined as:

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

where $I_{CW} \in \mathsf{I}$.

5.4.1 Call Waiting Axioms

The CW axioms are also a superset of the POTS axioms, therefore axioms TWC-A-1 through TWC-A-13 are defined in tables 5.1 and 5.2 where $I_R = I_{CW}$.

$$c5 \triangleq (\mathsf{Init}, n_2, n_3, n_4).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_2).(\mathsf{OffHook}, n_3).(\mathsf{Dial}, n_3, n_1)$$

No	Premise	Input	Output
A-14		$(\mathsf{Init}n_1).(\mathsf{CWAct}, n_1)$	ϕ
A-15	$distinct(n_1, n_2, n_3)$	c5	$\{(\mathsf{RingTone}, n_3), (\mathsf{WaitTone}, n_1)\}$
A-16		$(\mathsf{FlashHook}, n_1)$	$\{(\mathsf{Hold}, n_1, n_2), (\mathsf{Conn}, n_1, n_3)\}$
A-17		$(\mathsf{FlashHook}, n_1)$	$\{(\mathsf{Conn}, n_1, n_2), (\mathsf{Hold}, n_1, n_3)\}$
A-18		(OnHook, n_3)	ϕ

Table 5.11: Cumulative Table for a Successful CW Connection

No	Premise	Input	Output
A-19	$distinct(n_1, n_2, n_3, n_4)$	$(\mathsf{Init}, n_1).(\mathsf{CWAct}, n_1).c5(\mathsf{FlashHook}, n_1).(\mathsf{OffHook}, n_4).$ $(\mathsf{Dial}, n_4, n_1)$	$\{(\mathsf{BusyTone}, n_4)\}$
A-20		$x.(\mathsf{CWDeact}, n_1)$	ϕ

Table 5.12: Unsuccessful Connection Attempts

Axiom CW-A–15 states that if user n_3 tries to establish a connection with user n_1 which is presently engaged with user n_2, the telephone system will inform user n_1 that another user is attempting to call him by means of a WaitTone. Meanwhile user n_3 will be informed by a RingTone that user n_1 may be contacted. After n_1 has receive the notification axiom CW-A–16 states that he/she may accept the incoming call by pressing FlashHook. It may be observe that no axiom states that user n_1 may return to the previous conversation by merely pressing the FlashHook a second time. It will be shown in the section 5.4.2.1 that such an axiom is not needed.

5.4.2 Call Waiting Rules

In this section, we present both the reduction and swap rules for Call Waiting.

5.4.2.1 Call Waiting Reduction Rules

Rules CW-R–1 to can CW-R–16 can be constructed by means of tables 5.3 and 5.4 where $I_R = I_{CW}$.

Let

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

No	Condition	Prefix	Discard	Postfix
R-17	$n_1 \neq n_2$	x	(CWAct, n_1)	(Init, n_2)
R-18		x	(CWAct, n_1)	(OffHook, n_2)
R-19		x	(CWAct, n_1)	(Dial, n_2, n_3)
R-20		x	(CWAct, n_1)	(OnHook, n_2)
R-21	$n_1 \neq n_2$	x	(CWAct, n_1)	(FlashHook, n_2)
R-22	$n_1 \neq n_2$	x	(CWAct, n_1)	(CWAct, n_2)
R-23	$n_1 \neq n_2$	x	(CWAct, n_1)	(CWDeact, n_2)
R-24	$n_1 \neq n_2$	x	(Init, n_1)	(FlashHook, n_2)
R-25	$n_1 \neq n_2$	x	(Init, n_1)	(CWAct, n_2)
R-26	$n_1 \neq n_2$	x	(Init, n_1)	(TWCDeact, n_2)
R-27	$n_1 \neq n_2$	x	(OffHook, n_1)	(FlashHook, n_2)
R-28	$n_1 \neq n_2$	x	(OffHook, n_1)	(CWAct, n_2)
R-29	$n_1 \neq n_2$	x	(OffHook, n_1)	(TWCDeact, n_2)
R-30	$distinct(n_1, n_2, n_3)$	x	(Dial, n_1, n_2)	(FlashHook, n_3,)
R-31	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CWAct, n_3)
R-32	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CWDeact, n_3)
R-33	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Init, n_2)
R-34	$n_1 \neq n_2$	x	(FlashHook, n_1)	(OnHook, n_2)
R-35	$n_1 \neq n_2$	x	(FlashHook, n_1)	(OffHook, n_2)
R-36	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Dial, n_2, n_3)
R-37	$n_1 \neq n_2$	x	(FlashHook, n_1)	(FlashHook, n_2)
R-38	$n_1 \neq n_2$	x	(FlashHook, n_1)	(CWDeact, n_2)

Table 5.13: CW Reduction Rules - Connection in Progress

No	Condition	Prefix	Discard	Postfix
R-39	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	c6.(OnHook, n_2)	x'
R-40	$x' \neq \varepsilon \wedge n_1 \neq n_2$	x	c6.(FlashHook, n_1).(OnHook, n_2)	x'
R-41	$x' \neq \varepsilon$	x.(OffHook, n_1)	(FlashHook, n_1)	x'
R-42		x	(FlashHook, n_1).(FlashHook, n_1)	x'
R-43	$x' \neq \varepsilon$	x	(CWAct, n_1).(CWDeact, n_1)	x'

Table 5.14: CW Reduction Axioms - Complete Cycle

5.4.2.2 Call Waiting Swap Rules

Rules CW-S–1 to CW-S–10 can be constructed from tables 5.5. A prefix is needed for some of the swap rules for a reason similar to the Three-Way Calling feature (Section 5.3.2.2).

Here is a proof to show why an axiom for toggling between two users is not needed (as stated in section 5.4.1)

PROOF.

$$(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2).(\mathsf{OffHook}, u_3).$$
$$(\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1).(\mathsf{FlashHook}, u_1) \triangleleft CW \triangleright \{(\mathsf{Conn}, n_1, n_2), (\mathsf{Hold}, n_1, n_3)\}$$
$$\Leftarrow \quad \langle \text{ CW-S–1 } \rangle$$

No	Condition	Prefix	Input 1	Input 2	Postfix
S-11	$n_1 \neq n_2$	x	(CWAct, n_1)	(Init, n_2)	x'
S-12		x	(CWAct, n_1)	(OffHook, n_2)	x'
S-13		x	(CWAct, n_1)	(Dial, n_2, n_3)	x'
S-14		x	(CWAct, n_1)	(OnHook, n_2)	x'
S-15	$n_1 \neq n_2$	x	(CWAct, n_1)	(FlashHook, n_2)	x'
S-16	$n_1 \neq n_2$	x	(CWAct, n_1)	(CWAct, n_2)	x'
S-17	$n_1 \neq n_2$	x	(CWAct, n_1)	(CWDeact, n_2)	x'
S-18	$n_1 \neq n_2$	x	(CWDeact, n_1)	(Init, n_2)	x'
S-19	$n_1 \neq n_2$	x	(CWDeact, n_1)	(OffHook, n_2)	x'
S-20	$n_1 \neq n_2$	x	(CWDeact, n_1)	(Dial, n_2, n_3)	x'
S-21	$n_1 \neq n_2$	x	(CWDeact, n_1)	(OnHook, n_2)	x'
S-22	$n_1 \neq n_2$	x	(CWDeact, n_1)	(FlashHook, n_2)	x'
S-23	$n_1 \neq n_2$	x	(CWDeact, n_1)	(CWDeact, n_2)	x'
S-24	$n_1 \neq n_2$	x	(FlashHook, n_1)	(Init, n_2)	x'
S-25	$n_1 \neq n_2$	x.(Init, n_2).(OffHook, n_2	(FlashHook, n_1)	(Dial, n_2, n_3)	x'
S-26	$n_1 \neq n_2$	x	(FlashHook, n_1)	(FlashHook, n_2)	x'
S-27	$n_1 \neq n_2$	x.(OnHook, n_2)	(FlashHook, n_1)	(OnHook, n_2)	x'
S-28	$n_1 \neq n_2$	x.(OnHook, n_2)	(FlashHook, n_1)	(OffHook, n_2)	x'

Table 5.15: CW Swap Rules

$$
\begin{aligned}
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_3).(\mathsf{OffHook}, u_2).\\
&(\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1).(\mathsf{FlashHook}, u_1) \triangleleft CW \triangleright \{(\mathsf{Conn}, n_1, n_2), (\mathsf{Hold}, n_1, n_3)\}\\
\Leftarrow \quad & \langle \text{ CW-S-2 } \rangle
\end{aligned}
$$

$$
\begin{aligned}
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_3).(\mathsf{Dial}, u_3, u_1).\\
&(\mathsf{OffHook}, u_2).(\mathsf{FlashHook}, u_1).(\mathsf{FlashHook}, u_1) \triangleleft CW \triangleright \{(\mathsf{Conn}, n_1, n_2), (\mathsf{Hold}, n_1, n_3)\}\\
\Leftarrow \quad & \langle \text{ CW-R-42 } \rangle
\end{aligned}
$$

$$
\begin{aligned}
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_3).(\mathsf{Dial}, u_3, u_1).\\
&(\mathsf{OffHook}, u_2) \triangleleft CW \triangleright \{(\mathsf{Conn}, n_1, n_2), (\mathsf{Hold}, n_1, n_3)\}\\
\Leftarrow \quad & \langle \text{ CW-R-11 } \rangle
\end{aligned}
$$

$$
\begin{aligned}
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_3).(\mathsf{OffHook}, u_2)\\
&\quad \triangleleft CW \triangleright \{(\mathsf{Conn}, n_1, n_2), (\mathsf{Hold}, n_1, n_3)\}\\
\Leftarrow \quad & \langle \text{ CW-R-7 } \rangle
\end{aligned}
$$

$$
\begin{aligned}
&(\mathsf{Init}, u_1, u_2, u_3, u_4).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2)\\
&\quad \triangleleft CW \triangleright \{(\mathsf{Conn}, n_1, n_2), (\mathsf{Hold}, n_1, n_3)\}\\
\Leftarrow \quad & \langle \text{ CW-A-15 } \rangle
\end{aligned}
$$

$$\mathbf{true} \qquad \square$$

5.5 Originating Call Screening

The Originating Call Screening (OCS) feature allows a subscriber to prevent a connection between some undesirable phone id's. The disallowed number n_2 is added to the database

for subscriber n_1 by initiating a $(\mathsf{OCSAdd}, n_1, n_2)$. Similarly, the number is removed from the database by the subscriber by initiating a $(\mathsf{OCSDel}, n_1, n_2)$. The space associated with this feature is as followed:

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

where $I_{OCS} \in \mathsf{I}$.

5.5.1 Originating Call Screening Axioms

Since OCS is a superset of POTS and that all axioms relevant to POTS is also relevant to OCS, rules OCS-A–1 to OCS-A–13 can be constructed from tables 5.1 and 5.2. Note that I_R must be substitute with I_{OCS}.

No	Premise	Input	Output
A-14		$(\mathsf{Init}, n_1, n_2).(\mathsf{OCSAct}, n_1)$	ϕ
A-15	$n_1 \neq n_2$	$(\mathsf{OCSAdd}, n_1, n_2)$	ϕ
A-16		$(\mathsf{OffHook}, n_1)$	$\{(\mathsf{DialTone}, n_1)\}$
A-17		$(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{ErrorTone}, n_1)\}$
A-18		(OnHook, n_1)	ϕ

Table 5.16: Cumulative Table for a Successful OCS Connection

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

Axiom OCS-A-17 state that a connection will not be established if user n_1 tries to dial a phone id n_2 which as been added to the OCS database.

5.5.2 Originating Call Screening Rules

We present the OCS reduction and swap rules in sections 5.5.2.1 and 5.5.2.2 respectively.

No	Premise	Input	Output
A-19	$distinct(n_1, n_2)$	c7.(OffHook, n_1).(Dial, n_1, n_2)	{(ErrorTone, n_1)}
A-20		(OCSAct, n_1).(OCSAdd, n_1, n_1)	{(ErrorTone, n_1)}
A-21		$x.OCS\,Deact$	φ
A-22		$x.OCS\,Del$	φ
A-23	$x' \neq (\mathsf{OCSDel}, n_1, n_2) \wedge x' \neq \varepsilon \wedge (\mathsf{Conn}, n_1, n_2) \notin y$	$x.(\mathsf{OCSAdd}, n_1, n_2).x'$	y

Table 5.17: Table for an Unsuccessful OCS Connection

5.5.2.1 Originating Call Screening Reduction Rules

All rules for the OCS feature may be constructed from tables 5.3, 5.4, 5.18 and 5.19.

No	Condition	Prefix	Discard	Postfix
R-17	$x' \neq \varepsilon \wedge i \in \natural$	x	(OCSAct, n_1).(OCSAdd, n_1, $-$)*.(OCSDeact, n_1)	x'
R-18	$x' \neq \varepsilon$	x	(OCSAdd, n_1).(OCSDel, n_1)	x'

Table 5.18: OCS Reduction Axioms - Complete Cycle

OCS complete cycle reduction rule differs from the complete cycle reduction rule described in the previous features. Rule OCS-R-18 states that any valid number added to the OCS database is automatically deleted when the OCS feature is deactivated.

5.5.2.2 Originating Call Screening Swap Rules

Rules CW-S–1 to CW-S–10 can be constructed from tables 5.5. All other rules may be constructed by means of table 5.20 Let assume that the validation team generated the sequence $OCS1$ define as:

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OCSAdd}, u_1, u_2).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OCSAdd}, u_1, u_3).$$
$$(\mathsf{OCSDeact}, u_1).(\mathsf{OnHook}, u_1).(\mathsf{OCSAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\}$$

The purpose of this validation sequence is to verify whether all entries in the OCS database for a user are deleted when the OCS feature is disactivated. The $OCS1$ sequence activates the OCS feature for user u_1, added user u_2 in the OCS database and tries to call him. Afterwards another user is added to the database. After the OCS feature has been deactivated all entries should be deleted. It would not be a desirable feature if the new phone id was given to another subscriber and he was stuck with the list of the previous user. Therefore by dis-activating the feature and reactivating it, the OCS database for that phone id should be re-initialized.

No	Condition	Prefix	Discard	Postfix
R-19	$n_1 \neq n_2$	x	(OCSAct, n_1)	(Init, n_2)
R-20		x	(OCSAct, n_1)	(OffHook, n_2)
R-21		x	(OCSAct, n_1)	(Dial, n_2, n_3)
R-22		x	(OCSAct, n_1)	(OnHook, n_2)
R-23	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSAdd, n_2, n_3)
R-24	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSAct, n_2)
R-25	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSDeact, n_2)
R-26	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSAdd, n_2, n_3)
R-27	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSDel, n_2, n_3)
R-28	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSAct, n_2)
R-29	$n_1 \neq n_2$	x	(OffHook, n_1)	(OCSDeact, n_2)
R-30	$n_1 \neq n_3$)	x	(Dial, n_1, n_2)	(OCSAdd, n_3, n_4)
R-31	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OCSDel, n_3, n_4)
R-32	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OCSAct, n_3)
R-33	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(OCSDeact, n_3)
R-34	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(Init, n_3)
R-35		x	(OCSAdd, n_1, n_2)	(OffHook, n_3)
R-36	$n_1 \neq n_4$	x	(OCSAdd, n_1, n_2)	(Dial, n_3, n_4)
R-37		x	(OCSAdd, n_1, n_2)	(OnHook, n_3)
R-38	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSAdd, n_3, n_4)
R-39	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSDel, n_3, n_4)
R-40	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSAct, n_3)
R-41	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSDeact, n_3)

Table 5.19: OCS Reduction Rules - Cycle in Progress

The following proof, shows that the OCS specification meets this requirement.

PROOF.

