Chapter 1

Introduction

1.1 Importance of Software Testing

Software development involves a series of production activities in which there are many opportunities to make mistakes [1] [7, 8]. Such mistakes may begin to occur at an initial stage of the process where user/systems requirements are poorly specified, and in subsequent design, development, and implementation stages where design and programming faults are introduced. Faulty software may cause inconvenience, annoyance, misinformation, loss of information, and loss of money. For safety critical systems used in medical devices, aviation, nuclear power generation, and weapon systems, a software fault may result in personal injury or even death. Potential catastrophes of these faulty systems have been widely discussed in the literature [41, 56, 57, 61]. The importance of software quality is beyond doubt and, hence, should be addressed seriously. This explains why software development should always be accompanied by quality assurance

[1] We follow the IEEE standard terminology, that is, a *failure* is an observed malfunction of a software system, which is caused by a *fault* in that system, which in turn is caused by a human *mistake*.

activities [55, 69].

To address the software quality issue, many validation and verification techniques have been proposed, including: (a) program proving [1, 2, 47, 86], (b) code review methods such as walkthroughs, inspections, and technical reviews [17, 18, 26, 33, 34, 63, 64, 68, 71, 76], and (c) software testing (or simply testing [2]) [5, 35, 43, 55]. On one hand, program proving is generally agreed to be difficult to scale up for practical use in large software systems [27]. On the other hand, although code review methods are useful in detecting software faults, these methods are mainly used only in large organizations with well-established software development standards [43]. Thus, many organizations rely primarily on testing to reveal failures [22, 31, 48] and to estimate the reliability of their software systems [3, 29, 37, 52, 53].

Indeed testing has long been recognized as the most practical quality assurance technique [3, 35]. It is the primary means to detect failures and to prevent software faults from propagating through to the final production system, where the cost of removal is far greater [52, 59, 70]. It is reported in the literature [8, 41] that the cost-escalation factors range from 5:1 to 100:1, depending on the size and type of software systems. Unlike other quality assurance techniques such as code review methods, testing evaluates how a software system actually performs its functions in its intended or simulated environment [36]. Testing is argued to be the unavoidable part of any responsible effort for software development [45, 46, 65, 79, 83], and has long been a focus in software engineering research [35].

[2] Numerous types of software testing exist; each of which is done for different purposes. Examples are functional testing, compatibility testing, performance testing, scalability testing, and stress testing. In this thesis, we shall mainly focus on *functional testing*, whose purpose is to determine how well a software system meets its functionality requirements.

1.2 Two Approaches to Test Suite Generation

Ideally, the entire *input domain* (the set of all possible inputs) of a program should be tested so that all failures can be revealed. This "exhaustive" testing approach, however, is practically infeasible unless the program is trivial or the tester is given unlimited testing resources (such as people and time). Often, only a subset of the input domain, called a *test suite*, can be selected for testing. [3] Obviously, no matter how the test suite is generated, the comprehensiveness of the testing will be affected. This problem immediately gives raise to the following key question:

> "*How to generate a test suite* (*or test cases*) *so that the comprehensiveness of the testing will be jeopardized as less as possible?*"

The above question has inspired a vast body of studies on various test case generation methods. Generally speaking, all these methods can be classified into one of the two broad categories: white-box and black-box approaches. They will be discussed in Sections 1.2.1 and 1.2.2 below. Neither of these approaches is sufficient; they complement each other [12, 19, 84].

1.2.1 White-Box Testing

White-box testing, also called *implementation-based testing*, involves generating test cases based on the information derived from the source code of the software system under test. White-box testing typically requires the coverage of certain "aspects" of the software structures. The rationale is that, unless a given aspect is executed, there is no way to reveal the faults associated with it. Three common examples of white-box testing are listed below:

[3] In other words, a *test suite* is the set of test cases actually used for testing.

(a) Control Flow Testing

It is based on the decision structure of the software system [32, 55, 58, 67, 74, 75, 80, 85]. Several control flow testing strategies exist, among which statement coverage, branch coverage, and path coverage are the most popular ones [55, 67]. Such testing strategies require that every statement, branch, and path in the software system, respectively, be executed at least once. Among these three strategies, statement coverage is the weakest form of testing. This is because, even if a test covers all the program statements, many faults may still remain undetected. Branch coverage is somewhat stronger, because it requires that every outcome of every decision be exercised, but it also may leave undetected faults. Path coverage is the strongest form of control flow testing. However, it is well known that path coverage is usually very difficult or impossible to achieve because a program may contain a huge or even infinite number of paths (for example, a program with loops) [55, 67].

(b) Data Flow Testing

Its main idea is to generate test cases that follow the pattern of definitions and uses of the program variables [6, 50, 67, 85]. A *definition* (or *def*) of a variable occurs when it is assigned a value. The *use* of a variable x may be in one of the two forms. When x occurs in a predicate such as a conditional expression of a loop or selection statement, x is said to have a *predicate use* (or *p-use*). Alternatively, when x is used in a computation such as the evaluation of an expression, x is said to have a *computational use* (or *c-use*). P-uses directly affect the flow of control through the program and thus may indirectly affect the computations performed. On the other hand, c-uses directly affect computations that may indirectly affect the flow of control. Well known criteria in data flow testing are all-defs, all-uses, all-p-uses, all-p-uses/some-c-uses, and all-c-uses/some-p-uses [32].

(c) Domain Testing

Its essence is to divide the input domain of a program $\mathbb{P}$ into some regions (or *domains*) so that all the test cases chosen from one region will cause the same path in $\mathbb{P}$ to be executed [23, 77]. The testing strategy involves examining every border of each region in detail and selecting test cases near the border. The strategy is fault-based, aiming at detecting domain errors (that is, incorrect borders).

A main advantage of domain testing is that it gives explicit guidance in the selection of data to test the predicates in conditional statements. There are, however, some restrictions imposed on the method. For example, the original domain testing method developed by White and Cohan [77] requires that the input domain is continuous and the predicate interpretations are simple linear inequalities. Later, Clarke et al. [23] have extended the original method by relaxing some of the restrictions so that discrete domains can be handled.

In general, white-box testing requires massive tasks such as determining branch coverage for control flow testing and variable usage for data flow testing [24]. Obviously, these tasks are tedious and prone to human mistakes when performed by hand and, hence, they are best left to automated tools. An example of these tools is the coverage analyzer used to measure the percentage of program statements, branches, or paths that has been executed by one single or a series of test runs. Because there is a shortage of experienced users of these tools, most commercial organizations are reluctant to adopt the white-box approach for testing their software systems [24]. Another drawback of white-box testing is that it cannot be applied when the source codes of the software systems under test are not available.

1.2.2 Black-Box Testing

Black-box testing, also called *specification-based testing*, involves generating test cases based on the information derived from the specification document (or simply the specification), without requiring the knowledge of the internal structure of the software. In other words, it treats the software system as a "black box" whose internal structure is unknown.

In software development, user and systems requirements are established before implementation and, hence, the specification should exist prior to program coding. In this respect, the black-box approach is particularly useful because test cases can be generated before coding has been completed. This facilitates software development phases to be performed in parallel, thus allowing time for preparing more thorough test plans and yet shortening the duration of the whole development process. Another merit of the black-box approach, when comparing to the white-box approach is that, the former (but not the latter) can be applied to test off-the-shelf software packages where the source codes are normally not available from the vendors. Such reasons make black-box testing very popular in the commercial sector.

Let us take a close examination of specifications — the sources for generating test cases in black-box testing. These documents often exist in a spectrum of forms as depicted in Figure 1.1. At the left extreme is the completely *informal* specification primarily written in natural languages (possibly supplemented by graphical languages). On the other hand, the right extreme of the spectrum corresponds to the completely *formal* specification written in a mathematical notation such as Z [81] and Boolean predicates [51, 73]. A specification, in general, may be in a format lying somewhere between these two extremes.

Formal specifications, because of the mathematical basis, are more precise than informal specifications. Formal specifications can be verified, for instance, for consis-

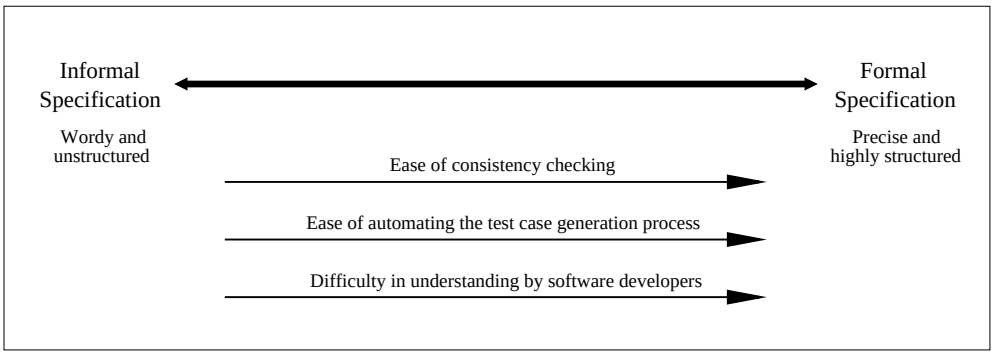


Figure 1.1: Spectrum of Different Forms of Specifications

tency. Furthermore, the rigorous nature of formal specifications eases the automatic generation of test cases [38]. Despite these advantages, formal specifications and their associated test case generation methods [9, 13, 25] are not as popular as they should be, mainly because most software developers are not familiar with the mathematical concepts involved and find the techniques difficult to understand and use. Because of this reason, informal specifications are more popular in the commercial software industry [60, 82]. It is also the reason why this thesis focuses mainly on test case generation from informal specifications. The following are some popular black-box test case generation methods that are applicable to informal specifications:

(a) Random Testing

Random testing, also called *statistical testing*, is unarguably the simplest black-box testing method, in which test cases are selected at random from the input domain [28, 42, 66, 78]. Selection of test cases is usually based on a uniform distribution or an operational profile, the latter being an approximation to the estimated probability distribution of usage of the inputs [53, 54]. To prevent the possible selection bias of the tester, different test cases should be selected statistically independent of one another. Two advantages of random testing are its simplicity in concept and relative ease of automation by generating inputs using pseudo-random numbers. It requires

minimal knowledge of the specification; only the knowledge of the input domain is necessary. Recently, Kuo and her coworkers have extended the original random testing method, resulting in their adaptive random testing method [49].

(b) Cause-Effect Graphing

In this method, a specification is analyzed, and *causes* (inputs or stimuli that elicit responses from the software system) and *effects* (outputs, changes in the system state, or other observable responses) are identified [30]. The causes and effects must be expressed in a way such that they may either be true or false. Suppose, for example, a specification states that "if the record exists in the master file then a report line will be written." With this information, the cause and the effect would have to be expressed as "Record exists in the master file" and "Report line is written", respectively. This way of expression is necessary because the causes and effects will be combined in a boolean graph (called a *cause-effect graph*), which describes the relationships between them. In the graph, several boolean operators, such as NOT, AND, NAND, OR, and NOR are used.

With the cause-effect graph, the next step is to convert it into a decision table. This table shows the effects occurring for every possible combination of causes. Thus, given n causes, the decision table will have a total of 2^n entries. In practice, the presence of environment constraints will help reduce the number of entries [67]. Thereafter, each column of the decision table is converted into a test case, by referring back to the original list of causes and effects and invoking those that are true and suppressing those that are false. The expected output is defined in a similar way.

(c) Choice Relation Framework

In the **CHO**i**C**e re**LAT**ion fram**E**work (abbreviated as CHOC'LATE) [21, 62], an

early and important step is to "formalize" the original informal specification, by converting it into a highly structured intermediate format, known as a choice relation table and is denoted by $\mathcal{T}$. Compared to the original and informal specification, $\mathcal{T}$ is closer to the right of the spectrum in Figure 1.1.

Basically, a $\mathcal{T}$ captures some *choices* (distinct subsets of values of the input domain, so that all values in the same choice are assumed to be similar either in their effect on the system's behavior, or in the type of output they produce) and their relations, which are identified from the original specification. Test cases are then generated from $\mathcal{T}$ using some predefined algorithms provided by CHOC'LATE. Note that the correctness of choice relations captured in $\mathcal{T}$ is very important because it affects the comprehensiveness of the generated test suite. In view of the importance of choice relations, CHOC'LATE provides useful mechanisms for: (i) checking the consistency of manually defined choice relations, and (ii) automatically deducing some choice relations yet to be determined whenever possible. We shall further discuss CHOC'LATE in detail in Chapter 2 of this thesis.