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OCSAdd}, u_1, u_2).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).$$
$$(\mathsf{OCSAdd}, u_1, u_3).(\mathsf{OCSDeact}, u_1).(\mathsf{OCSAct}, u_1).(\mathsf{OnHook}, u_1).(\mathsf{OffHook}, u_1).$$
$$(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\}$$
$$\Leftarrow \quad \langle \text{ OCS-S-21 } \rangle$$

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OCSAdd}, u_1, u_2).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).$$
$$(\mathsf{OCSAdd}, u_1, u_3).(\mathsf{OnHook}, u_1).(\mathsf{OCSDeact}, u_1).(\mathsf{OCSAct}, u_1).(\mathsf{OffHook}, u_1).$$
$$(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\}$$
$$\Leftarrow \quad \langle \text{ OCS-S-28 } \rangle$$

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OCSAdd}, u_1, u_2).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).$$
$$(\mathsf{OnHook}, u_1).(\mathsf{OCSAdd}, u_1, u_3).(\mathsf{OCSDeact}, u_1).(\mathsf{OCSAct}, u_1).(\mathsf{OffHook}, u_1).$$
$$(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\}$$
$$\Leftarrow \quad \langle \text{ OCS-R-3 } \rangle$$

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OCSAdd}, u_1, u_2).(\mathsf{OCSAdd}, u_1, u_2, u_3).(\mathsf{OCSDeact}, u_1).$$
$$(\mathsf{OCSAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\}$$
$$\Leftarrow \quad \langle \text{ OCS-R-17 } \rangle$$

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\}$$
$$\Leftarrow \quad \langle \text{ OCS-S-11, OCS-S-10 \& OCS-S-9 } \rangle$$

No	Condition	Prefix	Input 1	Input 2	Postfix
S-11	$n_1 \neq n_2$	x	(OCSAct, n_1)	(Init, n_2)	x'
S-12		x	(OCSAct, n_1)	(OffHook, n_2)	x'
S-13		x	(OCSAct, n_1)	(Dial, n_2, n_3)	x'
S-14		x	(OCSAct, n_1)	(OnHook, n_2)	x'
S-15	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSAdd, n_2, n_3)	x'
S-16	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSAct, n_2)	x'
S-17	$n_1 \neq n_2$	x	(OCSAct, n_1)	(OCSDeact, n_2)	x'
S-18	$n_1 \neq n_2$	x	(OCSDeact, n_1)	(Init, n_2)	x'
S-19	$n_1 \neq n_2$	x	(OCSDeact, n_1)	(OffHook, n_2)	x'
S-20	$n_1 \neq n_2$	x	(OCSDeact, n_1)	(Dial, n_2, n_3)	x'
S-21	$n_1 \neq n_2$	x	(OCSDeact, n_1)	(OnHook, n_2)	x'
S-22	$n_1 \neq n_2$	x	(OCSDeact, n_1)	(OCSAdd, n_2, n_3)	x'
S-23	$n_1 \neq n_2$	x	(OCSDeact, n_1)	(OCSDel, n_2, n_3)	x'
S-24	$n_1 \neq n_2$	x	(OCSDeact, n_1)	(OCSDeact, n_2)	x'
S-25		x	(OCSAdd, n_1, n_2)	(Init, n_3)	x'
S-26		x	(OCSAdd, n_1, n_2)	(OffHook, n_3)	x'
S-27	$n_1 \neq n_4$	x	(OCSAdd, n_1, n_2)	(Dial, n_3, n_4)	x'
S-28		x	(OCSAdd, n_1, n_2)	(OnHook, n_3)	x'
S-29		x	(OCSAdd, n_1, n_2)	(OCSAdd, n_3, n_4)	x'
S-30	$n_1 \neq n_3$	x	(OCSAdd, n_1, n_2)	(OCSDel, n_3, n_4)	x'
S-31	$n_1 \neq n_3$	x	(OCSDel, n_1, n_2)	(Init, n_3)	x'
S-32	$n_1 \neq n_3$	x	(OCSDel, n_1, n_2)	(OffHook, n_3)	x'
R-33	$n_1 \neq n_3$	x	(OCSDel, n_1, n_2)	(Dial, n_3, n_4)	x'
R-34	$n_1 \neq n_3$	x	(OCSDel, n_1, n_2)	(OnHook, n_3)	x'
R-35		x	(OCSDel, n_1, n_2)	(OCSDel, n_3, n_4)	x'

Table 5.20: OCS Swap Rules

$$
\begin{aligned}
& (\mathsf{Init}, u_1, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Init}, u_2).(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\} \\
\Leftarrow \quad & \langle \text{ OCS-R-15 \& OCS-S-9 } \rangle \\
& (\mathsf{Init}, u_1, u_3).(\mathsf{OCSAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\} \\
\Leftarrow \quad & \langle \text{ OCS-S-12 } \rangle \\
& (\mathsf{Init}, u_1, u_3).(\mathsf{OffHook}, u_1).(\mathsf{OCSAct}, u_1).(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\} \\
\Leftarrow \quad & \langle \text{ OCS-R-21 } \rangle \\
& (\mathsf{Init}, u_1, u_3).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_3) \triangleleft OCS \triangleright \{\mathsf{Ring}, u_1, u_3)\} \\
\Leftarrow \quad & \langle \text{ OCS-A-3 } \rangle \\
& \mathbf{true}
\end{aligned}
$$

□

5.6 Abbreviated Dialing

The Abbreviated Dialing (ABD) features (sometimes referred to as Speed Call) allows a subscriber to create an alias for a phone id which is used repeatedly. The alias n_3 for phone id n_2 is added to the database for subscriber n_1 by initiating (ABDAdd, $n1, n2, n3$). Similarly, the alias is removed from the subscribers database by initiating a (ABDDdel, n_1, n_3).

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

where $I_{ABD} \in \mathsf{I}$.

5.6.1 Abbreviated Dialing Axioms

The axioms for the Abbreviated Dialing feature can be produced by applying the proper formula discussed previously to tables 5.1, 5.2, 5.21 and 5.22.

No	Premise	Input	Output
A-14		$(\mathsf{Init}, n_1, n_2).(\mathsf{ABDAct}, n_1)$	ϕ
A-15		$(\mathsf{OffHook}, n_1)$	$\{(\mathsf{DialTone}, n_1)\}$
A-16	$distinct(n_1, n_2, n_3)$	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	ϕ
A-17		$(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{Ring}, n_1, n_3)\}$
A-18		$(\mathsf{OffHook}, n_3)$	$\{(\mathsf{Conn}, n_1, n_3)\}$
A-19	$n \in \{n_1, n_3\}$	$(\mathsf{OffHook}, n)$	$\{(\mathsf{Disc}, n_1, n_3)\}$
A-20	$n' \in \{n_1, n_3\} \wedge n' \neq n$	(OnHook, n')	ϕ

Table 5.21: Cumulative Table for a Successful ABD Connection

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

No	Premise	Input	Output
A-21	$distinct(n_1, n_2, n_3)$	$\mathsf{c8}.(\mathsf{OffHook}, n_3).(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{BusyTone}, n_1)\}$
A-22	$distinct(n_1, n_2, n_3, n_4)$	$\mathsf{c8}.(\mathsf{OffHook}, n_4).(\mathsf{Dial}, n_4, n_3).(\mathsf{Dial}, n_1, n_2)$	$\{(\mathsf{BusyTone}, n_1)\}$
A-23		$x.(\mathsf{OCSDeact}, n_1)$	ϕ
A-24		$x.(\mathsf{ABDDel}, n_1, n_2)$	ϕ

Table 5.22: Table for an Unsuccessful ABD Connection

5.6.2 Abbreviated Dialing Rules

We present the reduction and swap rules for this feature in sections 5.6.2.1 and 5.6.2.2 respectively.

5.6.2.1 Abbreviated Dialing Reduction Rules

The reduction rules of ABD can be contructed from tables 5.3, 5.4, 5.23 and 5.24. The rule

No	Condition	Prefix	Discard	Postfix
R-17	$x' \neq \varepsilon \wedge i \in Natural$	x	$(\mathsf{ABDAct}, n_1).(\mathsf{ABDAdd}, n_1, -)^i.(\mathsf{ABDDeact}, n_1)$	x'
R-18	$x' \neq \varepsilon$	x	$(\mathsf{ABDAdd}. n_1).(\mathsf{ABDDel}, n_1)$	x'

Table 5.23: ABD Reduction Axioms - Complete Connection Cycle

ABD-R–17 is interpreted similarly to the rule OCS-R–17 describe in section 5.5.2.1.

No	Condition	Prefix	Discard	Postfix
R-19	$n_1 \neq n_2$	x	(ABDAct, n_1)	(Init, n_2)
R-20		x	(ABDAct, n_1)	$(\mathsf{OffHook}, n_2)$
R-21		x	(ABDAct, n_1)	$(\mathsf{Dial}, n_2, n_3)$
R-22		x	(ABDAct, n_1)	(OnHook, n_2)
R-23	$n_1 \neq n_2$	x	(ABDAct, n_1)	$(\mathsf{ABDAdd}, n_2, n_3, n_4)$
R-24	$n_1 \neq n_2$	x	(ABDAct, n_1)	(ABDAct, n_2)
R-25	$n_1 \neq n_2$	x	(ABDAct, n_1)	$(\mathsf{ABDDeact}, n_2)$
R-26	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{ABDAdd}, n_2, n_3, n_4)$
R-27	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{OCSDel}, n_2, n_3)$
R-28	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	(ABDAct, n_2)
R-29	$n_1 \neq n_2$	x	$(\mathsf{OffHook}, n_1)$	$(\mathsf{ABDDeact}, n_2)$
R-30	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{ABDAdd}, n_3, n_4, n_5)$
R-31	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{ABDDel}, n_3, n_4)$
R-32	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	(ABDAct, n_3)
R-33	$n_1 \neq n_3$	x	$(\mathsf{Dial}, n_1, n_2)$	$(\mathsf{ABDDeact}, n_3)$
R-34	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	(Init, n_4)
R-35		x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{OffHook}, n_4)$
R-36		x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{Dial}, n_4, n_5)$
R-37		x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	(OnHook, n_4)
R-38	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{OCSAdd}, n_4, n_5)$
R-39	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{OCSDel}, n_4, n_5)$
R-40	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	(ABDAct, n_4)
R-41	$n_1 \neq n_4$	x	$(\mathsf{ABDAdd}, n_1, n_2, n_3)$	$(\mathsf{ABDDeact}, n_4)$

Table 5.24: ABD Reduction Rules - Connection in Progress

5.6.2.2 Abbreviated Dialing Swap Rules

Swap rules for the ABD feature may be constructed from tables 5.3, 5.4, 5.23 and 5.24.

No	Condition	Prefix	Input 1	Input 2	Postfix
S-11	$n_1 \neq n_2$	x	(ABDAct, n_1)	(Init, n_2)	x'
S-12		x	(ABDAct, n_1)	(OffHook, n_2)	x'
S-13		x	(ABDAct, n_1)	(Dial, n_2, n_3)	x'
S-14		x	(ABDAct, n_1)	(OnHook, n_2)	x'
S-15	$n_1 \neq n_2$	x	(ABDAct, n_1)	(ABDAdd, n_2, n_3, n_4)	x'
S-16	$n_1 \neq n_2$	x	(ABDAct, n_1)	(ABDAct, n_2)	x'
S-17	$n_1 \neq n_2$	x	(ABDAct, n_1)	(ABDDeact, n_2)	x'
S-18	$n_1 \neq n_2$	x	(ABDDeact, n_1)	(Init, n_2)	x'
S-19	$n_1 \neq n_2$	x	(ABDDeact, n_1)	(OffHook, n_2)	x'
S-20	$n_1 \neq n_2$	x	(ABDDeact, n_1)	(Dial, n_2, n_3)	x'
S-21	$n_1 \neq n_2$	x	(ABDDeact, n_1)	(OnHook, n_2)	x'
S-22	$n_1 \neq n_2$	x	(ABDDeact, n_1)	(ABDAdd, n_2, n_3, n_4)	x'
S-23	$n_1 \neq n_2$ x		(ABDDeact, n_1)	(ABDDel, n_2, n_3)	x'
S-24	$n_1 \neq n_2$	x	(ABDDeact, n_1)	(ABDDeact, n_2)	x'
S-25	$n_1 \neq n_4$	x	(ABDAdd, n_1, n_2, n_3)	(Init, n_4)	x'
S-26		x	(ABDAdd, n_1, n_2, n_3)	(OffHook, n_4)	x'
S-27	$n_1 \neq n_4$	x	(ABDAdd, n_1, n_2, n_3)	(Dial, n_4, n_5)	x'
S-28	$n_1 \neq n_4$	x	(ABDAdd, n_1, n_2, n_3)	(OnHook, n_4)	x'
S-29	$n_1 \neq n_4$	x	(ABDAdd, n_1, n_2, n_3)	(ABDAdd, n_4, n_5, n_6)	x'
S-30	$n_1 \neq n_4$	x	(ABDAdd, n_1, n_2, n_3)	(OCSDel, n_4, n_5)	x'
S-31	$n_1 \neq n_3$	x	(ABDDel, n_1, n_2)	(Init, n_3)	x'
S-32	$n_1 \neq n_3$	x	(ABDDel, n_1, n_2)	(OffHook, n_3)	x'
S-33	$n_1 \neq n_3$	x	(ABDDel, n_1, n_2)	(Dial, n_3, n_4)	x'
S-34	$n_1 \neq n_3$	x	(ABDDel, n_1, n_2)	(OnHook, n_3)	x'
S-35	$n_1 \neq n_3$	x	(ABDDel, n_1, n_2)	(ABDDel, n_3, n_4)	x'

Table 5.25: ABD Swap Rules

A validation sequence similar to sequence $OCS1$ may be used to validate the ABD specification. However we will show that a user may not use the abbreviated number after it has been deleted. This can be acheived by :

$$(\mathsf{Init}, u_1, u_2).(\mathsf{ABDact}, u_1).(\mathsf{Offhook}, u_1).(\mathsf{ABDAdd}, u_1, u_3, u_2).(\mathsf{OnHook}, u_1).$$
$$(\mathsf{ABDDel}, u_1, u_3).(\mathsf{Offhook}, u_1).(\mathsf{Dial}, u_1, u_3)$$

PROOF.

$$(\mathsf{Init}, u_1, u_2).(\mathsf{ABDAct}, u_1).(\mathsf{Offhook}, u_1).(\mathsf{ABDAdd}, u_1, u_3, u_2).(\mathsf{OnHook}, u_1).$$
$$(\mathsf{ABDDel}, u_1, u_3).(\mathsf{Offhook}, u_1).(\mathsf{Dial}, n_1, n_3) \triangleleft ABD \triangleright \{\mathsf{ErrorTone}, u_1)\}$$
$\Leftarrow$ ⟨ ABD-S–28 ⟩

$$(\mathsf{Init}, u_1, u_2).(\mathsf{ABDAct}, u_1).(\mathsf{Offhook}, u_1).(\mathsf{Onhook}, u_1).(\mathsf{ABDAdd}, u_1, u_3, u_2).$$
$$(\mathsf{ABDDel}, u_1, u_3).(\mathsf{Offhook}, u_1).(\mathsf{Dial}, n_1, n_3) \triangleleft ABD \triangleright \{\mathsf{ErrorTone}, u_1)\}$$
$\Leftarrow$ ⟨ ABD-R–2 & ABD-R–18 ⟩

$$(\mathsf{Init}, u_1, u_2).(\mathsf{ABDAct}, u_1).(\mathsf{Offhook}, u_1).(\mathsf{Dial}, n_1, n_3) \triangleleft ABD \triangleright \{\mathsf{ErrorTone}, u_1)\}$$
$\Leftarrow$ ⟨ ABD-S–12 ⟩

$$(\mathsf{Init}, u_1, u_2).(\mathsf{Offhook}, u_1).(\mathsf{ABDAct}, u_1).(\mathsf{Dial}, n_1, n_3) \triangleleft ABD \triangleright \{\mathsf{ErrorTone}, u_1)\}$$
$\Leftarrow$ ⟨ ABD-R–21 ⟩

$$(\mathsf{Init}, u_1, u_2).(\mathsf{Offhook}, u_1).(\mathsf{Dial}, n_1, n_3) \triangleleft ABD \triangleright \{\mathsf{ErrorTone}, u_1)\}$$
$\Leftarrow$ ⟨ ABD-S–7 ⟩

$$
\begin{array}{ll}
 & (\mathsf{Init}, u_1).(\mathsf{Offhook}, u_1).(\mathsf{Init}, u_1).(\mathsf{Dial}, n_1, n_3) \triangleleft ABD \triangleright \{\mathsf{ErrorTone}, u_1)\} \\
\Leftarrow & \quad \langle \text{ ABD-R–15 } \rangle \\
 & \\
 & (\mathsf{Init}, u_1).(\mathsf{Offhook}, u_1).(\mathsf{Dial}, n_1, n_3) \triangleleft ABD \triangleright \{\mathsf{ErrorTone}, u_1)\} \\
\Leftarrow & \quad \langle \text{ ABD-A–7 } \rangle \\
 & \\
 & \mathbf{true}
\end{array}
$$

□

5.7 Call Forward On Busy

The Call Forward On Busy feature,(CFB) let a subscriber redirect all is incoming call if and only if is line is busy. This feature is very similar to Voice Mail, however the destination may be chosen by the subscriber himself. This feature is enabled for subscriber n_1 to phone id n_2 by initiating $(\mathsf{CFBEna}, n_1, n_2)$. Similarly, it is disable by initiating (CFBDis, n_1). The input space definition of the CFB feature may be found on the next page.

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

to be the input used by CFB, where $I_{CFB} \in \mathsf{I}$.

5.7.1 Call Forward On Busy Axioms

POTS axioms are also relevant to CFB, therefore axioms TWC-A-1 through TWC-A-13 are define in tables 5.1 and 5.2 where $I_R = I_{CW}$. All other CFB axioms can be constructed from tables 5.26 and 5.27.

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

5.7.2 Call Forward On Busy Rules

We present the reduction and swap rules for CFB in section 5.7.2.1 and 5.7.2.2 respectively.

No	Premise	Input	Output
A-36		(Init, n_1, n_2, n_3).(CFBAct, n_1)	ϕ
A-37		(OffHook, n_1)	{(DialTone, n_1)}
A-38	$n_1 \neq n_2$	(CFBEna, n_1, n_2)	ϕ
A-39	$n_1 \neq n_3 \wedge n_2 \neq n_3$	(OffHook, n_3)	{(DialTone, n_3)}
A-40		(Dial, n_3, n_1)	{(Ring, n_3, n_2)}
A-41		(OffHook, n_2)	{(Conn, n_3, n_2)}
A-42	$n \in \{n_2, n_3\}$	(OffHook, n)	{(Disc, n_3, n_2)}
A-43	$n' \in \{n_2, n_3\} \wedge n' \neq n$	(OnHook, n')	ϕ

Table 5.26: Cumulative Table for a Successful CFB Connection

No	Premise	Input	Output
A-44	*distinct*(n_1, n_2, n_3)	c9.(OnHook, n_1).(Dial, n_3, n_1)	{(Ring, n_3, n_1)}
A-45	$n_1 \neq n_3$	c.(OffHook, n_1).(OffHook, n_2).(Dial, n_3, n_1)	{(BusyTone, n_3)}
A-46	*distinct*(n_1, n_2, n_3, n_4)	c.(OffHook, n_4).(Dial, n_4, n_1).(Dial, n_3, n_1)	{(Ring, n_3, n_2)}
A-47	*distinct*(n_1, n_2, n_3, n_4)	c.(OffHook, n_2).(OffHook, n_4).(Dial, n_4, n_2).(Dial, n_3, n_1)	{(BusyTone, n_3)}
A-48		(CFBDeact, n_1)	ϕ
A-49		(CFBDel, n_1)	ϕ

Table 5.27: Table for an Unsuccessful CFB Connection

5.7.2.1 Call Forward On Busy Reduction Rules

No	Condition	Prefix	Discard	Postfix
R-17	$x' \neq \varepsilon \wedge i \in 0, 1$	x	(CFBAct, n_1).(CFBEna, n_1, $-$)*.(CFBDeact, n_1)	x'
R-18	$x' \neq \varepsilon$	x	(CFBEna, n_1).(CFBDis, n_1)	x'

Table 5.28: CFB Reduction Axioms - Complete Cycle

Rule CFB-R–17 state that all input relevant to the CFB for user n_1 may be removed from the sequence when the CFB is deactivated.

5.7.2.2 Call Forward On Busy Swap Rules

The swap rules for CFB can be built using information from tables 5.5 and 5.30.

A proper validation sequence is to verify whether the call will be redirected if the user enable CFB, and the system disactivated CFB. This can be verified by the sequence CFB1:

(Init, u_1, u_2, n_3).(CFBAct, u_1).(Offhook, u_1).(CFBEna, u_1, u_2).(OnHook, u_1).
(Offhook, u_1).(CFBDeact, u_1).(Offhook, u_3).(Dial, n_3, n_1)}

which should show that the call is not forwarded.
PROOF.

No	Condition	Prefix	Discard	Postfix
R-19	$n_1 \neq n_2$	x	(CFBAct, n_1)	(Init, n_2)
R-20		x	(CFBAct, n_1)	(OffHook, n_2)
R-21		x	(CFBAct, n_1)	(Dial, n_2, n_3)
R-22		x	(CFBAct, n_1)	(OnHook, n_2)
R-23	$n_1 \neq n_2$	x	(CFBAct, n_1)	(CFBEna, n_2, n_3)
R-24	$n_1 \neq n_2$	x	(CFBAct, n_1)	(CFBAct, n_2)
R-25	$n_1 \neq n_2$	x	(CFBAct, n_1)	(CFBDeact, n_2)
R-26	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFBEna, n_2, n_3)
R-27	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFBDis, n_2)
R-28	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFBAct, n_2)
R-29	$n_1 \neq n_2$	x	(OffHook, n_1)	(CFBDeact, n_2)
R-30	$distinct(n_1, n_2, n_3)$	x	(Dial, n_1, n_2)	(CFBEna, n_3, n_4)
R-31	$distinct(n_1, n_2, n_3)$	x	(Dial, n_1, n_2)	(CFDis, n_3)
R-32	$n_1 \neq n_3$	x	(Dial, n_1, n_2)	(CFBAct, n_3)
R-33	$distinct(n_1, n_2, n_3)$	x	(Dial, n_1, n_2)	(CFBDeact, n_3)
R-34	$distinct(n_1, n_2, n_3)$	x	(CFBEna, n_1, n_2)	(Init, n_3)
R-35		x	(CFBEna, n_1, n_2)	(OffHook, n_3)
R-36	$n_1 \neq n_4$	x	(CFBEna, n_1, n_2)	(Dial, n_3, n_4)
R-37		x	(CFBEna, n_1, n_2)	(OnHook, n_3)
R-38	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(CFBEna, n_3, n_4)
R-39	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(CFBDis, n_3)
R-40	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(CFBAct, n_3)
R-41	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(CFBDeact, n_3)

Table 5.29: CFB Reduction Rules - Connection in Progress

$$(\mathsf{Init}, u_1, u_2, n_3).(\mathsf{CFBAct}, u_1).(\mathsf{Offhook}, u_1).(\mathsf{CFBEna}, u_1, u_2).(\mathsf{OnHook}, u_1).$$
$$(\mathsf{Offhook}, u_1).(\mathsf{CFBDeact}, u_1).(\mathsf{Offhook}, u_3).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{\mathsf{BusyTone}, u_3)\}$$
$\Leftarrow$ ⟨ CFB-S–25 ⟩

$$(\mathsf{Init}, u_1, u_2, n_3).(\mathsf{CFBAct}, u_1).(\mathsf{CFBEna}, u_1, u_2).(\mathsf{Offhook}, u_1).(\mathsf{OnHook}, u_1).$$
$$(\mathsf{Offhook}, u_1).(\mathsf{CFBDeact}, u_1).(\mathsf{Offhook}, u_3).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{\mathsf{BusyTone}, u_3)\}$$
$\Leftarrow$ ⟨ CFB-R–2 ⟩

$$(\mathsf{Init}, u_1, u_2, n_3).(\mathsf{CFBAct}, u_1).(\mathsf{CFBEna}, u_1, u_2).(\mathsf{Offhook}, u_1).(\mathsf{CFBDeact}, u_1)$$
$$(\mathsf{Offhook}, u_3).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{\mathsf{BusyTone}, u_3)\}$$
$\Leftarrow$ ⟨ CFB-S–18 ⟩

$$(\mathsf{Init}, u_1, u_2, n_3).(\mathsf{CFBAct}, u_1).(\mathsf{CFBEna}, u_1, u_2).(\mathsf{CFBDeact}, u_1).(\mathsf{Offhook}, u_1).$$
$$(\mathsf{Offhook}, u_3).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{\mathsf{BusyTone}, u_3)\}$$
$\Leftarrow$ ⟨ CFB-S–18 ⟩

$$(\mathsf{Init}, u_1, u_2, n_3).(\mathsf{CFBAct}, u_1).(\mathsf{CFBEna}, u_1, u_2).(\mathsf{CFBDeact}, u_1).(\mathsf{Offhook}, u_1).$$
$$(\mathsf{Offhook}, u_3).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{\mathsf{BusyTone}, u_3)\}$$
$\Leftarrow$ ⟨ CFB-R–17 ⟩

$$(\mathsf{Init}, u_1, u_2, n_3).(\mathsf{Offhook}, u_1).(\mathsf{Offhook}, u_3).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{\mathsf{BusyTone}, u_3)\}$$
$\Leftarrow$ ⟨ CFB-S–10, CFB-S–7 & CFB-S–7 ⟩

$$(\mathsf{Init}, u_1, n_3).(\mathsf{Offhook}, u_1).(\mathsf{Offhook}, u_3).(\mathsf{Init}, n_2).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{\mathsf{BusyTone}, u_3)\}$$
$\Leftarrow$ ⟨ CFB-R–15 ⟩

No	Condition	Prefix	Input 1	Input 2	Postfix
S-11	$n_1 \neq n_2$	x	(CFBAct, n_1)	(Init, n_2)	x'
S-12		x	(CFBAct, n_1)	(OffHook, n_2)	x'
S-13		x	(CFAct, n_1)	(Dial, n_2, n_3)	x'
S-14	$n_1 \neq n_2$	x	(CFBAct, n_1)	(OnHook, n_2)	x'
S-15	$n_1 \neq n_2$	x	(CFBAct, n_1)	(CFBEna, n_2, n_3)	x'
S-16	$n_1 \neq n_2$	x	(CFBAct, n_1)	(CFBAct, n_2)	x'
S-17	$n_1 \neq n_2$	x	(CFBAct, n_1)	(CFBDeact, n_2)	x'
S-18	$n_1 \neq n_2$	x	(CFBDeact, n_1)	(OffHook, n_2)	x'
S-19	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	(CFBDeact, n_1)	(Dial, n_2, n_3)	x'
S-20	$n_1 \neq n_2$	x	(CFBDeact, n_1)	(OnHook, n_2)	x'
S-21	$n_1 \neq n_2$	x	(CFBDeact, n_1)	(CFBEna, n_2, n_3)	x'
S-22	$n_1 \neq n_2$	x	(CFBDeact, n_1)	(CFBDis, n_2)	x'
S-23	$n_1 \neq n_2$	x	(CFBDeact, n_1)	(CFBDeact, n_2)	x'
S-24	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(Init, n_3)	x'
S-25	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(OffHook, n_3)	x'
S-26	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(Dial, n_3, n_4)	x'
S-27	$n_1 \neq n_2$	x	(CFBEna, n_1, n_2)	(OnHook, n_3)	x'
S-28	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(CFBEna, n_3, n_4)	x'
S-29	$n_1 \neq n_3$	x	(CFBEna, n_1, n_2)	(CFBDis, n_3)	x'
S-30	$n_1 \neq n_2$	x	(CFBDis, n_1)	(Init, n_2)	x'
S-31	$n_1 \neq n_2$	x	(CFBDis, n_1)	(OffHook, n_2)	x'
S-32	$n_1 \neq n_2 \wedge n_1 \neq n_3$	x	(CFBDis, n_1)	(Dial, n_2, n_3)	x'
S-33	$n_1 \neq n_2$	x	(CFBDis, n_1)	(OnHook, n_2)	x'
S-34	$n_1 \neq n_2$	x	(CFBDis, n_1)	(CFBDis, n_2)	x'

Table 5.30: CFB Swap Rules

$$(\mathsf{Init}, u_1, n_3).(\mathsf{Offhook}, u_1).(\mathsf{Offhook}, u_3).(\mathsf{Dial}, n_3, n_1) \triangleleft CFB \triangleright \{(\mathsf{BusyTone}, u_3)\}$$
$$\Leftarrow \quad \langle \text{ CFB-A–8 } \rangle$$

true □

Part III

Analyzing Feature Interactions

Chapter 6

Characterizing Feature Interaction

A solution to the feature interaction problem is becoming crucial to the survival of many systems which new functionalities are added on a regular basis. In this chapter, we give a relational specification based definition of feature interaction and then we will show various interactions between the features specified in Chapter 5

6.1 Definition of Feature Interaction

Before presenting the feature interaction problem lets define feature. A *feature* is a collection of functionalities that are added incrementally in a software system. A feature *interaction* occurs whenever their specifications contradict each other by producing conditions that no behavior can satisfy. In other words an interaction arises whenever there exists an input sequence where no one output can satisfy both.

All features should be defined independently. This is from the notion that in a perfect situation a feature added to the system should not interfere with any other feature(s) that already exist. To be able to detect or to prove that there exists no feature interaction between two distinct feature they must be defined independently and that a definition of the combination of all the feature exists. In our relational framework, each feature is given by a relation, and the combination of the features is given by the least upper bound of the features [3]. By intuition, it can be shown that a program p which satisfies the combination of all features must also satisfy them individually. Hence, we have $P \sqsubseteq R_i$ for each feature R_i. It follows that P is an upper bound, in the refinement ordering, of the set of features $\{R_1, \ldots, R_n\}$. A sound representation of the combination of all features is the least upper bound, since a program refines the least upper bound if and only if it refines each feature individually —by definition. Stated formally, we have:

$$(\bigwedge_{i=1}^{n} P \sqsubseteq R_i) \;\Leftrightarrow\; P \sqsubseteq \bigsqcup_{i=1}^{n} R_i \;.$$

It is the specifiers responsibility to ensure that there exist a least upper bound when specifying the features independently thus there exists a program that satisfies the set of features.

Definition 6.1 *Given a set* $\{R_1, \ldots, R_n\}$ *of relational specifications of features, we say that there is an interaction between these features if and only if* $\neg cs(R_1, \ldots, R_n)$. *We say that these features interact for input sequence* s *if and only if*

$$(\bigwedge_{i=1}^{n} s \in dom(R_i)) \ \wedge \ \neg\exists s' : \bigwedge_{i=1}^{n} (s, s') \in R_i \ .$$

A collection of features interact for some input sequence s if they are all defined for s, and if they cannot agree on one common output. We have found that several examples of feature interactions listed in [11] and [20], which contains a classification of feature interactions, satisfy Def. 6.1.