(d) Classification-Tree Methodology

The original classification-tree method is first developed by Grochtmann et al. [39, 40]. Later, Chen and his coworkers have extended the original method, resulting in an integrated classification-tree methodology [10, 11, 20]. For the rest of the thesis, both the original and the extended methods will be collectively referred to as the **C**lassification-**T**ree **M**ethodology and is denoted by CTM.

In general, CTM is fairly similar to CHOC'LATE in the approach to generating test cases, particularly the conversion of the informal specification to a relatively more formal representation through which test cases can be generated. With respect to this formal representation, while CHOC'LATE uses a choice relation table as the

intermediate representation, CTM uses a hierarchical tree structure to achieve this purpose. Same as CHOC'LATE, we shall discuss CTM in detail in Chapter 2 of this thesis.

Among the above four methods, CHOC'LATE and CTM have received more attention because of three major reasons. Firstly, it is argued that, in random testing, the automatic generation of pseudo-random values as test cases is biased and the operational profile used does not exist in most real-life situations [42]. Secondly, compared to CHOC'LATE and CTM, cause-effect graphing is more difficult to apply, especially for large and complex software systems. This is because of the tedious task of generating the boolean cause-effect graph. Thirdly, CHOC'LATE and CTM have been used in different application domains, including online telephone enquires (with more than 60 000 enquires a day), inventory management for a large group of hospitals, and airfield lighting control of an international airport. The results of such usage are very encouraging [21, 39, 62].

1.3 Motivation of Thesis

In CHOC'LATE and CTM, an early step is to identify a set of categories (also known as classifications) and their associated choices (also known as classes) from the specification.[4] A *category* is defined as a major property or characteristic of an explicit input or a state (at the time of execution) of the software system that affects its execution behavior. *Choices* are distinct subsets of possible values associated with each category. Consider, for example, a system related to the credit sales of goods by a wholesaler to retail customers. A possible category is [Customer's Credit Limit]. Two possible choices

[4] Categories and choices in CHOC'LATE are equivalent to classifications and classes in CTM, respectively. For ease of presentation, we shall refer to them only as categories and choices.

in this category are |Customer's Credit Limit: ≤ 0| and |Customer's Credit Limit: > 0|. [5]

With a set of categories and choices, the next step is to identify the constraints among the choices (in CHOC'LATE) or the categories (in CTM). Such constraints are used to guide the generation of valid combinations of choices by means of some predefined algorithms. Finally, such valid combinations of choices are used to generate test cases.

Obviously, the chance of detecting software faults depends on the comprehensiveness of the generated test cases, which in turn depends on the comprehensiveness of the identified categories and choices. If, for example, a valid choice is missing, then any fault associated with this choice may not be detected. We observe that neither the original developers of CHOC'LATE/CTM nor follow-up researchers have proposed a systematic method for identifying categories and choices from informal specifications. As a result, this identification process is often performed in an ad hoc, impromptu manner. Without doubt, an impromptu approach cannot assure the quality of the identified categories and choices.

1.4 Overview of Thesis

The work of this thesis is inspired by the problems of an impromptu identification approach associated with CHOC'LATE and CTM. Our ultimate objective is to develop a systematic methodology for identifying categories and choices that can be applied to complex specifications with many different types of component (for example, narrative descriptions, activity diagrams, and use cases). Through this identification methodology, the effectiveness and popularity of CHOC'LATE and CTM can be improved.

In Chapter 2, we shall give a literature review of CHOC'LATE and CTM. In addition,

[5] Categories are enclosed by square brackets [] and choices are enclosed by vertical bars | |. Furthermore, the notation $|X : x|$ denotes a choice $|x|$ in the category $[X]$.

we shall discuss in detail the major problems associated with an impromptu approach to identifying categories and choices in CHOC'LATE and CTM. Furthermore, we shall briefly discuss the previous work on category and choice identification by other researchers.

Chapter 3 reports and discusses our two rounds of empirical studies, with a view to providing the background motivation for our developed identification methodology to be described in Chapter 4. Chapter 3 consists of two parts. The first part focuses on our first round of studies. These studies aim to investigate the common mistakes made by *inexperienced* software testers in an impromptu identification approach. The studies serve three purposes: (a) to make the knowledge of common mistakes known to other testers so that they can avoid repeating the same mistakes, (b) to facilitate researchers and practitioners develop systematic identification methodologies, and (c) to provide a means of measuring the effectiveness of newly developed identification methodologies. Here we shall also present a checklist to help testers detect the common mistakes. This checklist is formulated based on the results of our studies.

In the second part of Chapter 3, we first describe our second round of studies and discuss the major results. While the first round of studies involves inexperienced testers as subjects, the second round of studies focuses on *experienced* testers with many years of commercial experience in software development and testing. The objectives of the second round of studies are: (a) to investigate the differences in the types and amounts of mistakes made between inexperienced and experienced testers in an impromptu identification approach, and (b) to determine, after discussing the mistakes with the testers and providing them with a checklist as a simple guideline for detecting problematic categories and choices, the extent of mistake reduction in the next identification exercise. Thereafter, we shall discuss which specification components are useful for category and choice identification, according to the opinions of the experienced subjects. We shall also report the suggestions by the subjects on how the effectiveness of the identification

exercises can be improved.

Chapter 4 presents the most significant work of the thesis. Here, we introduce a **D**ivid**E**-and-conquer methodology for identifying categorie**S**, choice**S**, and choic**E** **R**elations for **T**est case generation (abbreviated as DESSERT). The theoretical framework and the associated algorithms of DESSERT will be discussed in detail. (There are three algorithms in DESSERT, namely construct_table, integrate_table, and refine_table. The first algorithm construct_table only works with activity diagrams, while the other two algorithms do not have this limitation.) We shall also describe two case studies using commercial specifications to demonstrate the effectiveness of DESSERT.

Finally, Chapter 5 concludes this thesis by summarizing the contribution of our work, and suggests possible directions of future work.

Chapter 2

Literature Review

2.1 Overview of Choice Relation Framework (CHOC'LATE)

CHOC'LATE is a specification-based testing technique developed by Chen and his coworkers [21, 62]. It helps software testers generate test cases from specifications via the notion of a choice relation table (denoted by $\mathcal{T}$). The intuition of $\mathcal{T}$ is to capture the constraints imposed on the choices by the specification. These constraints are expressed as relations between pairs of choices. They are essential information for the automatic generation of test frames. Such test frames are then further used for generating test cases.

Basically, CHOC'LATE consists of the following major steps:

(1) Decompose a specification into functional units that can be tested independently. For each functional unit selected for testing, repeat steps (2) to (6) below.

(2) Identify the *parameters* (the explicit inputs to a functional unit) and *environment conditions* (the states of the system at the time of execution) that affect the execution behavior of the function. For the ease of presentation, parameters and environment conditions are collectively known as *factors* in this thesis. In addition, any factor is

said to be *influencing* if it affects the execution behavior of the function.

Thereafter, find *categories* (major properties or characteristics) of information that characterize each influencing factor. For each category $[X]$, partition it into *choices*, which include all the different kinds of values that are possible for $[X]$. Note that all the values in the same choice are assumed to be similar either in their effect on the system's behavior, or in the type of output they produce. Furthermore, given any $[X]$, all its associated choices must be distinct and they together should cover the input domain relevant to $[X]$.

(3) Construct a choice relation table $\mathcal{T}$ to capture the constraints among choices.

(4) Construct a choice priority table $\mathcal{P}$ and define the appropriate parameters.

(5) Construct a set of complete test frames.

(6) Create a test case from each generated complete test frame.

Before we give an example to illustrate the above steps, we shall introduce some important concepts related to CHOC'LATE. These concepts are originally defined by Chen and his coworkers in [21, 62].

Definition 2.1 (Test Frame and its Completeness)
*A **test frame** B is a set of choices. B is **complete** if, whenever a single element is selected from every choice in B, a test case is formed. Otherwise, B is **incomplete**.*

We shall use B^c and *tc* to denote any complete test frame and any test case, respectively. It follows immediately from Definition 2.1 that every B^c is a valid combination of choices. Only B^c's are used for the subsequent generation of test cases in step (6) of CHOC'LATE. On the other hand, incomplete B's are not useful for testing. They should either be discarded or extended into B^c's [21, 62].

Definition 2.2 (Set of Complete Test Frames Related to a Choice)

Let TF denote the set of all B^c's. Given any choice $|X : x|$, we define the ***set of complete test frames related to*** $|X : x|$ *as* $TF(|X : x|) = \{B^c \in TF : |X : x| \in B^c\}$.

Definition 2.3 (Validity of a Choice)

Any choice $|X : x|$ is ***valid*** *if $TF(|X : x|) \neq \emptyset$. Otherwise, it is* ***invalid****.*

For the rest of the thesis, unless otherwise stated, valid choices are simply referred to as "choices". Obviously, valid choices (albeit not invalid choices) are useful for generating B^c's.

In step (5) of CHOC'LATE, valid choice combinations are generated as B^c's. The generation process needs to consider the constraints among choices, as defined by the software tester and captured in a choice relation table $\mathcal{T}$ in step (3). Basically, $\mathcal{T}$ captures the constraints between each pair of choices. Thus, given k number of choices, the dimension of $\mathcal{T}$ is $k \times k$. The constraints between choices are formalized in the concept of choice relations, which are expressed in the following definition:

Definition 2.4 (Choice Relation between Two Choices)

Given any choice $|X : x|$, its ***relation*** *with another choice $|Y : y|$ (denoted by $|X : x| \mapsto |Y : y|$) is defined as follows:*

(*a*) $|X : x|$ *is* ***fully embedded*** *in $|Y : y|$ (denoted by $|X : x| \sqsubset |Y : y|$) if and only if $TF(|X : x|) \subseteq TF(|Y : y|)$.*

(*b*) $|X : x|$ *is* ***partially embedded*** *in $|Y : y|$ (denoted by $|X : x| \Subset |Y : y|$) if and only if $TF(|X : x|) \not\subseteq TF(|Y : y|)$ and $TF(|X : x|) \cap TF(|Y : y|) \neq \emptyset$.*

(*c*) $|X : x|$ *is* ***not embedded*** *in $|Y : y|$ (denoted by $|X : x| \not\sqsubset |Y : y|$) if and only if $TF(|X : x|) \cap TF(|Y : y|) = \emptyset$.*

In Definition 2.4, the symbols "$\sqsubset$", "$\boxminus$", and "$\not\sqsubset$" are called *relational operators*. Since the three types of choice relation in the definition are exhaustive and mutually exclusive, $|X : x| \mapsto |Y : y|$ can be uniquely determined. In addition, immediately from the definition, the relational operator for $|X : x| \mapsto |X : x|$ is "$\sqsubset$", and that for $|X : x_1| \mapsto |X : x_2|$ is "$\not\sqsubset$" if $|X : x_1| \neq |X : x_2|$ because of the following corollary:

Corollary 2.1 (Mutual Exclusion of Choices in a Category)

Given two distinct choices $|X : x_1|$ *and* $|X : x_2|$, $TF(|X : x_1|) \cap TF(|X : x_2|) = \emptyset$.

After introducing the above concepts, we are now ready to use an example to illustrate the major steps of CHOC'LATE.

Example 2.1 (Choice Relation Framework)

Suppose a software tester is given the following program, COUNT, for testing:

COUNT (IN list: LIST, IN element: ELEM, OUT number: INT)

The inputs to COUNT are a list of elements (LIST) and an element to be counted (ELEM), whereas the output from COUNT is the number of occurrences (INT) of ELEM in LIST.

Suppose, because of its simplicity, COUNT can be tested without decomposing it into smaller, independent functional units (see step (1) of CHOC'LATE). In other words, the entire program can be treated as a single functional unit denoted by $\mathbb{U}_{\mathsf{COUNT}}$. Also, suppose that the categories and choices for $\mathbb{U}_{\mathsf{COUNT}}$ are identified as listed in Table 2.1 (see step (2) of CHOC'LATE). In this table, the category [Status of LIST] is defined with respect to an influencing environment condition of $\mathbb{U}_{\mathsf{COUNT}}$, whereas the remaining categories are defined with respect to influencing parameters of $\mathbb{U}_{\mathsf{COUNT}}$. Note that:

- Choices |Status of LIST: Does Not Exist| and |Length of LIST: $= 0$| in Table 2.1 are used to test the execution behavior of $\mathbb{U}_{\mathsf{COUNT}}$ under the abnormal situation where LIST does not exist or LIST exists but is empty.

Category	Associated Choices
[Status of LIST]	|Status of LIST: Does Not Exist|, |Status of LIST: Exists|
[Length of LIST]	|Length of LIST: $= 0$|, |Length of LIST: $= 1$|, |Length of LIST: > 1|
[Existence of ELEM in LIST] *	|Existence of ELEM in LIST: Yes|, |Existence of ELEM in LIST: No|
[Number of Occurrences of ELEM in LIST]	|Number of Occurrences of ELEM in LIST: $= 0$|, |Number of Occurrences of ELEM in LIST: $= 1$|, |Number of Occurrences of ELEM in LIST: > 1|
[Sorting Sequence of Elements in LIST]	|Sorting Sequence of Elements in LIST: Ascending Order| †, |Sorting Sequence of Elements in LIST: Descending Order| †, |Sorting Sequence of Elements in LIST: Unsorted| †, |Sorting Sequence of Elements in LIST: All Elements are Identical|

($\star$) This category assumes that the length of LIST is 1.

($\dagger$) These three choices assume that some elements in LIST are different.

Table 2.1: Categories and Choices for $\mathbb{U}_{\mathsf{COUNT}}$

- A choice is considered as a set of its possible values. For instance, |Length of LIST: > 1| $= \{2,3,4,\dots\}$. Because a choice may correspond to a range of values, so although the union of choices for any category $[X]$ should cover the input domain relevant to $[X]$, the number of choices in $[X]$ is not necessarily large.

Now, consider step (3) of CHOC'LATE, in which the choice relation between every pair of choices is determined. Let us look at some examples. The relational operator for |Length of LIST: $= 1$| $\mapsto$ |Status of LIST: Exists| is "$\sqsubset$". Because the length of LIST is relevant only when LIST exists in the first place, therefore, given any complete test frame B^c (recall that every B^c is a valid combination of choices) containing |Length of LIST: $= 1$|, B^c must also contain |Status of LIST: Exists|. By similar reasons, the relational operator for |Length of LIST: $= 1$| $\mapsto$ |Status of LIST: Does Not Exist| is "$\not\sqsubset$". This is because |Length of LIST: $= 1$| and |Status of LIST: Does Not Exist| cannot coexist in any B^c. The relational operator for |Status of LIST: Exists| $\mapsto$ |Length of LIST: $= 0$| is

"$\mathbb{E}$". This is because, if LIST exists, it may be empty (corresponding to |Length of LIST: $= 0$|) or it may contain one or more elements (corresponding to |Length of LIST: $= 1$| or |Length of LIST: > 1|). Thus, given any B^c containing |Status of LIST: Exists|, B^c may or may not contain |Length of LIST: $= 0$|. The relational operator for the remaining choice relations can be determined in a similar way.

Let us proceed to step (4) of CHOC'LATE. Many real-life situations impose resource constraints on testing and, hence, not all B^c's generated will actually be used in the testing process. Intuitively, it would be more effective to have an idea of the kinds of fault that are most probable or most damaging, and then generate B^c's that are likely to reveal these significant faults. One approach is to define the relative priorities for the choices based on expertise in testing and experience in the application domain. In this way, the choices with higher priorities can first be used to generate test frames B's. This generation process will continue until the number of generated B's reaches the ceiling allowed by the testing resources. [1] The relative priorities of the choices are captured in a choice priority table $\mathcal{P}$.

Besides constructing $\mathcal{P}$, the tester also needs to define the ceiling permitted by the testing resources. This is achieved through the parameter *preferred maximum number of test frames* (denoted by $\overline{M}$). The word "preferred" implies that $\overline{M}$ is not absolute, as the ceiling may be overwritten by the parameter to be described in the next paragraph.

In addition to $\overline{M}$, we have another parameter, which indicates the minimal priority level (denoted by $\underline{m}$). Any choice having a relative priority higher than $\underline{m}$ will always be selected for inclusion as part of a B, no matter whether the number of generated B's exceeds $\overline{M}$. In essence, $\underline{m}$ guarantees that the choices more likely to detect significant faults will always be used to form B's irrespectively of the testing resources.

In step (5), CHOC'LATE adopts an incremental approach to generating B's based on

[1] The process adopts an incremental approach to generating B's. When the process ends, most or all of the B's will become B^c's.

$\mathcal{T}$, $\mathcal{P}$, $\overline{M}$, and $\underline{m}$. Most parts of the generation process are automatically performed by CHOC'LATE without human intervention. Readers may refer to [21, 62] for the details. Suppose we set $\overline{M}$ to the largest value supported by the system so that all possible B^c's are generated. In this case, a total of 15 B^cs will be generated as follows:

- $B_1^c = \{|\text{Status of LIST: Does Not Exist}|\}$
- $B_2^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} = 0|\}$
- $B_3^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} = 1|, |\text{Existence of ELEM in LIST: Yes}|\}$
- $B_4^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} = 1|, |\text{Existence of ELEM in LIST: No}|\}$
- $B_5^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} > 1|, |\text{Number of Occurrences of ELEM in LIST:} = 0|, |\text{Sorting Sequence of Elements in LIST: Ascending Order}|\}$
- $B_6^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} > 1|, |\text{Number of Occurrences of ELEM in LIST:} = 0|, |\text{Sorting Sequence of Elements in LIST: Descending Order}|\}$
- $B_7^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} > 1|, |\text{Number of Occurrences of ELEM in LIST:} = 0|, |\text{Sorting Sequence of Elements in LIST: Unsorted}|\}$
- $B_8^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} > 1|, |\text{Number of Occurrences of ELEM in LIST:} = 0|, |\text{Sorting Sequence of Elements in LIST: All Elements are Identical}|\}$
- $B_9^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} > 1|, |\text{Number of Occurrences of ELEM in LIST:} = 1|, |\text{Sorting Sequence of Elements in LIST: Ascending Order}|\}$
- $B_{10}^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} > 1|, |\text{Number of Occurrences of ELEM in LIST:} = 1|, |\text{Sorting Sequence of Elements in LIST: Descending Order}|\}$

- $B^c_{11} = \{|\text{Status of LIST: Exists}|,\ |\text{Length of LIST:} > 1|,\ |\text{Number of Occurrences of ELEM in LIST:} = 1|,\ |\text{Sorting Sequence of Elements in LIST: Unsorted}|\}$
- $B^c_{12} = \{|\text{Status of LIST: Exists}|,\ |\text{Length of LIST:} > 1|,\ |\text{Number of Occurrences of ELEM in LIST:} > 1|,\ |\text{Sorting Sequence of Elements in LIST: Ascending Order}|\}$
- $B^c_{13} = \{|\text{Status of LIST: Exists}|,\ |\text{Length of LIST:} > 1|,\ |\text{Number of Occurrences of ELEM in LIST:} > 1|,\ |\text{Sorting Sequence of Elements in LIST: Descending Order}|\}$
- $B^c_{14} = \{|\text{Status of LIST: Exists}|,\ |\text{Length of LIST:} > 1|,\ |\text{Number of Occurrences of ELEM in LIST:} > 1|,\ |\text{Sorting Sequence of Elements in LIST: Unsorted}|\}$
- $B^c_{15} = \{|\text{Status of LIST: Exists}|,\ |\text{Length of LIST:} > 1|,\ |\text{Number of Occurrences of ELEM in LIST:} > 1|,\ |\text{Sorting Sequence of Elements in LIST: All Elements are Identical}|\}$

The above list clearly shows that B^c's (and also B's) are sets of choices (see Definition 2.1). Let us provide an example of incomplete test frames. Consider $B_1 = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} = 1|\}$. B_1 is incomplete because we need additional information about the existence of ELEM in LIST in order to generate test cases for testing $\mathbb{U}_{\mathsf{COUNT}}$.

Now, consider the choices |Status of LIST: Does Not Exist| and |Length of LIST: $= 1$|. According to Definition 2.2, $TF(|\text{Status of LIST: Does Not Exist}|) = \{B^c_1\}$ and $TF(|\text{Length of LIST:} = 1|) = \{B^c_3, B^c_4\}$. Because $TF(|\text{Status of LIST: Does Not Exist}|)$ and $TF(|\text{Length of LIST:} = 1|)$ are nonempty, |Status of LIST: Does Not Exist| and |Length of LIST: $= 1$| are valid choices (see Definition 2.3).

In the last step (6) of CHOC'LATE, for every B^c, a test case is generated by randomly selecting and combining an element from every choice in B^c. Consider, for instance, B^c_{12}.

A possible test case *tc* generated from B_{12}^{c} is:

$$\begin{aligned} tc = \{&\text{Status of LIST} = \text{Exists},\ \text{Length of LIST} = 4,\\ &\text{Number of Occurrences of ELEM in LIST} = 2,\\ &\text{Sorting Sequence of Elements in LIST} = \text{Ascending Order}\} \end{aligned}$$

Note that, in *tc*, the values "4" and "2" are randomly selected from the choices |Length of LIST: > 1| and |Number of Occurrences of ELEM in LIST: > 1|, respectively. For the purpose of discussion, a test case is considered as a set of its values. Thus, for example, (Length of LIST $= 4$) $\in$ *tc*.

Let us revisit *tc*. Possible inputs to $\mathbb{U}_{\mathsf{COUNT}}$ corresponding to *tc* are: LIST = $(1,3,3,9)$ and ELEM $= 3$. Note that, here, LIST contains four elements in ascending order. Also, the element ELEM to be counted occurs twice in LIST. ■

2.2 Overview of Classification-Tree Methodology (CTM)

The original classification-tree method is first developed by Grochtmann et al. [39, 40]. Later, it is extended by Chen and his coworkers, resulting in their integrated classification-tree methodology [10, 11, 20]. In this thesis, both the original and the extended methods are collectively known as the **C**lassification-**T**ree **M**ethodology (CTM).

Basically, CTM helps testers generate test cases from specifications via the construction of classification trees. *Classifications* are defined as the criteria for partitioning the input domain of the software, whereas *classes* are defined as the disjoint subsets of values for each classification. Thus, classifications and classes in CTM are equivalent to categories and choices in CHOC'LATE, respectively. Because of this reason, for ease of presentation, we shall refer to them only as categories and choices. CTM is fairly similar to CHOC'LATE in the approach to test case generation. In general, CTM consists of the

following major steps:

(1) Decompose a specification into functional units that can be tested independently. For each functional unit selected for testing, repeat steps (2) to (5) below.

(2) Identify a set of categories and choices.

(3) Organize the identified categories and choices into a tree structure called a *classification tree*, by considering the constraints among categories.

(4) Construct a set of complete test frames from the classification tree.

(5) Construct a test case from each generated complete test frame.

Note that steps (1) and (5) of CTM are identical to steps (1) and (6) of CHOC'LATE, respectively. (Discussion on both steps is given in Section 2.1 and, hence, will not be repeated here.) Also, steps (2), (3), and (4) of CTM correspond to steps (2), (3), and (5) of CHOC'LATE, respectively. The following example illustrates steps (2), (3), and (4) of CTM in detail:

Example 2.2 (Classification-Tree Methodology)

Refer to the program COUNT and the functional unit $\mathbb{U}_{\mathsf{COUNT}}$ in Example 2.1. Suppose the categories and choices shown in Table 2.1 are identified for $\mathbb{U}_{\mathsf{COUNT}}$ in step (2) of CTM. In step (3) of CTM, we identify the constraints among *categories* and capture these constraints in a classification tree. As such, this step differs from step (5) of CHOC'LATE in which constraints among *choices* are identified and captured in a choice relation table $\mathcal{T}$.

Suppose further that the classification tree $\Upsilon_{\mathbb{U}_{\mathsf{COUNT}}}$ for $\mathbb{U}_{\mathsf{COUNT}}$ is constructed as depicted in Figure 2.1.[2] In the figure, the small circle at the top of $\Upsilon_{\mathbb{U}_{\mathsf{COUNT}}}$ is called

[2] In [20], Chen and his coworkers have developed an algorithm to construct a classification tree from a given set of categories and choices.