Definition 6.1 is strong enough to cover cases of interaction between s features even if none can be detected for any subset of $n-1$ features. For instance, assume that we have 3 features R_1, R_2, R_3 with the following behavior:

$s \triangleleft R_1 \triangleright 1 \quad s \triangleleft R_1 \triangleright 2$
$s \triangleleft R_2 \triangleright 2 \quad s \triangleleft R_2 \triangleright 3$
$s \triangleleft R_3 \triangleright 1 \quad s \triangleleft R_3 \triangleright 3$

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

6.2 Categorization of the Feature Interaction Problem

To better understand the feature interaction problem in telecommunication systems, E.J. Cameron et al [10] have categorized feature interactions in two ways:

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

6.2.1 Categorization By the Nature of Interactions

The categorization by the nature of interactions has three dimensions

- the kind of features involved in the interaction
- the number of users involved in the interaction
- the number of network components involved in the interaction

The first dimension distinguishes between interactions that involve only *customer features* from interactions that involve *system features*, instead of, or in addition to, customer features. Customer features include all the call processing features visible to the general public such as Call Waiting, Call Forward Busy, Three-Way Calling, Originated Call Screening, Abreviated Dialing.

The second dimension distinguishes between *single-user* interaction from *multiple-user* interactions. The former are caused when different features are simultaneously activated by a single user, while the latter is cause when, a feature activated by a user interferes with features activated by another user.

The third dimension distinguishes between *single-component* interactions and *multiple-component* interactions. The former is caused by a single-component, while the latter is caused when features supported on a network component interfere with the operations of features supported on another network component.

Some of the possible interaction types are:

- SUSC (Single-User-Single-Component) are interactions which occur when a single user activates different features on a single network.
- SUMC (Single-User-Multiple-Component) are interactions which occur when a single user activates a feature which interferes with the operation of a feature on another network.
- MUSC (Multiple-User-Single-Component) are interactions which occur when a feature activated by a user interferes with a feature activated by another user on the same network.
- MUMC (Multiple-User-Multiple-Component) are interactions which occur when a feature activated by a user interferes with a feature activated by another user on a different network.
- CUSY (CUstomer-SYstem) are interactions which occur when a customer feature interferes with any feature for operations, administration service or maintenance.

6.2.2 Categorization By the Cause of Interactions

The categorization by cause of interactions include:

- violation of assumption about system,
- limitations on network support,
- and problems due to large heterogeneous distributed system.

Features in a Telecommunications network need to operate under a set of assumptions such as naming, data availability, administrative domain, call control and signaling protocol, once violated can result in some interactions.

6.2.2.1 Limitation of Network Support

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

The set of signals for switches is limited to *,#,the ten digit 0-9,, flashhook, and disconnect. However, telecommunication systems have many services which are restricted to these signals. Therefore the same signal is often used to mean different things in different contexts. Interactions arise when the same signal has different meanings in the context of different concurrently active features.

6.2.2.2 Intrinsic Problems in Distributed Systems

Telecomunications systems are huge, real-time, reactive, distributed systems. Many difficulties are caused because of resource contention, personalized instantiation, timing and race conditions, as well as non-atomic operation.

6.3 Sample of Feature Interaction

In the next section we will prove the feature interaction problem between Three Way Calling and Call Waiting, Originating Call Screening and Abbreviated Dialing , Call Forward Busy and Call Waiting, and Call Forward Busy and Originating Call Screening.

6.3.1 Three-Way Calling and Call Waiting

The feature interaction problem between TWC and CW is one of the most known in the literature. Since signaling capability of a telephone is limited, often a signal can mean different things depending on which feature is anticipated. The signal FlashHook issued by a user u_1 engaged with user u_2 could mean that user u_1 wishes to add a third party to an established call (TWC) or to accept a connection attempt from a new caller while putting u_2 on hold. The problem occurs for the following input sequence, which is denoted by s_1

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{TWCAct}, u_1).(\mathsf{CWAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).$$
$$(\mathsf{OffHook}, u_2).(\mathsf{OffHook}, u_3).(\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1)$$

In feature Three-Way Calling, the only output (since this feature is deterministic) corresponding to s_1 is:

$$\{(\mathsf{DialTone}, u_1), (\mathsf{Hold}, u_1, u_2)\} \ .$$

PROOF.

$$\begin{array}{ll}
& (\mathsf{Init}, u_1, u_2, u_3).(\mathsf{TWCAct}, u_1).(\mathsf{CWAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2). \\
& (\mathsf{OffHook}, u_3).(\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1) \lhd TWC \rhd (\mathsf{DialTone}, u_1), (\mathsf{Hold}, u_1, u_2) \\
\Leftarrow & \quad \langle \text{ TWC-R-1 } \rangle \\
& (\mathsf{Init}, u_1, u_2, u_3).(\mathsf{TWCAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2)(\mathsf{OffHook}, u_3). \\
& (\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1) \lhd TWC \rhd (\mathsf{DialTone}, u_1), (\mathsf{Hold}, u_1, u_2) \\
\Leftarrow & \quad \langle \text{ TWC-R-35 } \rangle \\
& (\mathsf{Init}, u_1, u_2, u_3).(\mathsf{TWCAct}, u_1).\mathsf{c1}.(\mathsf{OffHook}, u_3).(\mathsf{FlashHook}, u_1) \\
& \quad \lhd TWC \rhd (\mathsf{DialTone}, u_1), (\mathsf{Hold}, u_1, u_2) \\
\Leftarrow & \quad \langle \text{ TWC-R-32 } \rangle \\
& (\mathsf{Init}, u_1, u_2, u_3).(\mathsf{TWCAct}, u_1).\mathsf{c1}.(\mathsf{OffHook}, u_2).(\mathsf{FlashHook}, u_1) \\
& \quad \lhd TWC \rhd (\mathsf{DialTone}, u_1), (\mathsf{Hold}, u_1, u_2) \\
\Leftarrow & \quad \langle \text{ TWC-A-15 } \rangle \\
& \mathbf{true}
\end{array}$$

□

In feature Call Waiting, the only output (CW is also deterministic) corresponding to s_1 is:

$$\{(\mathsf{Conn}, u_1, u_3), (\mathsf{Disc}, u_1, u_2)\} \ .$$

PROOF.

$$\begin{array}{ll}
& (\mathsf{Init}, u_1, u_2, u_3).(\mathsf{TWCAct}, u_1).(\mathsf{CWAct}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2).(\mathsf{OffHook}, u_3). \\
& (\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1) \lhd CW \rhd (\mathsf{Conn}, u_1, u_3), (\mathsf{Hold}, u_1, u_2) \\
\Leftarrow & \quad \langle \text{ CW-R-1 } \rangle \\
& (\mathsf{Init}, u_1, u_2, u_3).(\mathsf{CWAct}, u_1).(\mathsf{Dial}, u_1, u_2).(\mathsf{OffHook}, u_2).(\mathsf{OffHook}, u_3). \\
& (\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1) \lhd CW \rhd (\mathsf{Conn}, u_1, u_3), (\mathsf{Hold}, u_1, u_2) \\
\Leftarrow & \quad \langle \text{ CW-A-16 } \rangle \\
& \mathbf{true}
\end{array}$$

□

From proposition 3.2 and corollary 3.1 it can be shown that the consistency condition between TWC and CW is not satisfied for input sequence s_1. Thus, there does not exist a least upper bound for these two features, nor does there exist a program that satisfies both features simultaneously. There are other sequences causing an interaction between CW and TWC which are shown in table 6.1.

Axiom TWC-A-13 and axiom CW-A-13 both say that for any input which is not an element of there respective input space, that the specifier is indifferent to the output return. Therefore both features can return a different output for the same sequence. However this does not mean that they interact. Equation 6.1 explicitly informs us that there is feature interaction if and only if no common output s' can be produced by input s. A common s' is possible since the

No	Sequence	TWC Output	CW Output
S-1	(Init, u_1, u_2, u_3).(TWCAct, u_1).(CWAct, u_1).(OffHook, u_1).(Dial, u_1, u_2).(OffHook, u_2).(OffHook, u_3).(Dial, u_3, u_1).(FlashHook, u_1)	{(DialTone, u_1), (Hold, u_1, u_2)}	{(Conn, u_1, u_3), (Disc, u_1, u_2)}
S-2	(Init, u_1, u_2, u_3).(TWCAct, u_1).(CWAct, u_1).(OffHook, u_1).(Dial, u_1, u_2).(OffHook, u_2).(OffHook, u_3).(Dial, u_3, u_1)	{(BusyTone, u_1)}	{(Ring, u_3), (WaitTone, u_1)}

Table 6.1: Sequences that interact between TWC and CW.

specifier really does not care about the output. Therefore it can be concluded that no features will interact on a input sequence s if either one or both have a *undefined* (U) output.

6.3.2 Originating Call Screening and Abbreviated Dialing

The features OCS and ABD interact when the features logic are combined together. For example if a user with the phone id u_1 does not want is kid to call the user with the phone id u_2. Therefore OCS is invoked to deny user u_1 to connect to user u_2. However user u_1 has also invoke ABD and as abbreviated the phone id u_3 to the phone id u_2. OCS may check the abbreviated phone id u_3 and find that it is not in the OCS database, after which ABD translate the abbreviated phone id u_3 into phone id u_2 which is in the OCS database, and a connection is set up between users u_1 and u_2

(Init, u_1, u_2, u_3).(ABDAct, u_1).(OCSAct, u_1).(ABDAdd, u_1, u_2, u_3).(OCSAdd, u_1, u_2).
(OffHook, u_1).(Dial, u_1, u_3).(OffHook, u_2)

In this sequence, u_3 is an alias for u_2 and OCS prevents calls from u_1 to u_2. The interaction occurs when u_1 dials u_2. Feature ABD will redirect this call to n_2, but feature OCS prevents that. In ABD, the output for the sequence above is {(Conn, u_1, u_2)}.

PROOF.

(Init, u_1, u_2).(ABDAct, u_1).(OCSAct, u_2).(ABDAdd, u_1, u_2, u_3).(OCSadd, u_1, u_2).
(OffHook, u_1).(Dial, u_1, u_3).(OffHook, u_2)$\triangleleft ABD \triangleright$ {(Conn, u_1, u_2)}
$\Leftarrow$ ⟨ ABD-R-1 ⟩

(Init, u_1, u_2).(ABDAct, u_1).(ABDAdd, u_1, u_2, u_3).(OffHook, u_1).(Dial, u_1, u_3).
(OffHook, u_2)$\triangleleft ABD \triangleright$ {(Conn, u_1, u_2)}
$\Leftarrow$ ⟨ ABD-A–18 ⟩

true □

By OCS axiom OCS-A–23 that this is not a valid output, hence the consistency condition does not hold.

PROOF.

$$(\mathsf{Init}, u_1, u_2).(\mathsf{ABDAct}, u_1).(\mathsf{OCSAct}, u_2).(\mathsf{ABDAdd}, u_1, u_2, u_3).(\mathsf{OCSadd}, u_1, u_2).$$
$$(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_3).(\mathsf{OffHook}, u_2) \triangleleft ABD \triangleright \{(\mathsf{Conn}, u_1, u_2)\}$$
$$\Leftarrow \quad \langle \text{ OCS-A–23 } \rangle$$

false

□

Note that our specification of OCS may seem to catch the inconsistency one input element "too late". For instance, in the example above, OCS allows the phone a_3 to ring, but it does not allow to make a connection when a_3 is picked up. However, it does not really matter at what point the interaction is picked up, as long as it is at some point. After an interaction is detected, the specifier should review the entire input sequence causing it, not only the last symbol.

6.3.3 Call Forward on Busy and Originating Call Screening

The feature interaction problem between CFB and OCS exists when user u_1 adds phone id u_2 to the OCS database and then user u_1 forwards the call to a phone id which as forwarded is call to u_2. This situation occurs with the following input sequence:

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{CFBAct}, u_1).(\mathsf{OCSAct}, u_2).(\mathsf{OffHook}, u_1).(\mathsf{CFBEna}, u_1, u_3).$$
$$(\mathsf{OCSadd}, u_2, u_3).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_2, u_1).(\mathsf{OffHook}, u_3)$$

The only output possible (since CFB is deterministic) for the above sequence using the CFB specification is $\{(\mathsf{Conn}, u_2, u_3)\}$.

PROOF.

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{CFBAct}, u_1).(\mathsf{OCSAct}, u_2).(\mathsf{OffHook}, u_1).(\mathsf{CFBEna}, u_1, u_3).$$
$$(\mathsf{OCSadd}, u_2, u_3).(\mathsf{OffHook}, u_2).(\mathsf{Dial}, u_2, u_1).(\mathsf{OffHook}, u_3) \triangleleft CFB \triangleright \{(\mathsf{Conn}, u_2, u_3)\}$$
$$\Leftarrow \quad \langle \text{ CFB-R-1 } \rangle$$

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{CFBAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{CFBEna}, u_1, u_3).(\mathsf{OffHook}, u_2).$$
$$(\mathsf{Dial}, u_2, u_1).(\mathsf{OffHook}, u_3) \triangleleft CFB \triangleright \{(\mathsf{Conn}, u_2, u_3)\}$$
$$\Leftarrow \quad \langle \text{ CFB-A–41 } \rangle$$

true

□

However using only one axiom OCS-A–23 it can be shown that this output is not acceptable, therefore the consistency condition does not hold.

6.3.4 Three-Way Calling and Originating Call Screening

An interaction between the TWC and OCS features is possible. Assume that user u_1 is subscribe to TWC, and that the user at phone id u_2 is block from calling user u_3. It is possible

for users u_1 of establishing a three way call and indirectly a connection between u_2 and u_3. The sequence presented below describes this situation.

$$(\mathsf{Init}, n_1, n_{2,3}).(\mathsf{TWCAct}, n_1).(\mathsf{OCSAct}, n_2).(\mathsf{OCSAdd}, n_2, n_3).(\mathsf{OffHook}, n_1).$$
$$(\mathsf{Dial}, n_1, n_2).(\mathsf{OffHook}, n_2).(\mathsf{FlashHook}, n_1).(\mathsf{Dial}, n_1, n_3).(\mathsf{OffHook}, n_3).$$
$$(\mathsf{FlashHook}, n_1)$$

The feature TWC generates by applying the TWC-R–1 and TWC-A–18 rules the output $\{(\mathsf{Conn}, n_1, n_2), (\mathsf{Conn}, n_2, n_3)\}$. It is obvious by axiom OCS-A–23 that the second part is not allowed by the OCS feature. Therefore the consistency condition does not hold.

6.3.5 Call Forward on Busy and Call Waiting

An interaction occurs when the same user is subscribed to both Call Waiting and Call Forward on Busy. To demonstrate this, let sequence CFCW be defined as:

$$(\mathsf{Init}, n_1, n_2, n_3).(\mathsf{CWAct}, n_1).(\mathsf{CFBAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{CFBEna}, n_1, n_3).$$
$$(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_2, n_1)$$

If the input sequence CFCW is entered, the system may generate a $(\mathsf{WaitTone}, n_1)$ or $(\mathsf{Ring}, n_2\ n_3)$.