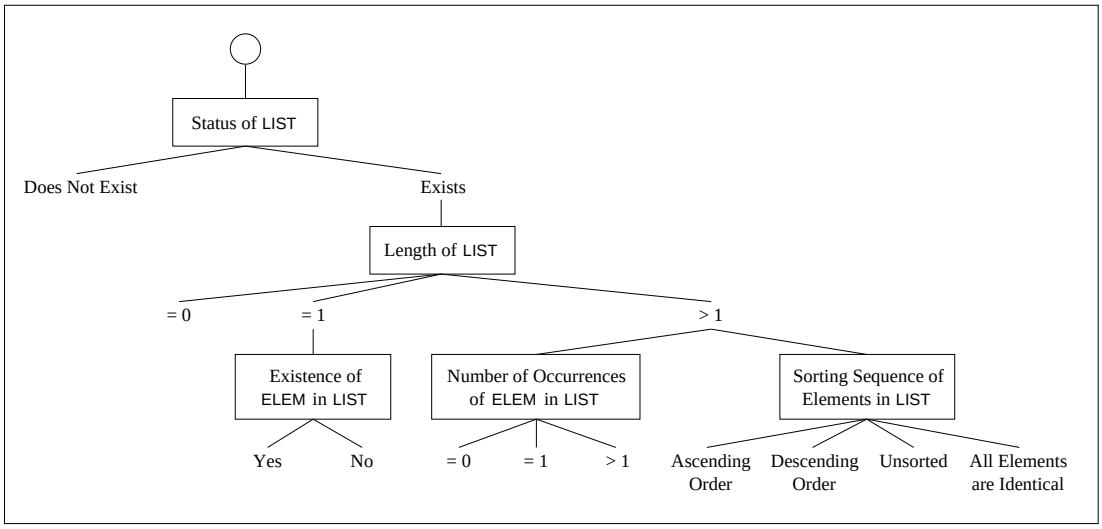


Figure 2.1: Classification Tree $\Upsilon_{\mathbb{U}_{\mathsf{COUNT}}}$

the *general root node,* representing the input domain of $\mathbb{U}_{\mathsf{COUNT}}$. In the tree, categories are enclosed in boxes where choices are not. Figure 2.1 shows that a classification tree which arranges categories and choices at alternate levels.

CTM provides predefined rules to generate a set of test frames from a classification tree (readers may refer to [10, 11, 20, 39, 40] for the details of these rules). For example, after applying the rules to $\Upsilon_{\mathbb{U}_{\mathsf{COUNT}}}$, a total of 16 test frames are generated. These 16 test frames include the 15 B^c's (that is, B^c_1, B^c_2, ..., B^c_{15}) listed in Example 2.1, plus the following:

$$B' = \{|\text{Status of LIST: Exists}|,\ |\text{Length of LIST:} > 1|,$$
$$|\text{Number of Occurrences of ELEM in LIST:} = 1|,$$
$$|\text{Sorting Sequence of Elements in LIST: All Elements are Identical}|\}$$

Note that B' above represents an invalid combination of choices and, hence, not useful for testing. This is because, when LIST contains two or more elements and all these elements are identical, ELEM cannot occur only once in LIST. Thus, after generating a set of test frames in step (4) of CTM, we need to check all of them and discard those which are not

useful. After discarding the unuseful test frames, only B^c's remain. Thereafter, in step (5) of CTM, a test case is generated from each of these remaining B^c's, by randomly selecting and combining an element from every choice in B^c. ■

2.3 Major Problems

There are three major problems in CHOC'LATE and CTM as mentioned in Sections 2.1 and 2.2:

Problem 1: Examples 2.1 and 2.2 show that the effectiveness of both methods largely depends on how well steps (1)–(6) of CHOC'LATE (listed in Section 2.1) and steps (1)–(5) of CTM (listed in Section 2.2) are performed. Although most of these steps are supported by well-developed techniques or algorithms (such as the construction of choice relation tables and classification trees from a *given* set of categories and choices, and the subsequent construction of B^c's), not much has been developed to help software testers identify categories and choices, especially for *informal* specifications. Thus, the identification of categories and choices for informal specifications is often done in an ad hoc, impromptu manner. Such a practice is unacceptable. The comprehensiveness of the identified categories and choices cannot be ascertained by an impromptu approach. If, for example, the software tester fails to identify a choice $|X : x|$, then any B^c (and any test cases constructed from it) containing $|X : x|$ will not be generated. Consequently, any software fault associated with $|X : x|$ may not be detected.

Motivated by this problem, Grochtmann and Grimm [39] have investigated the feasibility of applying artificial intelligence techniques to automatically identify categories and choices from informal specifications.[3] For instance, if X is an input parameter of a test

[3] Although identifying categories and choices from *informal* specifications is a general problem for black-box testing, the work by Grochtmann and Grimm [39] and our DESSERT methodology described in Chapter 4 of the thesis are the only attempts on this problem.

object, their identification approach is to generate a category $[X]$ with choices $|X :< 0|$, $|X := 0|$, and $|X :> 0|$. Obviously, the categories and choices generated in this way are often too "rough" and their identification approach does not result in much success. Grochtmann and Grimm argue that identifying categories and choices is a creative process that probably can never be done automatically in its entirety [39]. They have then shifted their attention to the identification process based on *formal* specifications. Other researchers, such as Hierons, Singh, and their coworkers [44, 72], have also conducted work in this direction. While we concur with the view that the identification of categories and choices from informal specifications is challenging and cannot be fully automatic, we take the position that research on identification processes for informal specifications is a must, because this type of specification is more commonly accepted by the software industry.

Problem 2: Steps (2) and (3) of both methods are not totally distinct and independent. More specifically, we need to consider the constraints among choices (in CHOC'LATE) or categories (in CTM) in step (3) when identifying choices in step (2). Otherwise, the identification process may not be well executed, which in turn affects the comprehensiveness of the generated test cases [15] (further discussion of this problem will be given in Section 3.2.5 of Chapter 3 later). This need makes the identification process more complicated.

Problem 3: Software testers may find difficult identifying categories, choices, and constraints from the entire specification, especially for an informal specification with many different components such as use cases, activity diagrams, class diagrams, state machines, and data flow diagrams.

Inspired by these problems, in Chapter 4, we propose a systematic methodology known as ***DESSERT*** (which stands for *a **D**ivid**E**-and-conquer methodology for identifying*

*categorie**S**, choice**S**, and choic**E** **R**elations for **T**est case generation*) to help software testers identify categories and choices (for problem 1), as well as choice relations (for problem 2) from specifications. An appealing feature of DESSERT is that the identification process is conducted by focusing on one specification component or model at a time (for problem 3). Before we present the details of DESSERT, we shall first discuss our related empirical and case studies in Chapter 3.

Chapter 3

Our Two Rounds of Empirical Studies

This chapter reports and discusses our two rounds of empirical studies, with a view to providing the background motivation for our developed identification methodology to be described in Chapter 4.

3.1 First Round of Studies

3.1.1 Purposes of Studies

Motivated by the three problems as listed in Section 2.3, we have conducted the first round of empirical studies on the identification of categories and choices from informal specifications [15]. Our primary objective is to find out the common mistakes made by software testers when identifying categories and choices in the absence of a systematic process.[1] Our studies have three contributions:

(a) To reduce the chance of repeating these mistakes by making them known to testers.

[1] The checking of mistakes is manually done by us, who have substantial experience in CHOC'LATE and CTM. This is also the case for our second round of studies to be described later.

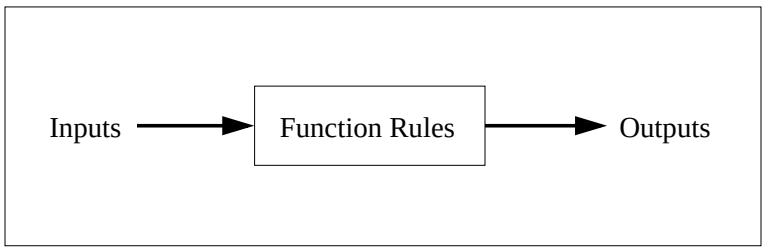


Figure 3.1: The Function Model of a Software System

(b) To shed light on the development of systematic techniques for identifying categories and choices.

(c) To provide a means of measuring the effectiveness of newly developed identification techniques in terms of their ability to avoid or reduce mistakes.

Based on the observations of the studies, we also formulate a checklist to help testers detect such mistakes [15]. The checklist is included in Section 3.1.5.

3.1.2 Overview of Function Models and Function Rules

Before we proceed further, we have to introduce the notions of function models and function rules [81]. A *function model* represents the behavior of the system at an abstract level, so that software developers and users can agree on the system behavior without the need for programming details. The mapping between a given set of inputs and the corresponding set of outputs is expressed by means of a *function rule*. This rule states precisely the preconditions for the function to execute and how the outputs are related to the inputs (see Figure 3.1). Note that most function models assume that the system is *deterministic*, or in other words, the same set of inputs would always lead to the same system behavior and, hence, the same set of outputs. In this thesis, we also make this assumption.

Function Rule	Set of Valid Inputs	Corresponding Output	Corresponding Complete Test Frame
1	LIST does not exist	Output the message "LIST does not exist"	$B_1^c = \{\lvert$Status of LIST: Does Not Exist$\rvert\}$
2	LIST exists but is empty	Output the message "LIST is empty"	$B_2^c = \{\lvert$Status of LIST: Exists$\rvert$, $\lvert$Length of LIST: $= 0\rvert\}$
3	LIST exists, contains ELEM only once, and does not contain any other element	Output the message "Number of Occurrences of ELEM in LIST $= 1$"	$B_3^c = \{\lvert$Status of LIST: Exists$\rvert$, $\lvert$Length of LIST: $= 1\rvert$, $\lvert$Existence of ELEM in LIST: Yes$\rvert\}$
⋮	⋮	⋮	⋮

Table 3.1: A Partial Function Model for Program COUNT

Consider Example 2.1 in Section 2.1 again. Part of the function model for the program COUNT is depicted in Table 3.1. Each row corresponds to a function rule. The rightmost element in each row is a possible complete test frame B^c, from which a test case can be generated to execute the rule associated with this row. Without doubt, sufficient B^c's should be generated with a view to uncovering any possible fault associated with each rule.

3.1.3 Terminology and Definitions

Readers may recall in Section 2.1 that *categories* are the major properties or characteristics of influencing factors of a functional unit (denoted by $\mathbb{U}$) or the entire system. For every category $[X]$ proposed by the subjects of our studies, it may either be identified according to the definition, or incorrectly identified with something else in mind. In view of this situation, we shall refer to any $[X]$ identified by the subjects as a *potential category*. Similarly, any $|X : x|$ identified by the subjects is called a *potential choice*.

In addition to Definitions 2.1 to 2.4, further terminology and definitions will be required to lay the foundation for the problems related to the identification of categories and choices. Such terminology and definitions are given below:

Definition 3.1 (Set of Complete Test Frames Related to a Potential Category)
Let $TF_{\mathbb{U}}$ denote the set of complete test frames of a functional unit $\mathbb{U}$. Given any potential category $[X]$ for $\mathbb{U}$ and all its associated potential choices $|X : x_1|$, $|X : x_2|$, ..., $|X : x_n|$, we define the ***set of complete test frames related to*** **[*X*]** *as $TF_{\mathbb{U}}([X]) = \{B^c \in TF_{\mathbb{U}} : |X : x_i| \in B^c$ for some $1 \leq i \leq n\}$.*

Example 3.1 (Set of Complete Test Frames Related to a Potential Category)
Refer to Example 2.1. The set of complete test frames for the functional unit $\mathbb{U}_{\mathsf{COUNT}}$, denoted by $TF_{\mathbb{U}_{\mathsf{COUNT}}}$ is $\{B_1^c, B_2^c, \ldots, B_{15}^c\}$. Consider the potential category [Existence of ELEM in LIST]. Here $TF_{\mathbb{U}_{\mathsf{COUNT}}}([\text{Existence of ELEM in LIST}]) = \{B_3^c, B_4^c\}$. ■

Definition 3.2 (Set of Complete Test Frames Related to a Potential Choice)
Given any potential choice $|X : x|$ in a functional unit $\mathbb{U}$, we define the ***set of complete test frames related to*** **$|X : x|$** *as $TF_{\mathbb{U}}(|X : x|) = \{B^c \in TF_{\mathbb{U}} : |X : x| \in B^c\}$.*

Example 3.2 (Set of Complete Test Frames Related to a Potential Choice)
Consider the potential choices |Status of LIST: Does Not Exist| and |Length of LIST: > 1| in Example 2.1. Here $TF_{\mathbb{U}_{\mathsf{COUNT}}}(|\text{Status of LIST: Does Not Exist}|) = \{B_1^c\}$ and $TF_{\mathbb{U}_{\mathsf{COUNT}}}(|\text{Length of LIST: } > 1|) = \{B_5^c, B_6^c, \ldots, B_{15}^c\}$. ■

Definition 3.3 (Set of Complete Test Frames Related to a Test Frame)
Given any test frame B for a functional unit $\mathbb{U}$, we define the ***set of complete test frames related to B*** *as $TF_{\mathbb{U}}(B) = \{B^c \in TF_{\mathbb{U}} : B \subseteq B^c\}$. A test frame B is said to be* ***valid*** *if $TF_{\mathbb{U}}(B) \neq \emptyset$. Otherwise, it is said to be* ***invalid****.*

We observe from Definitions 2.1 and 3.3 that a valid test frame may or may not be complete.

Example 3.3 (Set of Complete Test Frames Related to a Test Frame)

Refer to Example 2.1. Suppose we have a test frame $B = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST: } > 1|, |\text{Number of Occurrences of ELEM in LIST: } = 0|\}$. The set of complete test frames related to B is $TF_{\mathbb{U}_{\mathsf{COUNT}}}(B) = \{B_5^c, B_6^c, B_7^c, B_8^c\}$. Because $TF_{\mathbb{U}_{\mathsf{COUNT}}}(B) \neq \emptyset$, B is a valid test frame. ■

Definition 3.4 (Relevant and Irrelevant Categories)

*Given any potential category $[X]$ for a functional unit $\mathbb{U}$, if $TF_{\mathbb{U}}([X]) \neq \emptyset$, then $[X]$ is known as a **relevant category**, or simply as a **category**. Otherwise, $[X]$ is known as an **irrelevant category**.*

Example 3.4 (Relevant and Irrelevant Categories)

Refer to Examples 2.1 and 3.1. Because $TF_{\mathbb{U}_{\mathsf{COUNT}}}([\text{Existence of ELEM in LIST}]) = \{B_3^c, B_4^c\}$, it is nonempty. Thus, [Existence of ELEM in LIST] is a relevant category.

Suppose a tester identifies [Some Elements in LIST are 0] as a potential category with |Some Elements in LIST are 0: Yes| and |Some Elements in LIST are 0: No| as its associated potential choices. Note that the set of complete test frames for $\mathbb{U}_{\mathsf{COUNT}}$ is $\{B_1^c, B_2^c, \ldots, B_{15}^c\}$, and neither |Some Elements in LIST are 0: Yes| nor |Some Elements in LIST are 0: No| appears in B_1^c, B_2^c, …, B_{15}^c. Hence, $TF_{\mathbb{U}_{\mathsf{COUNT}}}([\text{Some Elements in LIST are } 0]) = \emptyset$ and [Some Elements in LIST are 0] is therefore an irrelevant category. ■

Definition 3.5 (Missing Category)

Let $[X_K]$ denote the category associated with the influencing factor K. Suppose PC is a set of potential categories and their associated potential choices identified for a functional

unit $\mathbb{U}$. *If* $[X_K] \notin PC$, *then* $[X_K]$ *is a* ***missing category in PC****. In this case, we also say that PC is a* ***set with a missing category****.*

Intuitively, some complete test frames may not be generated because of missing categories. As a result, some function rules of a functional unit are not being tested, so that any faults associated with such rules may not be detected.

Example 3.5 (Missing Category)

Refer to Example 2.1 and Table 2.1. Suppose we have only identified the categories [Status of LIST], [Length of LIST], [Number of Occurrences of ELEM in LIST], and [Sorting Sequence of Elements in LIST] (and their associated choices), as if the category [Existence of ELEM in LIST] did not exist. In such circumstances, [Existence of ELEM in LIST] does not exist in the set *PC* of potential categories and choices identified for the functional unit $\mathbb{U}_{\mathsf{COUNT}}$. Consider the influencing factor "Existence of ELEM in LIST", and in particular whether "ELEM exists in LIST" or "ELEM does not exist in LIST". Let $[X_{\text{Existence of ELEM in LIST}}]$ denote the category corresponding to the influencing factor "Existence of ELEM in LIST". Since $[X_{\text{Existence of ELEM in LIST}}] \notin PC$, $[X_{\text{Existence of ELEM in LIST}}]$ is a missing category in *PC*, and *PC* is said to have a missing category.

Consider again the complete test frames $B_3^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} = 1|, |\text{Existence of ELEM in LIST: Yes}|\}$ and $B_4^c = \{|\text{Status of LIST: Exists}|, |\text{Length of LIST:} = 1|, |\text{Existence of ELEM in LIST: No}|\}$. Both of them contain a choice in the category [Existence of ELEM in LIST]. Thus, if [Existence of ELEM in LIST] is missing, B_3^c and B_4^c will be omitted by mistake because they cannot be generated solely from the choices in other categories. ■

Before we proceed any further, we shall briefly describe a real-life specification used in our first round of empirical studies, because it will be referred to in later examples

(Examples 3.6 to 3.9).[2] This specification, denoted by $\mathbb{S}_{\mathsf{MOS}}$, is prepared for an international company providing catering service for many different airlines. The company prefers to remain anonymous and will only be referred to as AIR-FOOD. $\mathbb{S}_{\mathsf{MOS}}$ has been produced for a meal ordering system (MOS) that helps AIR-FOOD determine the types (such as normal, child, and vegetarian) and numbers of meals to be prepared and loaded onto each flight served by AIR-FOOD. The system MOS has been fully developed and released for production use in AIR-FOOD for several years.

The specification $\mathbb{S}_{\mathsf{MOS}}$ contains various components such as narrative descriptions, screen layouts, and report layouts. $\mathbb{S}_{\mathsf{MOS}}$ can be decomposed into several functional units. For example, there is a unit, denoted by $\mathbb{U}_{\mathsf{MEAL}}$, directly related to the generation of daily meal schedules and other units related to the maintenance of the airline codes and city codes.

The system MOS stores many master flight schedules. Each of these schedules contains a data element called "Weekly Departure Pattern (WDP)", which indicates whether a flight departs on a daily basis. For a non-daily flight, WDP further indicates the day(s) of the week that the flight will depart. Consider, for example, the following two values of WDP:

(a) **WDP = "1234567":** The flight is a daily-flight. Note that a "1", "2", ..., and "7" in WDP indicate that the flight departs on Mondays, Tuesdays, ..., and Sundays, respectively.

(b) **WDP = "––34–6–":** The flight is a non-daily flight. It only departs on Wednesdays, Thursdays, and Saturdays.

According to the specification $\mathbb{S}_{\mathsf{MOS}}$, the master flight schedule for a daily flight will always be used to generate the corresponding daily meal schedule on every day of the

[2] Note that the mistakes described in these later examples are genuine ones which are made by subjects of our empirical studies.

week without further checks, as long as this master flight schedule is "current". (To avoid lengthy discussion, we shall skip the criteria for determining whether a given master flight schedule is current.) On the other hand, further checking is required for a non-daily flight even though its master flight schedule is current, in order to determine whether the corresponding daily meal schedule should be generated on a particular day of the week.

With the above description of the system MOS, we shall now introduce some other important definitions.

Definition 3.6 (Missing Choice)

Given a category $[X]$ *and its associated set of choices* $H_{[X]} = \{|X : x_1|, |X : x_2|, \ldots, |X : x_n|\}$ *for a functional unit* $\mathbb{U}$*, if there exist some other choice* $|X : x|$ *yet to be identified and some value* $v \in |X : x|$ *such that* $v \notin |X : x_i|$*, for any* $1 \leq i \leq n$*, then* $|X : x|$ *is a* ***missing choice with respect to*** $[X]$ *and* $H_{[X]}$*. In this case, we also say that* $[X]$ *is a* ***category with a missing choice with respect to*** $H_{[X]}$*. When there is no ambiguity, we shall simply refer to* $|X : x|$ *as a* ***missing choice****, and* $[X]$ *as a* ***category with a missing choice****.*

Similar to missing categories as defined in Definition 3.5, the existence of categories with missing choices will cause the omission of some complete test frames. As a result, we may overlook the testing of some parts of the system.

Example 3.6 (Missing Choice)

According to the specification $\mathbb{S}_{\mathsf{MOS}}$, there are three different types of master flight schedule (MFS), namely "Outdated", "Current", and "Future". (To avoid lengthy discussion, the details of how to determine the type of an MFS are not included here.) The types of MFS, together with some other information such as the weekly departure pattern (WDP), determine which MFS's are used to generate the corresponding daily meal schedules on a particular date. It is also specified in $\mathbb{S}_{\mathsf{MOS}}$ that the kitchen of AIR-FOOD installs a monitor to display all types of MFS. This arrangement helps kitchen staff to plan and

produce the required meals. Outdated, current, and future MFS's are displayed on the monitor in "Red", "Blue", and "Green" colors, respectively.

Given the above information, [Type of MFS] should be identified as a category, with |Type of MFS: Outdated|, |Type of MFS: Current|, and |Type of MFS: Future| as its associated choices. Suppose we have identified only |Type of MFS: Current| and |Type of MFS: Future| as choices. In this case, |Type of MFS: Outdated| is a missing choice and [Type of MFS] is a category with a missing choice. ■

Definition 3.7 (Overlapping Choices)

Given a category $[X]$ *for a functional unit* $\mathbb{U}$*, two distinct choices* $|X : x_1|$ *and* $|X : x_2|$ *are said to be* ***overlapping*** *if* $|X : x_1| \cap |X : x_2| \neq \emptyset$*. In this case,* $[X]$ *is a* ***category with overlapping choices****.*

Note that, according to Definition 3.7, given any pair of overlapping choices $|X : x_1|$ and $|X : x_2|$, they must be distinct.

Example 3.7 (Overlapping Choices)

The specification $\mathbb{S}_{\mathsf{MOS}}$ states that every seat in a flight belongs to one of the three classes, namely "First", "Business", and "Economy". For each class, passengers can order regular meals or special meals such as vegetarian meals. The types of regular and special meals are different across the three cabin classes. As expected, the regular and special meals offered in the first-class cabin are of better quality than those in the other cabins. The numbers of regular and special meals for each cabin class in a flight is kept in a master flight schedule associated with this flight, so that a corresponding daily meal schedule can be generated later.

With the above information, the following categories and choices should be identified for the functional unit $\mathbb{U}_{\mathsf{MEAL}}$:

(a) [Number of Regular Meals in First-Class Cabin] with |Number of Regular Meals in First-Class Cabin: $= 0$| and |Number of Regular Meals in First-Class Cabin: > 0| as its associated choices.

(b) [Number of Special Meals in First-Class Cabin] with |Number of Special Meals in First-Class Cabin: $= 0$| and |Number of Special Meals in First-Class Cabin: > 0| as its associated choices.

(c) [Number of Regular Meals in Business-Class Cabin] with |Number of Regular Meals in Business-Class Cabin: $= 0$| and |Number of Regular Meals in Business-Class Cabin: > 0| as its associated choices.

(d) [Number of Special Meals in Business-Class Cabin] with |Number of Special Meals in Business-Class Cabin: $= 0$| and |Number of Special Meals in Business-Class Cabin: > 0| as its associated choices.

(e) [Number of Regular Meals in Economy-Class Cabin] with |Number of Regular Meals in Economy-Class Cabin: $= 0$| and |Number of Regular Meals in Economy-Class Cabin: > 0| as its associated choices.

(f) [Number of Special Meals in Economy-Class Cabin] with |Number of Special Meals in Economy-Class Cabin: $= 0$| and |Number of Special Meals in Economy-Class Cabin: > 0| as its associated choices.