PROOF.

$$(\mathsf{Init}, n_1, n_2, n_3).(\mathsf{CWAct}, n_1).(\mathsf{CFBAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{CFBEna}, n_1, n_3).$$
$$(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_2, n_1) \triangleleft CW \triangleright \{(\mathsf{WaitTone}, n_1)\}$$
$$\Leftarrow \quad \langle \text{ CW-R–1 } \rangle$$

$$(\mathsf{Init}, n_1, n_2, n_3).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_2, n_1)$$
$$\triangleleft CW \triangleright \{(\mathsf{WaitTone}, n_1)\}$$
$$\Leftarrow \quad \langle \text{ CW-S–11, CW-S–7 \& CW-S–7 } \rangle$$

$$(\mathsf{Init}, n_1, n_2).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{OffHook}, n_2).(\mathsf{Init}, n_3).$$
$$(\mathsf{Dial}, n_2, n_1) \triangleleft CW \triangleright \{(\mathsf{WaitTone}, n_1)\}$$
$$\Leftarrow \quad \langle \text{ CW-R–15 } \rangle$$

$$(\mathsf{Init}, n_1, n_2).(\mathsf{CWAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{OffHook}, n_2).(\mathsf{Init}, n_3).$$
$$(\mathsf{Dial}, n_2, n_1) \triangleleft CW \triangleright \{(\mathsf{WaitTone}, n_1)\}$$
$$\Leftarrow \quad \langle \text{ CW-A–15 } \rangle$$

true

□

PROOF.

$$(\mathsf{Init}, n_1, n_2, n_3).(\mathsf{CWAct}, n_1).(\mathsf{CFBAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{CFBEna}, n_1, n_3).$$
$$(\mathsf{OffHook}, n_2).(\mathsf{Dial}, n_2, n_1) \triangleleft CFB \triangleright \{(\mathsf{Ring}, n_2, n_3)\}$$
$$\Leftarrow \quad \langle \text{ CFB-R–1 } \rangle$$

$$(\mathsf{Init}, n_1, n_2, n_3).(\mathsf{CFBAct}, n_1).(\mathsf{OffHook}, n_1).(\mathsf{CFBEna}, n_1, n_3).(\mathsf{OffHook}, n_2).$$
$$(\mathsf{Dial}, n_2, n_1) \triangleleft CFB \triangleright \{(\mathsf{Ring}, n_2, n_3)\}$$
$$\Leftarrow \quad \langle \text{ CFB-A–40 } \rangle$$

true □

6.3.6 Categorization of the Interaction

In section 6.2. we described a technique that allows the categorization of the feature interaction problem. In this section, we will categorize the feature interactions that were described in this chapter.

The cause of the interaction between CW and TWC, is a limitation on the network support, while the nature of interaction is SUSC. CW and TWC interpret the flashook signal differently. The former uses the flashook signal to toggle between two connected callers while the latter uses it to first initiate a third party call, and secondly to connect all three parties.

The cause of the interaction between OCS and ABD is a problem in the distributed systems, while the nature of interaction is MUSC. Even though a user assigned number n_1 to be screened, a user may be able connect to number n_1 if the number n_1 was aliased to another number which is not assigned to be screened.

The cause of the interaction between OCS and CFB is a problem in the distributed systems, while the nature of interaction is MUSC. Similar to the interaction between OCS and ABD, a user which assigned number n_1 to be screened may be able to connect to number n_1 if the user tries to connect to a number which has redirected the call to the number n_1. The cause of the interaction between TWC and OCS is a problem in the distributed systems, while the nature of interaction is MUSC. Again even though a user's assigned number n_1 to be screened, a user may be able to connect to number n_1 by means of a third party which initiated a three way call between the original user, the user at n_1 and himself.

The cause of the interaction between CW and CFB, is a violation of feature assumptions, while the nature of interaction is SUSC. Let user n_1 be subscribed to both CW and CFB, and that CFB is enable to redirect calls to user n_2. If a user n_1 is already on the line, when a second call comes in, should user n_1 receive a WaitTone or should the call be redirected to user n_2?

Chapter 7

Automatic Detection of Feature Interaction

In this chapter we present a method to automatically detect interactions between features. According to our definition of feature interaction (Def. 6.1), one could detect interactions between feature CW and TWC by proving that the following formula holds:

2 $\exists s : s \in dom(CW) \wedge s \in dom(TWC) \wedge \neg\exists s' : (s, s') \in CW \wedge (s, s') \in TWC$.

This formula will be proven with the help of logic programming and SLD-resolution. Since the TWC and CW axioms and rules are in the form of clauses (i.e. universally quantified formulas of the form $A \Leftarrow B$), and the formula we wish to prove is a goal, these can be easily translated into a logic program.

7.1 The Prolog Connection

A simple program written in any programming logic language such as Prolog and LISP can detect interaction between a set of features. In this section, we discuss how this can be accomplished using Prolog.

7.1.1 Prolog Semantic

Prolog's *facts* are very similar to our definition of axioms. A prolog fact shows relations between two objects. Some rules must be followed when providing Prolog with facts. These simple, however very important rules are called syntax rules. The name that defines the relationship appears first, then the names of the object appear within parentheses, separated by commas followed by a full stop which is denoted by a period.

Syntax : facts

- relationship name first

- object in parentheses, separated by commas
- period at the end (full stop)

Example :

```
likes(joe,mary).
```

The relationship name and object names start with lower-case letters, which indicate to Prolog that these represent names of specific objects and that they will not change; they are called constants. An underscore may be used to represent a space within a name (ie. british_columbia).

The query *likes(joe,mary)* may either mean that Joe likes Mary or that Mary likes Joe. It is the responsibility of the programmer to decide the proper interpretation and be consistent throughout the program.

Facts are the basis for Prolog answers to a query. Facts are statements that are true in our database, and thus fundamental building blocks of the database that describes our system or features. We can expand the description of the system we have defined by adding *rules*. These rules, which are built on facts or on other rules and facts, add another dimension to the informational power of Prolog's responses.
A Prolog rule has two parts:

1. a conclusion and
2. the requirements for the conclusion.

If the requirement component is true implies that the conclusion must also be true, therefore if all the requirement are true the conclusion must be true.
Syntax : rules

- head
- :-
- one or more requirements, separated by commas (and) or semi-colon (or)
- full stop

<conclusion> :- <requirements>.

Variables can stand of place are various objects. Variables can be recognized in Prolog code since they must begin with a capital letter. They may be used in rules and queries. Queries are used to ask questions about the system from Prolog. Queries follow the same rules as facts except that the must be prefix by ?-.

Example

```
likes (joe,mary).
?- likes(joe, mary).
yes
```

Another very useful operator is the "*not*" which is denoted by forward slash followed by a plus sign, and the "*don't care*" variable denoted by the underscore or a variable name starting with an underscore.

7.1.2 Relational Calculus to Prolog

Lists are used to represent sequences, sets and tuples, and we use the predicate *append* to manipulate lists. Since there are many details involved, the method to translate the specification and the formula 2 into Prolog is presented by the help of examples.

Knowing the valid input for the feature is very important to certain axioms and rules therefore the input space may be represented by simple facts containing only one object. Thus the TWC input space can be represented by :

```
twcInput(twcAct,\_N1).
twcInput(twcDeact,\_N1).
twcInput(flashHook,\_N1).
twcInput(X) :- potsInput(X).
```

Note that the last entry is not a fact but rather a rule. In this thesis we have decided for sake of simplicity that POTS is a subset of all other features. However, POTS specifications have been separated from other features throughout the thesis. This will make it an easier process when converting to a more realistic telephone system. In reality TWC is added to POTS therefore inheriting all of POTS functionality and inputs just as it is defined here.

The axioms can be represented by Prolog facts. For instance, the axiom TWC-A-15 in Table 5.6 translates into the following Prolog clause in Edinburgh syntax:

```
twc([[Init,N1],[Init,N2],[TWCAct,N1],[OffHook,N1],[Dial,N1,N2],
[OffHook,N2],[FlashHook,N1]], [[DialTone,N1],[Hold,N1,N2]])
      :- N1\==N2.
```

In the clause above, relation TWC is represented by binary predicate twc. Its first argument is the input sequence, which is given as a list of lists. The inner lists represent tuples of I. The second argument of twc is the output, given again by a list of lists, since the outputs of our telephone system are sets. Unfortunately the TWC-A-13 is not so straight forward. In reality this is the only axiom which may not be represented with a fact. This axioms is represented by the following Prolog rule :

```
twc(X,[[u]])
        :- \+last(Element,X),
           twcInput(Element).
```

which states that if the last input of the list is not a valid input for twc, that the system may return anything. This is denoted by u which mean undefined.
The reduction rules TWC-R-35 and TWC-R-28 are translated to

```
twc(X,Y) :- Post=[[FlashHook,N3]],
            append(X1,Post,X),
            append(Pre,[[dial,N1,N2]],X1),
            N1\==N3,
            append(Pre,Post,X2),twc(X2,Y).
```

and

```
twc(X,Y) :- append([[twcAct,N1],[twcDeact,N1]],Post,Rem),
            append(Pre,Rem,X),
            Post \== [[]],
            append(Pre,Post,X2).
```

Both the complete cycle and incomplete cycle reduction rules can be represented by a general rule. In fact, the general rule can be written in such a way that a direct mapping from the tabular format used in this thesis is possible. These and other rules may be found in an appendix attached at the end of this thesis.

Once again, a special reduction rule is needed when an invalid input is found in the middle of the sequence which in our specification is handled be TWC-R-1 rule. This can be handled in Prolog by the following rule :

```
twc(X,Y) :-     lastIsTwcInput(X),
                append(Head,Tail,X),
                checkTwcSeq(Head,X1),
                append(X1,Tail,X2),
                twc(X2,Y).
```

Swap rules can be translated in much the same way as the complete cycle reduction rules. For instance rule TWC-S-11 can be represented in Prolog by :

```
twc(X,Y):-      Input1 = [[init,\_N1]], Input2 = [[twcAct]],
                append(Input1,Input2,Temp1),
                append(Input2,Input1,Temp2),
                append(Temp1,Post,Rem),
                Post \== [[]],
                append(Pre,Rem,X),
                append(Pre,Temp2,Temp3),
                append(Temp3,Post,X2).
                twc(X2,Y).
```

Axioms and rules can be easily translated into Prolog clauses following this pattern. Eq. 2 is translated in the following conjunction of goals:

```
cw(X,\_),twc(X,\_),not((cw(X,Y),twc(X,Y))).
```

The meta-predicate `not` succeeds if the proof of the conjunction of its goal arguments fails. However since an undefined output is actually a join of all possible outputs which can be generated by the system, the following goal was used :

```
cw(X,_),twc(X,_),not((cw(X,Y),twc(X,Y))), Y \== [[u]].
```

7.2 Proving Feature Interaction

The search tree generated by SDL-resolution using the linear search which is used by Prolog is lengthy. It adds unnecessary complexity, therefore we content ourselves with listing the axioms (clauses) involved in a successful proof of the conjunction of goals.

Using sequence s_1 define in Section 6.3.1

$$(\mathsf{Init}, u_1, u_2, u_3).(\mathsf{TWCAct}, u_1).(\mathsf{CWAct}, u_1).(\mathsf{OffHook}, u_1).(\mathsf{Dial}, u_1, u_2).$$
$$(\mathsf{OffHook}, u_2).(\mathsf{OffHook}, u_3).(\mathsf{Dial}, u_3, u_1).(\mathsf{FlashHook}, u_1)$$

By following the proofs also given in Section 6.3.1, we show that `twc(X,_)` succeeds, by applying TWC-R-1, TWC-R-35, TWC-R-32 and TWC-A-15, and that `cw(X,_)` succeeds, by applying CW-R-1, and CW-A-16. The goal given by the meta-predicate `not` fails, since there is no axiom that allows TWC and CW to produce a common output for the current substitution of `X`. Therefore the goal `not(...)` which represents that feature interaction has been detected.

7.3 Prolog Experimentations

Some problems may arise from using a Prolog program to automatically detect interaction between features. Prolog's linear search algorithm implies looping over the first recursive axiom, hence an infinite tree may be generated without ever reaching a successful conclusion. The next few sections describe problems encountered while running some experiments

7.3.1 The Validation Experiment

The first experiment was to try to validate a specification. The full POTS specification was used for this experiment. This was to keep the experiment as simple as possible, and to verify that the Prolog program would return whether the sequence was a POTS input sequence. The basic library, lists library, and the fct.ari module found in Appendix A. must be loaded for all experiments discussed in this thesis. A predetermined sequence is used to verify that the rules reflect the system which was specified, hence the name "The Validation Experiment". What was concluded from this experiment is that the swap rule causes the program to loop indefinitely. With a partial specification of the POTS it is shown that Prolog may enter into an infinite looping state.

Let query q_1

$$? - pots([[init, n1], [init, n2], [offHook, n2], [offHook, n1], [onHook, n2]], [[]]).$$

let the following be some POTS rule and facts

```
...

pots([[init,n1],[offHook,n1],[onHook,n1]],[[]]).                F1

...

pots(X,Y) :- Post=[[onHook,N2]],                                F2
             append(X1,Post,X),
             append(Pre,[[offHook,N1]],X1),
             N1\==N2,
             append(Pre,Post,X2),pots(X2,Y).
...

pots(X,Y) :- Post=[[onHook,N2]],                                F3
             append(X1,Post,X),
             append(Pre,[[init,N1]],X1),
```

```
            N1\==N2,
            append(Pre,Post,X2),pots(X2,Y).
...

pots(X,Y):- Input1 = [[init,N1]], Input2 = [[init,N2]],      F4
            append(Input1,Input2,Temp1),
            append(Input2,Input1,Temp2),
            append(Temp1,Post,Rem),
            Post \== [[]],
            N1\==N2,
            append(Pre,Rem,X),
            append(Pre,Temp2,Temp3),
            append(Temp3,Post,X2).
            twc(X2,Y).
...

pots(X,Y):- Input1 = [[init,N1]], Input2 = [[offHook,N2]], F5
            append(Input1,Input2,Temp1),
            append(Input2,Input1,Temp2),
            append(Temp1,Post,Rem),
            Post \== [[]],
            N1\==N2,
            append(Pre,Rem,X),
            append(Pre,Temp2,Temp3),
            append(Temp3,Post,X2).
            twc(X2,Y).
```

Query q_1 will never terminate. This can be demonstrated by the following simple trace:

```
Q1
     -> [[init,n1], [init,n2],[offHook,n2],[offHook,n1],[onHook,n2]],
[[]]
F1,...,F2
     -> [[init,n1], [init,n2],[offHook,n2],[onHook,n2]],[[]]
F1,...,F2,...,F3,...,F4
     -> [[init,n1], [init,n2],[offHook,n2],[onHook,n2]],[[]]
F1,...,F2,...,F3,...,F4
     -> [[init,n2], [init,n1],[offHook,n2],[onHook,n2]],[[]]
F1,...,F2,...,F3,...,F4
```

```
     -> [[init,n1], [init,n2],[offHook,n2],[onHook,n2]],[[]]
...
```

It can be readily seen that input [init,n1] and input [init,n2] will be swap interchangeably until the system runs out of resources. A mechanism has to be written so the program does not return to the previous state. This can be achieved by incorporating C-like code with Prolog. It would be ridiculous to keep track of all previous sequences since it would decrease performance or even terminate due to lack of resources. However, we propose that the sequence used to call the swap rule be saved to a list when entering the rule. Then a rule should be written to verify that the input sequence used is not a member of this list, which the swap uses to make it fail. This list should be reset any time a reduction rule is called since it is impossible to return to a previous sequence and this will save significant resources. Now tracing query q_1 should give:

```
Q1
     -> [[init,n1], [init,n2],[offHook,n2],[offHook,n1],[onHook,n2]],
[[]]
F1,...,F2
     -> [[init,n1], [init,n2],[offHook,n2],[onHook,n2]],[[]]
F1,...,F2,...,F3,...,F4
     -> [[init,n2], [init,n1],[offHook,n2],[onHook,n2]],[[]]
  list: [[init,n1], [init,n2],[offHook,n2],[onHook,n2]],[[]]
F1,...,F2,...,F3,...,F4,...,F5
     -> [[init,n2], [offHook,n2],[init,n1],[onHook,n2]],[[]]
  list: [[init,n1], [init,n2],[offHook,n2],[onHook,n2]],[[]],
        [[init,n2], [init,n1],[offHook,n2],[onHook,n2]],[[]]
F1,...,F2,...,F3
     -> [[init,n2],[offHook,n2],[init,n1],[onHook,n2]],[[]]
  list: [ ]
F1
     -> yes
```

A similar experiment was run without the swap rules. The program was successful in determining whether the sequence was a POTS input sequence. This was achieved by carefully constructing an input sequence, which could be reduced without the help of swap rules.