Thus, by selecting one choice from each of the above categories, $2^6 = 64$ different situations are possible when generating test frames.

Consider [Number of Regular Meals in First-Class Cabin] in (a) above. Suppose it is now identified with two associated choices |Number of Regular Meals in First-Class Cabin: $= 0$| and |Number of Regular Meals in First-Class Cabin: ≥ 0|. The two choices are overlapping because the element (Number of Regular Meals in First-Class Cabin $= 0$)

exists in both choices. Accordingly, [Number of Regular Meals in First-Class Cabin] is a category with overlapping choices. ■

Definition 3.8 (Combinable Choices)

Suppose $[X]$ *is a category for a functional unit* $\mathbb{U}$*. Two distinct choices* $|X : x_1|$ *and* $|X : x_2|$ *in* $[X]$ *are said to be* ***combinable*** *if, for any test frame B, both of the following conditions are satisfied:*

(*a*) $(B \cup \{|X : x_1|\})$ *is a complete test frame if and only if* $(B \cup \{|X : x_2|\})$ *is a complete test frame.*

(*b*) *If* $(B \cup \{|X : x_1|\})$ *and* $(B \cup \{|X : x_2|\})$ *are complete test frames, then they are associated with the same function rule of* $\mathbb{U}$*.*

In this case, $[X]$ *is known as a* ***category with combinable choices****.*

Following Definition 3.8, we should combine the choices $|X : x_1|$ and $|X : x_2|$ into a single choice $|X : x_1| \cup |X : x_2|$ so as to reduce the number of complete test frames and, hence, save testing effort. This replacement will not jeopardize the coverage of the resulting set of complete test frames with respect to the execution of the function rules of $\mathbb{U}$.

Example 3.8 (Combinable Choices)

Consider the category [Number of Regular Meals in First-Class Cabin] in bullet (a) of Example 3.7 again. Suppose this category is identified with the following three associated distinct choices:

(i) |Number of Regular Meals in First-Class Cabin: $= 0$|,

(ii) |Number of Regular Meals in First-Class Cabin: $= 1$|, and

(iii) |Number of Regular Meals in First-Class Cabin: > 1|.

According to the specification $\mathbb{S}_{\mathsf{MOS}}$, however, the functional unit $\mathbb{U}_{\mathsf{MEAL}}$ should treat (ii) and (iii) in exactly the same way under the same function rules. In particular, given any test frame B, B combines with (ii) to form a complete test frame B_1^c if and only if B combines with (iii) to form a complete test frame B_2^c. Furthermore, B_1^c and B_2^c should produce the same test results because there is no function rule that states otherwise. In such circumstances, (ii) and (iii) are combinable into a single choice |Number of Regular Meals in First-Class Cabin: ≥ 1| = |Number of Regular Meals in First-Class Cabin: $= 1$| $\cup$ |Number of Regular Meals in First-Class Cabin: > 1|. Thus, [Number of Regular Meals in First-Class Cabin] is a category with combinable choices.

By combining the choices (ii) and (iii), we can reduce the number of complete test frames and, hence, alleviate the testing effort. On the other hand, the coverage of the function rules is not compromised. ■

Definition 3.9 (Composite Choice)

Given a category $[X]$ *for a functional unit* $\mathbb{U}$*, any choice* $|X : x|$ *is said to be* ***composite*** *if there exist non-overlapping and non-combinable choices* $|X : x_1|$ *and* $|X : x_2|$ *in* $[X]$ *such that* $|X : x_1| \cup |X : x_2| \subseteq |X : x|$*. In this case,* $[X]$ *is known as a* ***category with composite choices****.*

It is obvious from Definition 3.9 that we should consider replacing the composite choice $|X : x|$ by choices $|X : x_1|$ and $|X : x_2|$ in order to improve on the preciseness of complete test frames with respect to the execution of the function rules of $\mathbb{U}$.

Example 3.9 (Composite Choice)

In the functional unit $\mathbb{U}_{\mathsf{MEAL}}$ of the system, the information captured in master flight schedules will be used to generate the corresponding daily meal schedules. During the

generation process, some information in master flight schedules can be overridden by the following exceptional schedules/records if they have been defined:

- *Exceptional flight schedules* (*EFS's*): They allow users to change the estimated time of departure of a flight on a particular date after the master flight schedule of this flight has been defined.

- *Exceptional crew configuration records* (*ECCR's*): They allow users to change the number of crewmembers in a flight on a particular date after the master flight schedule of this flight has been defined. Such records are necessary since AIR-FOOD needs to prepare meals for the crews as well as for passengers.

In view of the possible definition of EFS's and ECCR's, any master flight schedule will fall into one of the following situations:

(a) It is associated with an EFS but not an ECCR.

(b) It is associated with an ECCR but not an EFS.

(c) It is associated with an EFS and an ECCR.

(d) It is not associated with any EFS or ECCR.

In order to generate sufficient complete test frames to cover the above four situations, one approach is to identify [Existence of EFS] and [Existence of ECCR] as two categories. The former has |Existence of EFS: Yes| and |Existence of EFS: No| as associated choices, whereas the latter has |Existence of ECCR: Yes| and |Existence of ECCR: No| as associated choices. Thus, all of the above situations can be catered for by selecting one choice in [Existence of EFS] and one in [Existence of ECCR]. For example, the choices |Existence of EFS: Yes| and |Existence of ECCR: Yes| will cover situation (c).

In our empirical studies to be described later, some subjects have identified [Existence of Exceptional Schedules/Records] as one category instead of defining [Existence of EFS]

and [Existence of ECCR]. This new category has |Existence of Exceptional Schedules/Records: EFS and ECCR Do Not Exist| and |Existence of Exceptional Schedules/Records: Otherwise| as associated choices. As a result, |Existence of Exceptional Schedules/Records: Otherwise| applies to situations (a), (b), and (c) above, whereas |Existence of Exceptional Schedules/Records: EFS and ECCR Do Not Exist| applies to situation (d). Note that |Existence of Exceptional Schedules/Records: Otherwise| cannot be used to generate different complete test frames to cover situations (a), (b), and (c) separately.

Consider |Existence of Exceptional Schedules/Records: Otherwise|. It can be replaced by three non-overlapping choices, namely |Existence of Exceptional Schedules/Records: Only EFS Exists|, |Existence of Exceptional Schedules/Records: Only ECCR Exists|, and |Existence of Exceptional Schedules/Records: Both EFS and ECCR Exist|. We note the following:

(i) $(|\text{Existence of Exceptional Schedules/Records: Only EFS Exists}| \cup |\text{Existence of Exceptional Schedules/Records: Only ECCR Exists}|) \subset |\text{Existence of Exceptional Schedules/Records: Otherwise}|$.

(ii) Let

- *B* be a valid but incomplete test frame {|Weekly Departure Pattern: Daily|, |Type of Master Flight Schedule: Current|, |Number of Regular Meals in First-Class Cabin: > 0|, |Number of Special Meals in First-Class Cabin: $= 0$|, |Number of Regular Meals in Business-Class Cabin: > 0|, |Number of Special Meals in Business-Class Cabin: > 0|, |Number of Regular Meals in Economy-Class Cabin: > 0|, |Number of Special Meals in Economy-Class Cabin: > 0|, ... }.[3]

[3] To avoid lengthy discussions, we only list the choices in *B* that have been introduced in earlier examples.

- $B_1^c = (B \cup |\text{Existence of Exceptional Schedules/Records: Only EFS Exists}|)$ be a complete test frame.
- $B_2^c = (B \cup |\text{Existence of Exceptional Schedules/Records: Only ECCR Exists}|)$ be a complete test frame.

B_1^c and B_2^c are associated with different function rules because, according to the specification $\mathbb{S}_{\mathsf{MOS}}$, B_1^c and B_2^c correspond to two different sources of overriding information in master flight schedules during the generation of daily meal schedules. In other words, the choices |Existence of Exceptional Schedules/Records: Only EFS Exists| and |Existence of Exceptional Schedules/Records: Only ECCR Exists| are non-combinable.

Hence, |Existence of Exceptional Schedules/Records: Otherwise| is a composite choice and [Existence of Exceptional Schedules/Records] is a category with a composite choice. Obviously, the composite choice could be replaced by |Existence of Exceptional Schedules/Records: Only EFS Exists|, |Existence of Exceptional Schedules/Records: Only ECCR Exists|, and |Existence of Exceptional Schedules/Records: Both EFS and ECCR Exist|, with a view to improving the comprehensiveness of the resulting set of complete test frames. ■

Definition 3.10 (Problematic Choice)

A potential choice $|X : x|$ *in a category* $[X]$ *for a functional unit* $\mathbb{U}$ *is said to be* ***problematic*** *if at least one of the following criteria is satisfied:*

(a) $|X : x|$ *is an invalid choice.*

(b) $|X : x|$ *is one of the overlapping choices.*

(c) $|X : x|$ *is one of the combinable choices.*

(*d*) $|X : x|$ *is a composite choice.*

Definition 3.11 (Problematic Category)

A potential category $[X]$ *for a functional unit* $\mathbb{U}$ *is said to be* ***problematic*** *if at least one of the following criteria is satisfied:*

(*a*) $[X]$ *is an irrelevant category.*

(*b*) $[X]$ *is a category with missing choices.*

(*c*) $[X]$ *is a category with problematic choices.*

It should be noted that a problematic category may satisfy more than one criterion listed in Definitions 3.10 and 3.11. Consider the category [Number of Regular Meals in First-Class Cabin] in Example 3.7. Suppose this category is identified with three associated choices, namely |Number of Regular Meals in First-Class Cabin: < 0|, |Number of Regular Meals in First-Class Cabin: $= 0$|, and |Number of Regular Meals in First-Class Cabin: ≥ 0|. As explained in Example 3.7, |Number of Regular Meals in First-Class Cabin: $= 0$| and |Number of Regular Meals in First-Class Cabin: ≥ 0| are overlapping choices. Furthermore, |Number of Regular Meals in First-Class Cabin: < 0| is an invalid choice because, for every element $v \in$ |Number of Regular Meals in First-Class Cabin: $<$ 0|, $v \notin E$([Number of Regular Meals in First-Class Cabin]). Hence, [Number of Regular Meals in First-Class Cabin] is a category with overlapping choices as well as an invalid choice.

Definition 3.12 (Problematic Set of Potential Categories and Potential Choices)

Given a set PC of potential categories and their associated potential choices for a functional unit $\mathbb{U}$*, it is said to be* ***problematic*** *if at least one of the following criteria is satisfied:*

(*a*) *PC has missing categories.*

(*b*) *PC has problematic categories.*

3.1.4 Experimental Setting

We have conducted three empirical studies to find out the common mistakes made by testers during an ad hoc, impromptu identification of categories and choices from informal specifications. The respective specifications used in the three studies are denoted by $\mathbb{S}_{\mathsf{TRADE}}$, $\mathbb{S}_{\mathsf{PURCHASE}}$, and $\mathbb{S}_{\mathsf{MOS}}$.

The first specification $\mathbb{S}_{\mathsf{TRADE}}$ is related to the credit sales of goods by a wholesaler to retail customers, and is mainly in the form of narrative descriptions. The main function of the system is to decide whether credit sales should be approved for individual retail customers. Such decision considers several issues, including the credit status and the credit limit of the customer, the billing amount of the transactions, and any special management approval by the wholesaler.

The second specification $\mathbb{S}_{\mathsf{PURCHASE}}$ is related to the purchase of goods using credit cards issued by an international bank. Each credit card is associated with several attributes such as status (diamond, gold, or classic), type (corporate or personal), and credit limit (different card statuses will have different credit limits). The main functions of the system are to decide whether a purchase using a credit card should be approved, and to calculate the number of reward points to be granted for an approved purchase. The number of reward points further determines the type of benefit (such as free airline tickets and shopping vouchers) that the customer is entitled to. Similar to $\mathbb{S}_{\mathsf{TRADE}}$, $\mathbb{S}_{\mathsf{PURCHASE}}$ is mainly written in narrative descriptions.

Finally, the third specification is $\mathbb{S}_{\mathsf{MOS}}$ described in Section 3.1.3. In order to protect the identity of AIR-FOOD and to make $\mathbb{S}_{\mathsf{MOS}}$ suitable for our study, we have slightly amended the original specification before commencing the studies. The majority of the contents of the original $\mathbb{S}_{\mathsf{MOS}}$, however, has remained unchanged.