7.3.2 A Simple Prolog Program for Detecting FI

A simple program (found in Appendix B) was used to verify that Prolog can indeed detect feature interactions. At first, the experiment was not successful. It was found that the program would add the input "[twcAct,N1]" only at the beginning of the sequence. Only after removing

the "[init,N1]" from all axioms was the program successful in detecting feature interaction. The reduction rules "reducecomp" and "reduceinc" must be modified to prevent the program from enterering an infinite loop.

7.3.3 Additional Experimentation

A few more experiments were run. Another problem encountered was with the ???-red-null reduction type rules. These rules remove all foreign input from an input sequence. Three versions of these rules were implemented, this thesis discusses only two of them. The first method would remove anything foreign to the feature. No extra work was needed to accomplish this. This method is a straight mapping from the specification found in chapter 5. This method was more than adequate when validating a sequence, however after a few experiments it was found that the insertion of foreign inputs was needed to allow the program to detect interaction between two features. This was achieved by adding to the controlling module (Appendix F) a rule call "universalInput". This would allow the program to insert inputs from other features into the input sequence while keeping the specification independent from one another. Even though this experiment did not run to completion (for reasons mentioned in the previous sections), it was successful in showing that the program does indeed add system input to the sequence.

Chapter 8

Conclusion

8.1 Summary

In this thesis, we have proposed a definition of feature interaction as well as a method to automatically detect interaction between features, which were both based on relational calculus and the refinement lattice. Because these foundations have been studied for system development and have been around for many years, our definition lays on a stable and solid theoretical basis.

Detecting feature interaction was accomplished by independently specificying telecommunication features which in turn were validated. Specification as a mapping between input-output pairs does not allow us to describe the behaviour of a software system. Relational specifications for dynamic systems which map the history of the inputs to an output was deemed more appropriate to detect feature interactions. The tabular format for writing the specifications make our specifications simple to read and quite concise. It also makes it easy to reuse a set of axioms by referring to the appropriate table, thus making the set of axioms and rules minimal. Note that the sample specifications presented in this thesis may be incomplete, however, they were found to be sufficient for the purpose of researching interaction between features.

Even though telecommunication features were used to demonstrate the definition and method presented in the thesis, the definition and method proposed in this thesis may be extended to other systems whose features are added incrementally as time goes on.

Furthermore Prolog may not be the best candidate for theorem proving, however, the simplicity of translating between our rules (an axiom is a basic rule) to Prolog axioms made it a good tool to show that our definition can be automated. A more intelligent search strategy to deal with the rules, equality and negation by failure should be the focus of future research.

8.1.1 Assessment

Relational Calculus has proven to be useful for specifying and capturing the behaviour of the systems. Even though writing the specification, using our model, has been shown to be a difficult and tedious task, the specifications produced represent the concise behaviour of systems and features under idealized behavior assumption. These specifications allowed us to detect undesired interaction which occured between specific features. Detecting interaction between features manually is a difficult and tedious task. Since our technique employs first order logic, our specifications can easily be mapped to Prolog. A relation can be represented by a binary predicate whose first argument is a list of inputs and second argument is a set of outputs.

8.2 Comparaison to Related Work

As mentioned in the introduction, there have been a number of approaches to detect feature interaction. In this section, we will compare our approach to LOTOS, state transition machine (STM), and two algorithms which were designed specifically for feature interaction detection for rule-based specifications.

The approach in [14] uses the LOTOS specification language based on the concept of labeled transition systems(LTSs). LTSs are a genereralization of finite state machines and provide a convenient way for expresssing step-by-step operational semantics of processes. Structurally, this approach captures the underlying concepts of structuring LOTOS specifications using three constraint types: local, end-to-end, and global constraints. Local constraints are used to enforce the appropriate sequence events at each users interaction point. End-to-End constraints are related to each connection and enforce the appropriate sequence of actions between between the interaction points for each connection.

Finally, global constraints are system-wide and involve information flow between all interaction points in the system. New features may be integrated to existing ones by classifying their roles as caller and called, and expressing their global constraint as a conjunction. This approach is analytically based on a reasoning mechanism, where the specifier can analyze features based on their local views (local constraints), where each view is an element of the global views of the system. Hence feature interaction are detected when there is a conflict between the feature views. This is achieved by deriving deriving a set of test cases, that when executed will cause a deadlock. Deriving appropriate cases for different types of interactions, and the limitation of the LOTOS testing theory still remain open issues.

Another approach presented in [24], employs STM to model feature behaviour, where each transition is triggered by a single input. Like our approach each feature is specified independently. Multiple STM's can be integrated 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. This method is limited due to the inability to specify data values, timing contraints and assumptions about the behaviour of the environment.

The Algorithm EXH for rule-based specification, proposed by Harada et al [23] consists of the following two phases:

- Enumerate all possible reachable states by the state enumeration then
- By Checking each enumerated state, detect a pair of rules which non-deterministically conflict with each other.

This technique can detect non-determinism interaction based on neccessary-sufficient condition.

The LOTOS, STM and the Algorithm EXH strategies were found to be very powerful. However the state explosion problem may be encountered since they all use reachability analysis to generate sequences to detect possible interactions. At the moment these strategies require manual inspection of the traces produced from the tools, hence better search strategies must be found.
The Algorithm Ω devised by Nakamura et al [4], based on the P-invariant and the Petri-Net Model overcomes the problem encountered in the Algorithm EXH . The Algorithm Ω requires no enumeration and can be outlined as follows:

- Transform the given service specification to the Petri-Net
- Obtain all pairs of rules which may cause the non-determinism
- Based on the rules, determine a set of states at which the non-determinsm may occur.
- Check if the determined states are reachable from the initial state (Using the P-invariant and Petri-Net model).

The result of the two experiments perform in [4] showed that Algorithm Ω attained a high quality of detection, drastic performance improvement over Algorithm EXH (up to 28000 time faster) and proved to be very scalable with respect to the number of users. However, it is possible to detect pairs of rules which do not cause interaction.

Our method has the same problems in terms of exponential growth of states as the LOTOS, STM, and the Algorithm EXH approaches. The valid number of sequences are exponentially proportional to the number of axioms and rules. Like LOTOS, deriving test cases to detect interaction between features has been found to be a tedious task. For these reasons, we proposed to automate this process using Prolog. This automation process did not simplify the problems, since our Prolog program attempts to reduce a sequence by exhausting all possible possibilities. Also since Prolog sequentially searches through the rules it is possible to enter into an infinite loop.

When we compared our approach to the Algorithm Ω approach, it was clear that Algorithm Ω is more efficient at detecting interactions as the number of users increases. The number of states is proportional to the number of users added to a service when Algorithm Ω is applied, while the relations grows exponentially using our approach.

No feature interaction detection model was proposed using the Trace Assertion representation [42]. However it is our belief that the Trace Assertion representation could detect feature interaction using similar concepts as our own model. The Trace Assertion method may offer insight on how to resolve the various problems encountered in the model propose in this thesis.

8.3 Future Work

Future research is needed to find a solution for the infinite looping problem caused by both the swap and reduction rules. A possible solution may exist using Prolog. However, another language or theorem prover may be more adequate to detect feature interaction automatically.

For the purpose of this thesis, POTS was assumed to be a feature, and that all other features were a superset of POTS. For this reason reference to the POTS tables were made frequently. In reality POTS is a system and the features are added incrementally to POTS. However, we found that using relational specifications the features would actually interact with the system itself. For example assume that OCS is defined by the tables 5.16, 5.17 , 5.18, 5.19 and 5.20. Furthermore, assume that the feature OCS is added to the POTS system. A feature interaction exists between POTS and OCS for the sequence

$$(\mathsf{Init}, u_1).(\mathsf{Init}, u_2).(\mathsf{OCSAct}, n_1).(\mathsf{OCSAdd}, n_1, n_2).(\mathsf{OffHook}, n_1).(\mathsf{Dial}, n_1, n_2)$$

This sequence using the POTS rules will generate a $\{(\mathsf{Ring}, n_1, n_2)\}$, however OCS will generate an $\{(\mathsf{ErrorTone}, n_1)\}$.

Therefore we proposed that a new operator "on top" denoted by $\triangle$ be defined as $X \triangle Y \triangleq (s, s') | \exists s : s \in dom(X) \ \wedge \ s \in dom(Y) \wedge$

$$((\exists s' : (s, s") \in X \ \wedge \ (s, s") \in Y) \vee (\exists s' : (s, s') \wedge \neg \exists s" : (s, s") \in X \ \wedge \ (s, s") \in Y))$$

which implies that if feature X interacts with system Y for some input sequence s, such that no common output can be agreed on, then the output corresponding to feature X will be considered. Work is currently in progress determining a simple definition for this operator and proving it's correctness.

Furthermore, the model presented in this thesis is an oversimplification and does not capture aspects such as real-time issues; latency, watchdog, resource competition and non-determinism due to concurrent and distributed races, but focus exclusively on functional aspects. Research is needed to expand our model to capture such aspects.
The following are also future research work:

- Devise a more intelligent search algorithm to deal with swap rule.
- Devise a strategy to deal with the state explosion problem.
- Try our approach to different systems.

Part IV

Appendices

Appendix A

Useful Functions

```
/* This file containes functions for writting relational specification.
 * Written by Serge Colle
 * Written on April 12th 1997
 * for thesis
 */
[library(lists)].
[library(basics)].


/*distinct make sure that all argument past are disctinct */
/*   distinct/3 */
/*   distinct/4 */

distinct(N1,N2,N3) :- N1 \== N2, N2 \== N3, N1 \== N3.
distinct(N1,N2,N3,N4) :- N1 \== N2, N2 \== N3, N3 \== N4,
                         N2 \== N4, N3 \== N1, N4 \== N1.

/* reducecomp reduces a complete input sequence by discarding a
portion of the list, this function is written in such as way to
be compatible with the specification table */

/* distinct/3   prefix is null (no condition).*/
/* distinct/4   this must be the sequence attach
to the discard sequence*/

reducecomp(X,Discard,X2) :-                append(Discard,Post,Rem),
                                           append(Pre,Rem,X),
Post \== [[]],
                                           append(Pre,Post,X2).
```

```
reducecomp(X,Discard, Prefix,X2) :-     append(Prefix,Discard,Temp1),
append(Temp1,Post,Rem),
                                        append(Temp2,Rem,X),
Post \== [[]],
append(Temp2,Prefix,Pre),
                                        append(Pre,Post,X2).

/* reduceinc reduces an incomplete input sequence by discarding
a portion of the list, this function is written in such as way
to be compatible with the specification table */

reduceinc(X,Discard,Post,X2) :-         append(X1,Post,X),
                                        append(Pre,Discard,X1),
                                        append(Pre,Post,X2).

/* swap swap to input element in the input sequence. This function
is written in such as way to be compatible with the specification
table */

/* distinct/3   prefix is null (no condition).
 distinct/4   this must be the sequence attach to the discard
      sequence*/


swap(X,Input1,Input2,X2)        :-      append(Input1,Input2,Temp1),
                                        append(Input2,Input1,Temp2),
                                        append(Temp1,Post,Rem),
                                        Post \== [[]],
                                        append(Pre,Rem,X),
                                        append(Pre,Temp2,Temp3),
                                        append(Temp3,Post,X2).

swap(X,Prefix,Input1,Input2,X2) :-      append(Input1,Input2,Temp1),
append(Prefix,Temp1, Temp2),
                                        append(Input2,Input1,Temp3),
                                        append(Temp2,Post,Rem),
                                        Post \== [[]],
                                        append(Temp4,Rem,X),
```

```
append(Temp4,Prefix,Temp5),
                                        append(Temp5,Temp3,Temp6),
                                        append(Temp6,Post,X2).
```

Appendix B

Test Program

```
/* Library needed to manipulate lists */
?- [library(basics)].
?- [library(lists)].
?- ['fct.ari'].          /* réduction and swap function */

/*
% base cases - cw
*/

cw([[init,N1],[cwAct],[offHook,N1],[dial,N1,N2],[offHook,N2],
    [offHook,N3],[dial,N3,N1],[flashHook,N1]],i
   [[conn,N1,N3],[d,N1,N2]]) :-
   N1\==N2,N1\==N3,N2\==N3.
/*
% reduction cases - cw
*/

cw(X,Y) :- Discard=[[twcAct,N1]],
           reducecomp(X,Discard,X2),
           cw(X2,Y).
/*
% base cases - twc
*/

twc([[init,N1],[twcAct,N1],[offHook,N1],[dial,N1,N2],[offHook,N2],
     [flashHook,N1]],
    [[dial,N1,N2],[dialTone,N1]]) :- N1\==N2.
%
```

```
% reduction cases - twc
%
twc(X,Y) :- Discard=[[cwAct,N1]],
            reducecomp(X,Discard,X2),
            twc(X2,Y).

twc(X,Y) :- Post=[[flashHook,N2]],
            Discard=[[offHook,N1]],
            N2\==N1,
            reduceinc(X,Discard,Post,X2),
            twc(X2,Y).

twc(X,Y) :- Post=[[flashHook,N3]],
            Discard=[[dial,N1,N2]],
            N3\==N1,
            reduceinc(X,Discard,Post,X2),
            twc(X2,Y).
/*
% feature interaction
*/
cw(X,Z), twc(X,W), \+((cw(X,Y), twc(X,Y))).
```