For empirical studies 1 and 2, the subjects are 48 final-year undergraduates in the computer science and software engineering programs at The University of Melbourne. On the other hand, for empirical study 3, the subjects are a mix of 44 undergraduates and postgraduates in the computer science, software engineering, and information technology programs in Swinburne University of Technology. In both universities, a one-hour lecture was devoted to the introduction of CTM. Teaching of the method was supported by related literature such as [14, 20, 21, 39, 44, 72]. The lecture was reinforced by a one-hour tutorial with various examples, including the program COUNT in Example 2.1. The subjects in both universities were being taught by the same instructor using the same teaching materials.

In study 3, after the subjects have learned CTM, we asked them to carry out the following tasks:

(a) Decompose the specification $\mathbb{S}_{\mathsf{MOS}}$ into several functional units that can be tested independently, including the unit $\mathbb{U}_{\mathsf{MEAL}}$ directly related to the generation of daily meal schedules. Such decomposition is needed because the system MOS contains numerous modules and is fairly complex in logic.

(b) Identify from the functional unit $\mathbb{U}_{\mathsf{MEAL}}$ a set of categories and their associated choices. [4]

(c) For every identified category or choice, state the reason of its identification.

In studies 1 and 2, on the other hand, we asked the subjects to treat each of the specifications $\mathbb{S}_{\mathsf{TRADE}}$ and $\mathbb{S}_{\mathsf{PURCHASE}}$ denoted by $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$, respectively, as a single functional unit. This is because $\mathbb{S}_{\mathsf{TRADE}}$ and $\mathbb{S}_{\mathsf{PURCHASE}}$ are less complex than $\mathbb{S}_{\mathsf{MOS}}$ and can therefore be tested in their entirety. For each of $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$, the subjects are asked to identify a set of categories and their associated choices, and to

[4] We choose $\mathbb{U}_{\mathsf{MEAL}}$ for our study because generating daily meal schedules is a core function of MOS.

provide justifications similarly to tasks (b) and (c) above. For all the three studies, the subjects were asked to complete tasks (a) to (c) in about three weeks.

3.1.5 Findings, Discussions, and Recommendations

An initial examination of the potential categories and potential choices identified by the subjects for the three functional units $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$ reveals the following:

(a) Table 3.2 shows the statistics of the potential categories and potential choices identified for each functional unit. Every subject is asked to identify one and only one set of potential categories and their associated potential choices. We shall use *PC*'s to denote these sets. Thus, the number of *PC*'s is equal to the number of subjects.

We observe that the mean numbers of potential categories and choices increase with the complexity of the functional units, with $\mathbb{U}_{\mathsf{TRADE}}$ being the least complex and $\mathbb{U}_{\mathsf{MEAL}}$ the most complex. A natural reason for this phenomenon is that software systems associated with complex specifications often contain many aspects for testing, thus contributing to more potential categories and choices to be identified.

We also note that the numbers of potential categories and choices vary substantially among the subjects, as evidenced by the large ranges and standard derivations. The latter observation indicates that the quality of *PC*'s, as identified by the subjects in an impromptu manner, also varies significantly. Thus, this observation clearly indicates that systematic identification techniques for identifying categories and choices from informal specifications are badly needed.

(b) The mean numbers of potential choices in each potential category are 2.2 ($= \frac{579}{265}$), 2.4 ($= \frac{1138}{475}$), and 2.4 ($= \frac{1488}{615}$) for $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$, respectively. Hence, the number of potential choices in each potential category is fairly small,

Functional Unit	Number of Sets of Potential Categories and Potential Choices	Number of Potential Categories (Choices)			
		Total	Mean *	Range	Standard Derivation
$\mathbb{U}_{\text{TRADE}}$	48	265 (579)	5.5 (12.1)	4–9 (10–20)	0.9 (1.5)
$\mathbb{U}_{\text{PURCHASE}}$	48	475 (1 138)	9.9 (23.7)	6–14 (15–35)	2.0 (4.4)
$\mathbb{U}_{\text{MEAL}}$	44	615 (1 488)	14.0 (33.8)	4–40 (10–83)	7.8 (16.7)

($\star$) For each subject

Table 3.2: Statistics of Potential Categories and Potential Choices Identified for Each Functional Unit

even though all the potential choices in a potential category should cover all the input elements relevant to that category. The main reason for a small number of potential choices in each potential category is that a potential choice consists of a set of values. For example, the choice |Number of Regular Meals in First-Class Cabin: > 0| in Example 3.7 consists of all positive integers.

(c) Table 3.3 shows the statistics of missing and problematic categories for each functional unit. Similar to the mean numbers of potential categories and choices as reported in (a), the mean numbers of missing categories and problematic categories also increase with the complexity of the functional units. Note the high percentages of *PC*'s with missing categories and/or problematic categories in all the three functional units. Here, we have two observations:

- The occurrence of missing categories in *PC*'s would mean that the *PC*'s are not comprehensive, since they do not contain sufficient categories (and associated choices) to generate enough complete test frames for testing the function rules of each functional unit.
- The occurrence of problematic categories in *PC*'s would mean that the *PC*'s are not effective, since these problematic categories will cause the generation

Functional Unit	Number of *PC*'s	Number of Missing Categories	Number (%) of *PC*'s with Missing Categories	Average Number of Missing Categories in Each *PC*	Number of Problematic Categories	Number (%) of *PC*'s with Problematic Categories	Average Number of Problematic Categories in Each *PC*
$\mathbb{U}_{\mathsf{TRADE}}$	48	1	1 (2.1%)	0.02	43	42 (87.5%)	0.90
$\mathbb{U}_{\mathsf{PURCHASE}}$	48	33	23 (47.9%)	0.69	79	46 (95.8%)	1.65
$\mathbb{U}_{\mathsf{MEAL}}$	44	158	44 (100.0%)	3.59	158	41 (93.2%)	3.59

PC = Set of potential categories and choices

Table 3.3: Statistics of Missing and Problematic Categories for Each Functional Unit

of incomplete test frames.

Let us further analyze the problematic categories identified by the subjects. Consider Table 3.4 that shows the numbers and percentages of different types of problematic category, and Table 3.5 that shows the numbers and percentages of *PC*'s containing different types of problematic category. A closer examination reveals that 42 (87.5%), 46 (95.8%), and 41 (93.2%) of the *PC*'s for $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$, respectively, contain at least one problematic category.

Refer to the second columns from the left in Tables 3.4 and 3.5. Out of the 615 potential categories identified for $\mathbb{U}_{\mathsf{MEAL}}$, we find that 123 (20.0%) are irrelevant with respect to $\mathbb{U}_{\mathsf{MEAL}}$. These irrelevant categories occur in 33 (75.0%) *PC*'s, and are identified with regard to factors related to the execution of functional units other than $\mathbb{U}_{\mathsf{MEAL}}$ in $\mathbb{S}_{\mathsf{MOS}}$. The occurrence of irrelevant categories clearly indicates that the logical decomposition of a specification into several independent functional units is not a trivial task that can be performed effectively without the help of systematic methodologies. For $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$, no irrelevant category is detected. The main reason for the absence of irrelevant categories in this case is that, for each specification $\mathbb{S}_{\mathsf{TRADE}}$ and $\mathbb{S}_{\mathsf{PURCHASE}}$, the subjects were asked to treat it as one single functional unit and, hence, no decomposition

Functional Unit	Number (%) of					
			Categories with Problematic Choices			
	Irrelevant Categories	Categories with Missing Choices	Categories with Invalid Choices	Categories with Overlapping Choices	Categories with Combinable Choices	Categories with Composite Choices
$\mathbb{U}_{\text{TRADE}}$	0 (0.0%)	3 (1.1%)	0 (0.0%)	6 (2.3%)	0 (0.0%)	34 (12.8%)
$\mathbb{U}_{\text{PURCHASE}}$	0 (0.0%)	9 (1.9%)	2 (0.4%)	26 (5.5%)	0 (0.0%)	42 (8.8%)
$\mathbb{U}_{\text{MEAL}}$	123 (20.0%)	12 (2.0%)	14 (2.3%)	4 (0.7%)	5 (0.8%)	4 (0.7%)

Table 3.4: Numbers and Percentages of Different Types of Problematic Category

is required. Thus, it is impossible to identify irrelevant categories for factors outside $\mathbb{U}_{\text{TRADE}}$ and $\mathbb{U}_{\text{PURCHASE}}$.

Functional Unit	Number (%) of Sets of Potential Categories and Potential Choices (*PC*'s) Containing					
			Categories with Problematic Choices			
	Irrelevant Categories	Categories with Missing Choices	Categories with Invalid Choices	Categories with Overlapping Choices	Categories with Combinable Choices	Categories with Composite Choices
$\mathbb{U}_{\text{TRADE}}$	0 (0.0%)	3 (6.3%)	0 (0.0%)	6 (12.5%)	0 (0.0%)	34 (70.8%)
$\mathbb{U}_{\text{PURCHASE}}$	0 (0.0%)	8 (16.7%)	2 (4.2%)	25 (52.1%)	0 (0.0%)	31 (64.6%)
$\mathbb{U}_{\text{MEAL}}$	33 (75.0%)	7 (15.9%)	11 (25.0%)	4 (9.1%)	3 (6.8%)	4 (9.1%)

Table 3.5: Numbers and Percentages of *PC*'s Containing Different Types of Problematic Category

If we compare Tables 3.4 and 3.5, we observe that:

(i) The relative frequency distributions of different types of problematic category are fairly similar across all three studies.

(ii) Categories with composite choices are the most common. On the other hand, categories with combinable choices are the least common.

The above observations together clearly suggest that an impromptu identification approach is highly ineffective. Without doubt, there is a strong need for systematic methods for identifying categories and choices from informal specifications.

Based on the above observations and discussions, we formulate the following checklist to help testers detect the existence of missing categories, problematic categories, and problematic choices.

A Checklist for Detecting Missing Categories, Problematic Categories, and Problematic Choices:

(1) Due care should be taken when decomposing an informal specification into functional units. In particular, check whether there exist any irrelevant categories identified for non-influencing factors of the selected functional unit $\mathbb{U}$.

(2) Check whether there exists any influencing factor K of the selected $\mathbb{U}$ but is not associated with any potential category. If this happens, there will be missing categories that we fail to identify.

(3) For every potential choice $|X : x|$, check whether $TF_{\mathbb{U}}(|X : x|) = \emptyset$. If so, $|X : x|$ is an invalid choice.

(4) For every category $[X]$, check whether the union of all its choices identified so far covers all the input values relevant to $[X]$. If not, $[X]$ contains missing choices yet to be identified.

(5) For any non-empty set of choices in every category, determine whether these choices are overlapping by checking the existence of common elements.

(6) When identifying potential categories and potential choices, consider also:

(a) the constraints among potential choices in the formation of complete test frames; and

(b) the function rules involving these choices.

This will help detect the occurrence of combinable choices and composite choices. The detection of categories with composite choices is particularly important, since our studies have indicated that they are the most common among various types of problematic category.

We cannot guarantee that a process based on the above checklist will necessarily detect all possible missing categories, problematic categories, and problematic choices. According to our analysis and empirical studies, however, such unwarranted cases can be greatly reduced. (The effectiveness of this checklist will be evaluated in our second round of empirical studies to be described in Section 3.2.4 later.)

3.1.6 Threads to Validity

Our first round of empirical studies has the following limitations due to the respective settings:

(a) The subjects of studies 1 and 2 were all undergraduates in The University of Melbourne, whereas those of study 3 were a mix of undergraduates and post-graduates in Swinburne University of Technology. Because of the differences in universities and the subjects' calibers, readers are recommended not to compare the mistakes made by the subjects among these studies, which is not the main objective of Section 3.1. Instead, this section aims to identify the types of common mistake

made by the subjects when identifying categories and choices in the absence of a systematic process.