Appendix C

POTS Prolog Rules

```
/* This file containes the specification of the POTS.
 * Written by Serge Colle
 * Written on April 12th 1997
 * for thesis
 */

/*******************POTS INPUT SPACE**********************/

potsInput([init,_N1]).
potsInput([offHook,_N1]).
potsInput([onHook,_N1]).
potsInput([dial,_N1,_N2]).

lastIsPotsInput(X) :-  last(Element,X),
potsInput(Element).

/********POTS AXIOMS for successful connection************/
/*pots-ax-suc-init*/
pots([[init,_N1]],
[[]]).

/*pots-ax-suc-off*/
pots([[init,N1],[offHook,N1]],[[dialTone,N1]]).

/*pots-ax-suc-dial*/
pots([[init,N1],[offHook,N1],[init,N2],[dial,N1,N2]],
        [[ring,N1,N2]])
        :- N1 \== N2.
```

```
/*pots-ax-suc-con*/
pots([[init,N1],[offHook,N1],[init,N2],[dial,N1,N2],[offHook,N2]],
        [[conn,N1,N2]])
:- N1 \== N2.

/*pots-ax-suc-disc*/
pots([[init,N1],[offHook,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[onHook,N]],
[[disc,N1,N2]])
        :- N1 \== N2, (N == N1; N == N2).

/*pots-ax-suc-on*/
pots([[init,N1],[offHook,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[onHook,N],[onHook,Nprime]],
        [[]])
        :- N1 \== N2, (N == N1; N == N2), N \== Nprime.

/*********POTS AXIOMS for unsuccessful connection****************/

/*pots-ax-uns-invalid-no*/
pots([[init,N1],[offHook,N1],[dial,N1,N2]],
[[errorTone,N1]])
:- N1 \== N2.


/*pots-ax-uns-busy*/
pots([[init,N1],[init,N2],[offHook,N1],[offHook,N2],[dial,N1,N2]],
        [[busyTone,N1]])
        :- N1 \== N2.

/*pots-ax-uns-busy-self-call*/
pots([[init,N1],[offHook,N1],[dial,N1,N1]],
        [[busyTone,N1]]).

/*pots-ax-uns-busy-2nd-caller*/
pots([[init,N1],[init,N2],[init,N3],[offHook,N1],[offHook,N2],
[dial,N1,N3],[dial,N2,N3]],
        [[busyTone,N2]])
        :- distinct(N1,N2,N3).
```

```
/*pots-ax-uns-no-call*/
pots([[init,N1],[offHook,N1],[onHook,N1]],
        [[]]).

/*pots-ax-uns--double-dial*/
pots([[init,N1],[init,N2],[init,N3],[offHook,N1],[dial,N1,N2],
[dial,N1,N3]],
        [[]])
        :- N1 \== N2.

/*pots-ax-uns-system-space*/
pots(X,
        [[u]])
        :- \+lastIsPotsInput(X).


/********POTS REDUCTION RULES for complete cycle***********/

/* x.prefix.discard.x'*/
/* Prefix is more the last input of the prefix from spec */
/* ignore prefix if it is empty.*/
/* Please choose only one of the following twc-red-null rule */

/*twc-red-null*/
/*
  Can be use for any input which is not define by TWC, Very
  pratical for validating the specification,  for example if
  you enter the sequence and it will remove all input foreign
  to CW, except if the foreign input is the last input of the
  sequence
*/
/*
pots(X,Y)                  :-     reducecomp(X,ForeignInput,X2),
                                  \+potsInput(ForeignInput),
                                  pots(X2,Y).
*/

/**** IMPORTANT THIS MAY RUN OUT OF MEMORY ****/
```

```
/*This rule should be used when trying to find and interaction.
  This will make sure that input from other specification are
  inserted in the sequence. For this rule to work you will needs
  as many universalInput(X,Y) :- twcInput(X,Y) in the fi.ari file
*/
/*

pots(X,Y)              :-    universalInput(ForeignInput),
                             reducecomp(X,ForeignInput,X2),
                             \+pots(ForeignInput),
                             universalInput(ForeignInput),
                             pots(X2,Y).
*/

/*this will simulate the functionality of the previous rule.
  you must add a rule for each input found in other features
  which foreign to CW
*/
/* FOREIGN RULES START HERE */

pots(X,Y)              :-    ForeignInput = [[twcAct,_]],
                             reducecomp(X,ForeignInput,X2),
                             cw(X2,Y).

pots(X,Y)              :-    ForeignInput = [[twcDeact,_]],
                             reducecomp(X,ForeignInput,X2),
                             cw(X2,Y).

pots(X,Y)              :-    ForeignInput = [[cwAct,_]],
                             reducecomp(X,ForeignInput,X2),
                             cw(X2,Y).

pots(X,Y)              :-    ForeignInput = [[cwDeact,_]],
                             reducecomp(X,ForeignInput,X2),
                             cw(X2,Y).

pots(X,Y)              :-    ForeignInput = [[flashHook,_]],
                             reducecomp(X,ForeignInput,X2),
```

```
                                cw(X2,Y).

pots(X,Y)              :-   ForeignInput = [[ocsAct,_]],
                            reducecomp(X,ForeignInput,X2),
                            cw(X2,Y).

pots(X,Y)              :-   ForeignInput = [[ocsDeact,_]],
                            reducecomp(X,ForeignInput,X2),
                            cw(X2,Y).

pots(X,Y)              :-   ForeignInput = [[ocsAdd,_,_]],
                            reducecomp(X,ForeignInput,X2),
                            cw(X2,Y).

pots(X,Y)              :-   ForeignInput = [[ocsDel,_,_]],
                            reducecomp(X,ForeignInput,X2),
                            cw(X2,Y).

/* FOREIGN RULE STOP HERE */

/*pots-red-off-on*/
pots(X,Y) :-    Discard = [[offHook,N1],[onHook,N1]],
reducecomp(X,Discard,X2),
pots(X2,Y).

/*pots-red-off-dial-on*/
pots(X,Y) :-    Discard = [[offHook,N1],[dial,N1,_N2],[onHook,N1]],
reducecomp(X,Discard,X2),
pots(X2,Y).

/*pots-red-dial-dial-same-origin*/
pots(X,Y) :-    Discard = [[dial,_N1,_N3]],
                Prefix  = [[dial,_N1,_N2]],
reducecomp(X,Discard,Prefix,X2),
pots(X2,Y).


/******POTS REDUCTION RULES for uncomplete cycle**********/

/*pots-red-inc-off-init*/
```

```
pots(X,Y) :-    Post = [[init,_N2]],
Discard = [[offHook,_N1]],
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-off-off*/
pots(X,Y) :-    Post = [[offHook,N2]],
Discard = [[offHook,N1]],
N1 \== N2,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-off-dial*/
pots(X,Y) :-    Post = [[dial,N2,N3]],
                Discard = [[offHook,N1]],
                N1 \== N2, N1 \== N3,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-off-on*/
pots(X,Y) :-    Post = [[onHook,N2]],
                Discard = [[offHook,N1]],
                N1 \== N2,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-dial-init*/
pots(X,Y) :-    Post = [[init,N3]],
                Discard = [[dial,_N1,N2]],
                N2 \== N3,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-dial-off*/
pots(X,Y) :-    Post = [[offHook,N3]],
                Discard = [[dial,_N1,N2]],
                N2 \== N3,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).
```

```
/*pots-red-inc-dial-dial*/
pots(X,Y) :-    Post = [[dial,N3,N4]],
                Discard = [[dial,N1,N2]],
                N1 \== N3, N2 \== N4,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-dial-on*/
pots(X,Y) :-    Post = [[onHook,_N3]],
                Discard = [[dial,_N1,_N2]],
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-init-off*/
pots(X,Y) :-    Post = [[offHook,N2]],
                Discard = [[init,N1]],
                N1 \== N2,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-init-dial*/
pots(X,Y) :-    Post = [[dial,N2,_N3]],
                Discard = [[init,N1]],
                N1 \== N2,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*pots-red-inc-init-on*/
pots(X,Y) :-    Post = [[onHook,N2]],
                Discard = [[init,N1]],
                N1 \== N2,
reduceinc(X,Discard,Post,X2),
pots(X2,Y).

/*****************POTS SWAP RULES*******************/

/*pots-swap-off-off*/
pots(X,Y) :-    Input1 = [[offHook,_N1]],
```

```
                Input2 = [[offHook,_N2]],
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

/*pots-swap-off-dial*/
pots(X,Y) :-    Input1 = [[offHook,N1]],
                Input2 = [[dial,N2,N3]],
                N1 \== N2, N1 \== N3,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

pots(X,Y) :-    Input1 = [[dial,N2,N3]],
                Input2 = [[offHook,N1]],
                N1 \== N2, N1 \== N3,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

/*pots-swap-off-on*/
pots(X,Y) :-    Input1 = [[offHook,N1]],
                Input2 = [[onHook,N2]],
                N1 \== N2,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

pots(X,Y) :-    Input1 = [[onHook,N1]],
                Input2 = [[offHook,N2]],
                N1 \== N2,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

/*pots-swap-on-dial*/
pots(X,Y) :-    Input1 = [[onHook,N1]],
                Input2 = [[dial,N2,N3]],
                N1 \== N2, N1 \== N3,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

pots(X,Y) :-    Input1 = [[dial,N1,N2]],
```

```
                Input2 = [[onHook,N3]],
                N1 \== N2, N1 \== N3,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

/*pots-swap-init-off*/
pots(X,Y) :-    Input1 = [[init,N1]],
                Input2 = [[offHook,N2]],
                N1 \== N2,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

pots(X,Y) :-    Input1 = [[offHook,N1]],
                Input2 = [[init,N2]],
                N1 \== N2,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

/*pots-swap-init-dial*/
pots(X,Y) :-    Input1 = [[init,N1]],
                Input2 = [[dial,N2,N3]],
                N1 \== N2, N1 \== N3,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

pots(X,Y) :-    Input1 = [[dial,N1,N2]],
                Input2 = [[dial,N3]],
                N1 \== N2, N1 \== N3,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

/*pots-swap-init-on*/
pots(X,Y) :-    Input1 = [[init,N1]],
                Input2 = [[onHook,N2]],
                N1 \== N2,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

pots(X,Y) :-    Input1 = [[onHook,N1]],
```

```
                Input2 = [[init,N2]],
                N1 \== N2,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).

/*pots-swap-init-init*/
pots(X,Y) :-    Input1 = [[init,N1]],
                Input2 = [[init,N2]],
                N1 \== N2,
                swap(X,Input1,Input2,X2),
                pots(X2,Y).
```

Appendix D

TWC Prolog Rules

```
/* This file containes the specification for the TWC feature.
 * Written by Serge Colle
 * Written on April 12th 1997
 * for thesis
 */

/*********TWC AXIOMS for successful connection***********/
/*twc-ax-suc-activate*/
twc([[init,N1],[twcAct,N1]],
        [[]]).

/*twc-ax-suc-flash*/
twc([[init,N1],[twcAct,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[flashHook,N1]],
        [[dialTone,N1],[hold,N1,N2]])
        :- N1 \== N2.

/*twc-ax-suc-flash-dial*/
twc([[init,N1],[twcAct,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[flashHook,N1],[init,N3], [dial,N1,N3]],
[[ring,N1,N3]])
        :- distinct(N1,N2,N3).

/*twc-ax-suc-flash-conn*/
twc([[init,N1],[twcAct,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[flashHook,N1],[init,N3],[dial,N1,N3],[offHook,N3]],
        [[conn,N1,N3]]).
:- distinct(N1,N2,N3).
```

```
/*twc-ax-suc-flash-conn-3*/
twc([[init,N1],[twcAct,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[flashHook,N1],[init,N3],[dial,N1,N3],[offHook,N3],
[flashHook,N1]],
        [[conn,N2,N3]])
:- distinct(N1,N2,N3).

/*twc-ax-suc-flash-disc-z*/
twc([[init,N1],[twcAct,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[flashHook,N1],[init,N3],[dial,N1,N3],[offHook,N3],
[flashHook,N1],[onHook,NZ]],
        [[disc,N1,NZ],[disc,NW,NZ]])
:- distinct(N1,N2,N3),
          (NZ == N2; NZ == N3), (NW == N2; NW == N3), NZ \== NW.

/*twc-ax-suc-flash-disc-w*/
twc([[init,N1],[twcAct,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[flashHook,N1],[init,N3],[dial,N1,N3],[offHook,N3],
[flashHook,N1],[onHook,NZ],[onHook,NW]],
        [[disc,N1,NW]])
:- distinct(N1,N2,N3),
   (NZ == N2; NZ == N3), (NW == N2; NW == N3), NZ \== NW.

/*twc-ax-suc-flash-on*/
twc([[init,N1],[twcAct,N1],[init,N2],[dial,N1,N2],[offHook,N2],
[flashHook,N1],[init,N3],[dial,N1,N3],[offHook,N3],
[flashHook,N1],[onHook,NZ],[onHook,NW],[onHook,N1]],
        [[]])
:- distinct(N1,N2,N3),
   (NZ == N2; NZ == N3), (NW == N2; NW == N3), NZ \== NW.


/************TWC AXIOMS for unsuccessful connection*************/

/*twc-ax-uns-flash-on*/
twc([[init,N1],[twcAct,n1],[offHook,N1],[init,N2],[dial,N1,N2],
[offHook,N2],[flashHook,N1],[init,N3],[dial,N1,N3],
[offHook,N3],[flashHook,N1], [onHook,N1],[flashHook,N1]],
        [[disc,N1,N3],[disc,N2,N3]])
        :- distinct(N1,N2,N3).
```

```
/*twc-ax-uns-on-n1*/
twc([[init,N1],[twcAct,n1],[offHook,N1],[init,N2],[dial,N1,N2],
[offHook,N2],[flashHook,N1],[init,N3],[dial,N1,N3],
[offHook,N3],[flashHook,N1],[onHook,N1]],
        [[disc,N1,N2],[disc,N1,N3],[disc,N2,N3]])
        :- distinct(N1,N2,N3).

/*twc-ax-uns-busy-2nd-caller*/
twc([[init,N1],[twcAct,n1],[init,N2],[offHook,N2],[init,N3],
[dial,N1,N3],[dial,N2,N3]],
        [[busyTone,N2]])
        :- distinct(N1,N2,N3).

/*twc-ax-uns-flash-busy*/
twc([[init,N1],[twcAct,n1],[offHook,N1],[init,N2],[dial,N1,N2],
[offHook,N2],[flashHook,N1],[init,N3],[offHook,N3],
[dial,N1,N3]],
        [[busyTone,N1]])
        :- distinct(N1, N2, N3).

/*twc-ax-uns-TWCDeact*/
/*
twc(X,[[]]) :- Post = [[twcDeact,N1]],
       append(X1,Post,X),
       member([init,N1],X1).
*/
/*twc-ax-uns-system-space*/
twc(X,
        [[u]])
        :- \+lastIsTwcInput(X).

/*********TWC REDUCTION RULES for complete cycle***********/

/* x.prefix.discard.x'*/
/* Prefix is more the last input of the prefix from spec */
/* ignore prefix if it is empty.*/

/*twc-red-null*/
twc(X,Y) :-     lastIsTwcInput(X),
```

```
                append(Head,Tail,X),
                checkTwcSeq(Head,X1),
                append(X1,Tail,X2),
                twc(X2,Y).


/*twc-red-flash-off-on*/
twc(X,Y) :-     Discard = [[init,N1],[twcAct,N1],[offHook,N1],
[init,N2],[onHook,N2]],
                N1 \== N2,
                reducecomp(X,Discard,X2),
twc(X2,Y).

/*twc-red-off-flash-on*/
twc(X,Y) :-     Discard = [[dial,N1,N2],[offHook,N2],
[flashHook,N1],[onHook,N2]],
                N1 \== N2,
                reducecomp(X,Discard,X2),
twc(X2,Y).

/*twc-red-off-flash*/
twc(X,Y) :-     Discard = [[flashHook,N1]],
Prefix = [[offHook,N1]],
                reducecomp(X,Discard,Prefix,X2),
                twc(X2,Y).

/*twc-red-act-deact*/
twc(X,Y) :-     Discard = [[twcAct,N1],[twcDeact,N1]],
                reducecomp(X,Discard,X2),
                twc(X2,Y).

/*twc-red-flash-flash*/
twc(X,Y) :-     Discard = [[flashHook,N1],[flashHook,N1]],
                reducecomp(X,Discard,X2),
                twc(X2,Y).

/*********TWC REDUCTION RULES for uncomplete cycle**********/


/*twx-red-inc-act-init*/
```

```
twc(X,Y) :-     Post = [[init,N2]],
                Discard = [[twcAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-act-off*/
twc(X,Y) :-     Post = [[offHook,_N2]],
                Discard = [[twcAct,_N1]],
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-act-dial*/
twc(X,Y) :-     Post = [[dial,_N2,_N3]],
                Discard = [[twcAct,_N1]],
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-act-on*/
twc(X,Y) :-     Post = [[onHook,_N2]],
                Discard = [[twcAct,_N1]],
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-act-flash*/
twc(X,Y) :-     Post = [[flashHook,N2]],
                Discard = [[twcAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-act-act*/
twc(X,Y) :-     Post = [[twcAct,N2]],
Discard = [[twcAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2, Y).

/*twc-red-inc-act-deact*/
twc(X,Y) :-     Post = [[twcDeact,N2]],
```

```
Discard = [[twcAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-init-flash*/
twc(X,Y) :-     Post = [[flashHook,N2]],
Discard = [[init,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-init-act*/
twc(X,Y) :-     Post = [[twcAct,N2]],
Discard = [[init,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-init-deact*/
twc(X,Y) :-     Post = [[twcDeact,N2]],
Discard = [[init,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-off-flash*/
twc(X,Y) :-     Post = [[flashHook,N2]],
Discard = [[offHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-off-act*/
twc(X,Y) :-     Post = [[twcAct,_N2]],
Discard = [[offHook,_N1]],
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-off-deact*/
```

```
twc(X,Y) :-     Post = [[twcDeact,_N2]],
Discard = [[offHook,_N1]],
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-dial-flash*/
twc(X,Y) :-     Post = [[flashHook,N3]],
Discard = [[dial,N1,_N2]],
                N1 \== N3,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-dial-act*/
twc(X,Y) :-     Post = [[twcAct,N3]],
Discard = [[dial,N1,_N2]],
                N1 \== N3,
                reduceinc(X,Discard,Post,X2),
twc(X2, Y).

/*twc-red-inc-dial-deact*/
twc(X,Y) :-     Post = [[twcDeact,N3]],
Discard = [[dial,N1,_N2]],
                N1 \== N3,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-flash-init*/
twc(X,Y) :-     Post = [[init,N2]],
Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).

/*twc-red-inc-flash-off*/
twc(X,Y) :-     Post = [[offHook,N2]],
Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
twc(X2,Y).
```

```
/*twc-red-inc-flash-dial*/
twc(X,Y) :-     Post = [[dial,N2,_N3]],
Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                twc(X2,Y).

/*twc-red-inc-flash-flash*/
twc(X,Y) :-     Post = [[flashHook,N2]],
Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                twc(X2,Y).

/*twc-red-inc-flash-act*/
twc(X,Y) :-     Post = [[twcAct,N2]],
Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                twc(X2,Y).

/*twc-red-inc-flash-deact*/
twc(X,Y) :-     Post = [[twcDeact,N2]],
Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                twc(X2,Y).

twc(X,Y) :- pots(X,Y).

/*****************TWC Input Space*****************/

twcInput([twcAct,_N1]).
twcInput([twcDeact,_N1]).
twcInput([flashHook,_N1]).
twcInput(X,Y) :- potsInput(X,Y).

lastIsTwcInput(X) :-  last(Element,X),
twcInput(Element).
```

```
checkTwcSeq(X,X2) :-   last(Element,X),
                       \+twcInput(Element),
                       delete(X,Element,X2).
```

Appendix E

CW Prolog Rules

```
/* This file containes the specification for the CW feature.
 * Written by Serge Colle
 * Written on April 12th 1997
 * for thesis
 */


/***********CW AXIOMS for successful connection*************/

/*cw-ax-suc-init*/
cw([[init,N1],[cwAct,N1]],
        [[]]).

/*cw-ax-suc-3rd-party-dial*/
cw([[init,N1],[cwAct,N1],[offHook,N1],[init,N2],[dial,N1,N2],
[offHook,N2],[init,N3],[offHook,N3],[dial,N3,N1]],
        [[ringTone,N3],[waitTone,N1]])
        :- distinct(N1,N2,N3).

/*cw-ax-suc-talk-to-3rd*/
cw([[init,N1],[cwAct,N1],[offHook,N1],[init,N2],[dial,N1,N2],
[offHook,N2],[init,N3],[offHook,N3],[dial,N3,N1],
[flashHook,N1]],
        [[hold,N1,N2],[conn,N1,N3]])
:- distinct(N1,N2,N3).

/*cw-ax-suc-finish-3rd*/
cw([[init,N1],[cwAct,N1],[offHook,N1],[init,N2],[dial,N1,N2],
[offHook,N2],[init,N3],[offHook,N3],[dial,N3,N1],
```

```
[flashHook,N1],[flashHook,N1]],
        [[conn,N1,N2],[hold,N1,N3]])
:- distinct(N1,N2,N3).

/*cw-ax-suc-3rd-off*/
cw([[init,N1],[cwAct,N1],[offHook,N1],[init,N2],[dial,N1,N2],
[offHook,N2],[init,N3],[offHook,N3],[dial,N3,N1],
[flashHook,N1],[flashHook,N1],[onHook,N3]],
        [[]])
:- distinct(N1,N2,N3).

/*************CW AXIOMS for unsuccessful connection***********/
/*cw-ax-uns-busy*/
cw([[init,N1],[cwAct,N1],[cwAct,n1],[offHook,N1],[dial,N1,N2],
[init,N2],[offHook,N2],[init,N3],[offHook,N3],
[dial,N3,N1],[flashHook,N1],[offHook,N4],[dial,N4,N1]],
        [[busyTone,N4]])
        :- distinct(N1,N2,N3,N4).


/*cw-ax-uns-deact*/
/*Eventhought this rule is logically true and precise, because
  of Prolog search method, it will cause the program to enter
  into an infinite loop

cw(X,[[]]) :- Post = [[cwDeact,N1]],
              append(X1,Post,X),
              member([init,N1],X1).
*/

/*cw-ax-uns-system-space*/
cw(X,
        [[u]])
        :- \+lastIsCwInput(X).

/*************CW REDUCTION RULES for complete cycle*************/

/* x.prefix.discard.x'*/
/* Prefix is more the last input of the prefix from spec */
/* ignore prefix if it is empty.*/
```

```
/* Please choose only one of the following cw-red-null rule */

/*cw-red-null*/
/*Can be use for any input which is not define by TWC, Very
  pratical for validating the specification,  for example if
  you enter the sequence and it will remove all input foreign
  to CW, except if the foreign input is the last input of the
  sequence
*/
/*
cw(X,Y)                 :-      reducecomp(X.ForeignInput.X2),
                                \+cwInput(ForeignInput),
                                twc(X2,Y).
*/

/*cw-red-null*/
       /* This rule should be used when trying to find and
          interaction. This will make sure that input from
          other specification are inserted in the sequence.
          For this rule to work you will needs as many
          universalInput(X) :- cwINput(X) in the fi.ari file */

/**** IMPORTANT THIS MAY RUN OUT OF MEMORY ****/

/*
cw(X,Y)                 :-      universalInput(ForeignInput),
                                reducecomp(X,ForeignInput,X2),
                                \+cwInput(ForeignInput),
                                universalInput(ForeignInput),
                                cw(X2,Y).
*/

/*this will simulate the functionality of the previous rule.
  you must add a rule for each input found in other features
  which foreign to CW
*/
/* FOREIGN RULES START HERE */

cw(X,Y)                 :-      ForeignInput = [[twcAct,N1]],
```

```
                                reducecomp(X,ForeignInput,X2),
                                cw(X2,Y).

cw(X,Y)                 :-      ForeignInput = [[twcDeact,N1]],
                                reducecomp(X,ForeignInput,X2),
                                cw(X2,Y).

cw(X,Y)                 :-      ForeignInput = [[ocsAct,N1]],
                                reducecomp(X,ForeignInput,X2),
                                cw(X2,Y).

cw(X,Y)                 :-      ForeignInput = [[ocsDeact,N1]],
                                reducecomp(X,ForeignInput,X2),
                                cw(X2,Y).

cw(X,Y)                 :-      ForeignInput = [[ocsAdd,N1,N2]],
                                reducecomp(X,ForeignInput,X2),
                                cw(X2,Y).

cw(X,Y)                 :-      ForeignInput = [[ocsDel,N1,N2]],
                                reducecomp(X,ForeignInput,X2),
                                cw(X2,Y).

/* FOREIGN RULE STOP HERE */

/*cw-red-dial-flash-off*/
cw(X,Y) :-      Discard = [[dial,N1,N2],[flashHook,N2],
[offHook,N2],[onHook,N2]],
                N1 \== N2,
                reducecomp(X,Discard,X2),
                cw(X2,Y).

/*cw-red-dial-flash-off-flash-on*/
cw(X,Y) :-      Discard = [[dial,N1,N2],[flashHook,N2],
[offHook,N2],[flashHook,N1],[onHook,N2]],
                N1 \== N2,
                reducecomp(X,Discard,X2),
                cw(X2,Y).

/*cw-red-on-flash*/
```

```
cw(X,Y) :-      Discard = [[offHook,N1]],
Prefix = [[flashHook,N1]],
                reducecomp(X,Discard,Prefix,X2),
                cw(X2,Y).

/*cw-red-flash-flash*/
cw(X,Y) :-      Discard = [[flashHook,N1],[flashHook,N1]],
                reducecomp(X,Discard,X2),
                cw(X2,Y).

/*cw-red-act-deact*/
cw(X,Y) :-      Discard = [[cwAct,N1],[cwDeact,N1]],
                reducecomp(X,Discard,X2),
                cw(X2,Y).



/********CW REDUCTION RULES for incomplete cycle*************/
/*cw-red-inc-act-init*/
cw(X,Y) :-      Post = [[init,N2]],
                Discard = [[cwAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-act-off*/
cw(X,Y) :-      Post = [[offHook,_N2]],
                Discard = [[cwAct,_N1]],
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-act-dial*/
cw(X,Y) :-      Post = [[dial,_N2,_N3]],
                Discard = [[cwAct,_N1]],
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-act-on*/
cw(X,Y) :-      Post = [[onHook,_N2]],
                Discard = [[cwAct,_N1]],
```

```
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-act-flash*/
cw(X,Y) :-      Post = [[flashHook,N2]],
                Discard = [[cwAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-act-act*/
cw(X,Y) :-      Post = [[cwAct,N2]],
                Discard = [[cwAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2, Y).

/*cw-red-inc-act-deact*/
cw(X,Y) :-      Post = [[cwDeact,N2]],
                Discard = [[cwAct,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-init-flash*/
cw(X,Y) :-      Post = [[flashHook,N2]],
                Discard = [[init,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-init-act*/
cw(X,Y) :-      Post = [[cwAct,N2]],
                Discard = [[init,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-init-deact*/
cw(X,Y) :-      Post = [[cwDeact,N2]],
```

```
                Discard = [[init,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-off-flash*/
cw(X,Y) :-      Post = [[flashHook,N2]],
                Discard = [[offHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).


/*cw-red-inc-off-act*/
cw(X,Y) :-      Post = [[cwAct,N2]],
                Discard = [[offHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-off-deact*/
cw(X,Y) :-      Post = [[cwDeact,N2]],
                Discard = [[offHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-dial-flash*/
cw(X,Y) :-      Post = [[flashHook,N3]],
                Discard = [[dial,N1, N2]],
                distinct(N1,N2,N3),
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-dial-act*/
cw(X,Y) :-      Post = [[cwAct,N3]],
                Discard = [[dial,N1,_N2]],
                N1 \== N3,
                reduceinc(X,Discard,Post,X2),
                cw(X2, Y).
```

```
/*cw-red-inc-dial-deact*/
cw(X,Y) :-      Post = [[cwDeact,N3]],
                Discard = [[dial,N1,_N2]],
                N1 \== N3,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-flash-init*/
cw(X,Y) :-      Post = [[init,N2]],
                Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

/*cw-red-inc-flash-off*/
cw(X,Y) :-      Post = [[offHook,N2]],
                Discard = [[flashHook,N1]],
                N1 \== N2,
                reduceinc(X,Discard,Post,X2),
                cw(X2,Y).

cw(X,Y) :- pots(X,Y).

cwInput([cwAct,_N1]).
cwInput([cwDeact,_N1]).
cwInput([flashHook,_N1]).
cwInput(X,Y) :- potsInput(X,Y).

lastIsCwInput(X) :-  last(Element,X), cwInput(Element).

checkCwSeq(X,X2) :-    last(Element,X),
                       \+cwInput(Element),
                       delete(X,Element,X2).
```

Appendix F

Control Module

```
/*Main file */
/*This file should contain all the features which are found
  in the system
*/


/* Library needed to manipulate lists */
?- [library(basics)].
?- [library(lists)].
?- ['fct.ari']. /* reduction and swap function */

/***************MODIFYABLE SECTION************************/

/*Please add a rule for every feature  ei.
  ['<feature>.ari'].
*/
/*Note for readibility every feature should be specified
in seperate files */

?- ['pots.ari'].  /*Plain Old Telephone System specification*/
?- ['cw.ari'].    /*Call Waiting specification*/
?- ['twc.ari'].   /*Three Way Calling specification*/

/*This section should is only used in Detecting feature
  interaction between features
*/
/*Please add a rule for every feature such that the input
  are known globally
*/
```

```
/* universalInput(X)  :-  <feature>Input(X).      */

universalInput(X) :- potsInput(X). /* POTS inputs */
universalInput(X) :- cwInput(X). /* CW inputs */
universalInput(X) :- twcInput(X). /* TWC inputs */

/* feature interaction queries */
?- cw(X,Z),twc(X,W),\+((cw(X,Y),twc(X,Y))),Z\==[[u]],W\==[[u]].
```

Bibliography

[1] Bergstra, J., Bouma W.: Models for Feature Description and Interactions. *Feature Interactions in Telecommuniactions and Distributed Systems IV* eds, IOS Press, 1997 31-45.

[2] Khousmi, A.:Detection and Resolution of Interaction between Services of a Telephone Networks. *Feature Interactions in Telecommuniactions and Distributed Systems IV* eds, IOS Press, 1997 61-78.

[3] Frappier M., Mili A., Desharnais J.,:Defining and Detecting Feature Interactions. *Feature Interactions in Telecommuniactions and Distributed Systems IV* eds, IOS Press, 1997 123-137.

[4] Nakamura, M., Kakuda, Y.,Kikuno, T.:Petri-Net Based Detection Method for Nondeterministic Feature Interactions and its Experimental Evaluation. *Feature Interactions in Telecommuniactions and Distributed Systems IV* eds, IOS Press, 1997 138-152

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

[6] 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.

[7] 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.

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

[9] Thomas, M.:Modelling and Analysing User Views of Telecommunications Services. *Feature INteractions in Telecommuniactions and Distributed Systems IV* eds, IOS Press, 1997 168-182.

[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, *Feature Interactions in Telecommunications Software Systems*, , IOS Press, 1994, 1-23.

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

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

[14] Faci, M., Logrippo, L.: Specifying Features and Analysing their Interactions in a LOTOS Environment, *Feature Interactions in Telecommunications Software Systems*, IOS Press, 1994, 136-151.

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

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

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

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

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

[20] Kimbler, K.H. Velthuijsen: Feature Interaction Benchmark, KPN Research, PO Box 421, 2260 AK Leidschendam, The Netherlands, 1995, 1–13.

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

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

[23] Harada, Y., Hirakawa, T., Takenaka, T., Terashima, N. : A conflict detection support method for telecommunication service description, *IEICE Trans Commun.*, Vol E75-B, No. 10, Oct, 1992.

[24] Au, P., Atlee, J.:Evalutation of State-Based Model of Feature Interactions. *Feature INteractions in Telecommuniactions and Distributed Systems IV* eds, IOS Press, 1997 138-152

[25] Boehm,B.W.: Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981.

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

[27] 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.

[28] 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.

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

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

[31] Boudriga, N., Mili A., Mili F., Zalila R.: Specifying and Verifying Data Types: A Relational Approach. *Programming Languages, an International Journal.* (Pergammon Press). To appear.

[32] Boudriga, N., Mili A., Mili F., Zalila R.: A Relational Model for the Specification of Data Type. *Computer language*, Pergamon Press, 1992, pp.101-131.

[33] Dwyer, B., Mili A. Issues on Program Specification: Towards a discipline of Structure specifying. Proceedings, ACSC-10, February 1987.

[34] Manna, Z. : Mathematical Theory of Computation. New York: McGraw Hill, 1974.

[35] Meyer, B.: On Formalism in Specifications. *IEEE Software.* Vol 1, pp 6-26. January 1985.

[36] Mili, A., Boudriga N., Mili F.: Towards Structured Specifying. Chichester, UK: Ellis Horwood, Ltd. 1989.

[37] Morgan, C.: Programming from Specifications. London, UK: Prentice Hall International, 1990.

[38] Waite, W.M., Goos G.: Compiler Construction. New York, NY: Springer Verlag, 1984.

[39] Bowen T.F., Dworak F.S. , Chow C.H., Griffeth N., Herman G.E., Lin Y-J.: The Feature Interaction Problem in Telecommunications Systems, *Proc. 7th Int. Conf. on Software Engineering for Telecommunication Switching Systems*, July 1989, 59–62.

[40] Bouma, W., Velthuijsen H.: Feature Interactions in Telecommunications Software Systems. IOS Press, 1994.

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

[42] Wang Y., Parnas T.L.: *Simulating the Behavior Software Modules by Trace Rewriting.* EEEI Transaction on Software Engineering, Vol. 20, No. 10, October 94.