(b) For studies 1 and 2, the specifications $\mathbb{S}_{\mathsf{TRADE}}$ and $\mathbb{S}_{\mathsf{PURCHASE}}$ were given to the subjects as one single assignment. Hence, we do not know which specification the subjects worked on first, although we think that the majority of the subjects should have started with $\mathbb{S}_{\mathsf{TRADE}}$ because it is less complex. One can argue that the subjects may gain in experience after doing the first case. However, this effect should be minimal, if any, because the subjects were advised of their errors only after they have completed all the tasks for both specifications.

(c) Obviously, the results of our studies might differ if the subjects were real software testers with substantial commercial software development experience instead of being undergraduates and postgraduates. If such were their backgrounds, they might make fewer mistakes in an impromptu identification of categories and choices.[5] We observe, however, that even in study 3, where most of the postgraduates in Swinburne University of Technology had real-life IT working experience, problematic categories and choices were identified. This observation supports our earlier argument that an impromptu identification approach cannot assure the quality of the resulting categories and choices, regardless of the caliber of the subjects.

3.1.7 Summary

We have analyzed and discussed the common mistakes when software testers use an impromptu approach to identifying categories and choices from informal specifications. We have conducted three empirical studies via different specifications and testers. To

[5] In our second round of empirical studies described in Section 3.2, we shall investigate the reduction of mistakes from inexperienced to experienced testers for the same functional unit.

facilitate the analysis of our empirical results, we have formally defined missing categories and various types of problematic category and choice. We have also discussed plausible reasons for the identification of such categories and choices. Our results confirm that missing categories, problematic categories, and problematic choices are likely to occur when the identification of categories and choices is performed in an impromptu manner. There is, therefore, a great demand for the introduction of systematic identification techniques to improve on the quality of the process and eventually the quality of the resulting test cases.

The contributions of our empirical studies are threefold. First, by defining missing categories and the various types of problematic category and choice and highlighting them to inexperienced users, testers will be alerted to avoid them. Secondly, the knowledge of such categories and choices and plausible reasons for their identification give researchers and practitioners an insight into the development of systematic identification methods. Thirdly, the effectiveness of any developed identification method can be measured in terms of its ability to screen out missing categories, problematic categories, and problematic choices.

Based on the results of our empirical studies, we have developed an identification checklist to help testers detect the existence of missing categories, problematic categories, and problematic choices when the identification process is performed in an impromptu manner.

3.2 Second Round of Studies

3.2.1 Purposes of Studies

At the time of our first round of studies [15] described in Section 3.1, the recruited subjects were undergraduates or postgraduates in the areas of computer science, software

engineering, or information technology, who did not have much commercial working experience in software development or testing. We conjecture that the type and the amount of mistakes made by the subjects in an ad hoc, impromptu identification approach may vary with their past testing experience.

To test our conjecture, we have conducted several comparative studies using the three functional units $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$. This second round of studies represents the follow-up work of our first round of studies described in Section 3.1. The current round of studies serves the following two purposes:

(a) To verify how the type and number of mistakes made by the testers in an impromptu identification approach vary with their working experience in software development and testing.

(b) To determine, after discussing the mistakes with the testers and providing them with our checklist (presented in Section 3.1.5) as a simple guideline for detecting problematic categories and choices, how many mistakes can be avoided in the next identification exercise.

Later, in Section 3.2.5, we shall also discuss a further study targeted to determine, from the opinions of the subjects, which specification components are useful for category and choice identification. Furthermore, we shall report a suggestion made by the subjects about how the effectiveness of the identification exercise could be improved.

3.2.2 Experimental Setting

Our studies use the *same* set of three functional units, namely $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$ mentioned in Section 3.1. The specifications from which these three functional units are decomposed are written primarily in an informal manner.

We have recruited 16 software practitioners as the subjects of our studies. They will be

referred to as Subjects 1, 2, . . . , 16. In general, their IT qualifications are undergraduate or postgraduate degrees in information technology, information systems, business computing, computer science, computing studies, and computer engineering. [6] In addition, the subjects have 8 to 20 years of commercial experience in software development and testing, with a mean of 11.9 years of experience. Thus, they are classified as *experienced* testers. On the other hand, the subjects in our previous studies were undergraduates or postgraduates with little or no working experience in software development or testing. Hence, they are classified as *inexperienced* testers.

Before commencing our studies, we have prepared all the subjects by giving them a one-hour introduction of CHOC'LATE and CTM, supported by related literature [14, 20, 21, 39, 44, 72]. The introduction is followed by a one-hour discussion in which some examples of CHOC'LATE and CTM are used to reinforce the subjects' understanding of these techniques. As far as possible, we have made the preparation exercise the same as in the first round of studies [15] in terms of the instructor, teaching method, and teaching materials. This arrangement, together with the use of the same set of functional units, allows us to compare the results with those reported in Section 3.1 in a meaningful way.

3.2.3 Study 1: Effectiveness of Tester Experience

Objective and Steps

The main objective of study 1 is to investigate how the types and amounts of mistakes made in an impromptu identification approach vary between inexperienced and experienced testers. After the subjects have learned CHOC'LATE and CTM, we ask each of them to do an identification exercise according to the following scheme:

[6] Some of these subjects also have other non-IT academic qualifications such as MBA degrees.

(a) **Subjects 1 to 8:** For each of $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$, identify from it a set of categories and their associated choices in an impromptu manner. Furthermore, for each identified category and choice, the reason of its identification has to be stated. We have asked the eight subjects to work on $\mathbb{U}_{\mathsf{TRADE}}$ before $\mathbb{U}_{\mathsf{PURCHASE}}$.

(b) **Subjects 9 to 16:** Repeat (a) above for $\mathbb{U}_{\mathsf{MEAL}}$ instead of $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$.

Because of the above scheme, there are eight experienced subjects involved in each functional unit. The rationale of breaking the subjects into two groups is to reduce biases in the later investigation on how they perform in study 2. Whilst the subjects are arbitrarily assigned in groups, we have kept the average years of commercial experience in software development and testing of each group of subjects as similar as possible.

In this identification exercise, each subject is asked to identify one set (denoted by *PC*) of potential categories and their associated potential choices. Therefore, the number of *PC*'s is equal to the number of subjects. This is also the case for our first round of studies [15] in Section 3.1. When analyzing the study results, readers may recall the following:

- In our first round of studies, the number of subjects for $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$ were 48, 48, and 44, respectively.
- Unless otherwise stated, relevant categories and valid choices are simply referred to as categories and choices, respectively.

Findings and Discussions

Potential Categories and Choices

Consider Table 3.6, which shows the statistics of potential categories and choices identified for each functional unit. Note that this table (as well as Tables 3.7 to 3.16)

By Inexperienced Testers / Experienced Testers:					
Functional Unit	**Numbers of *PC*'s**	**Numbers of Potential Categories (Choices)**			
		Totals	**Means** *	**Ranges**	**Standard Derivations**
$\mathbb{U}_{\mathsf{TRADE}}$	48 / 8	265 (579) / 54 (124)	5.5 (12.1) / 6.8 (15.5)	5 (10) / 2 (8)	0.9 (1.5) / 0.7 (2.5)
$\mathbb{U}_{\mathsf{PURCHASE}}$	48 / 8	475 (1 138) / 101 (278)	9.9 (23.7) / 12.6 (34.8)	8 (20) / 4 (11)	2.0 (4.4) / 1.3 (3.1)
$\mathbb{U}_{\mathsf{MEAL}}$	44 / 8	615 (1 488) / 134 (299)	14.0 (33.8) / 16.8 (37.4)	36 (73) / 3 (10)	7.8 (16.7) / 1.5 (3.7)
Averages			**9.8 (23.2) / 12.0 (29.2)**	**16.3 (34.3) / 3.0 (9.7)**	**3.6 (7.5) / 1.2 (3.1)**

PC = Set of potential categories and choices

($\star$) For each subject

Table 3.6: Statistics of Potential Categories and Choices Identified by Inexperienced and Experienced Testers

shows two sets of results separated by slashes "/". The first set corresponds to our first round of studies [15] in Section 3.1 involving inexperienced testers, while the second set corresponds to our present studies involving experienced testers. Also, data enclosed in brackets in Table 3.6 correspond to potential choices, while those not enclosed in brackets correspond to potential categories. We have the following observations from Table 3.6:

Observation 1: The mean numbers of potential categories and choices identified by both inexperienced and experienced testers increase with the complexity of the functional units, with the minimum numbers attached to the least complex $\mathbb{U}_{\mathsf{TRADE}}$ and the maximum numbers attached the most complex $\mathbb{U}_{\mathsf{MEAL}}$. A natural reason for this phenomenon is that software systems associated with complex specifications often contain many aspects for testing, thus contributing to more potential categories and choices to be identified.

Observation 2: The numbers of potential categories and choices vary substantially among the subjects, as evidenced by the large ranges and standard derivations. This, in turn, indicates that the quality of *PC*'s, as identified by the subjects in an

impromptu manner, also varies significantly. Thus, observation 2 clearly indicates that systematic identification techniques for identifying categories and choices from informal specifications are badly needed.

Observation 3: The mean numbers of potential categories and choices identified by experienced testers are about 22% and 26% larger than those by inexperienced testers, respectively. Readers should note, however, that this observation *alone* does not necessarily indicate that the *PC*'s identified by experienced testers are more comprehensive than those by inexperienced testers. The comprehensiveness of a *PC* depends on the number of *non-problematic* categories it contains. Even when a large number of potential categories are identified, some of them may be problematic and, hence, not useful for test case generation.

Observation 4: Among the three functional units, the ranges and standard derivations are generally much larger for inexperienced testers than for experienced testers. Thus, this observation suggests that the variation in the sizes of *PC*'s is much larger for inexperienced testers than for experienced testers. A plausible reason is that, by virtue of their experience, the subjects in the second group are able to identify *PC*'s with more consistent qualities. The observation clearly shows that experience in software development and testing is vital to the identification process.

We shall, however, remind readers to interpret this observation carefully. Our argument that experienced testers are able to identify *PC*'s with more *consistent* qualities is put forward in a relative sense, when compared to inexperienced testers. It does not mean that the *PC*'s identified by experienced testers are necessarily of *good* quality, as indicated by our later observations that even experienced testers have numerous mistakes in the identification process. These later observations also suggest that, although practice and experience in software development and

By Inexperienced Testers / Experienced Testers:			
Functional Unit	**Total Numbers of Missing Categories**	**Mean Numbers of Missing Categories in Each *PC***	**Percentages of Mean Numbers of Missing Categories in Each *PC* in Relation to Mean Numbers of Potential Categories in Each *PC***
$\mathbb{U}_{\mathsf{TRADE}}$	1 / 5	0.02 / 0.63	0.38% / 9.26%
$\mathbb{U}_{\mathsf{PURCHASE}}$	33 / 5	0.69 / 0.63	6.95% / 4.95%
$\mathbb{U}_{\mathsf{MEAL}}$	158 / 11	3.59 / 1.38	25.69% / 8.21%
Averages		**1.43 / 0.88**	**11.01% / 7.47%**

PC = Set of potential categories and choices

Table 3.7: Total Numbers, Mean Numbers, and Mean Percentages of Missing Categories by Inexperienced and Experienced Testers

testing do contribute to the identification of categories and choices, such practice and experience cannot eliminate the need for systematic techniques.

Missing Categories

We turn our attention to Table 3.7, which shows the statistics of missing categories for each functional unit. We observe the following from the table:

Observation 5: Similarly to observation 1, the mean numbers of missing categories in each *PC* generally increase with the complexity of the functional units for both groups of subjects. A plausible reason for this phenomenon is that, in an impromptu identification approach, the chance of omitting relevant categories is higher for more complex functional units.

Observation 6: In addition, Table 3.7 shows that, when considering all the three functional units together, the mean number of missing categories in each *PC* is signif-

By Inexperienced Testers / Experienced Testers:	
Functional Unit	**Percentage Increase in Mean Numbers of Missing Categories in Each *PC***
From $\mathbb{U}_{\mathsf{TRADE}}$ to $\mathbb{U}_{\mathsf{PURCHASE}}$	3 350% / 0%
From $\mathbb{U}_{\mathsf{PURCHASE}}$ to $\mathbb{U}_{\mathsf{MEAL}}$	420% / 119%

PC = Set of potential categories and choices

Table 3.8: Percentage Increase in Mean Numbers of Missing Categories by Inexperienced and Experienced Testers

icantly larger (about 63%) for inexperienced testers than for experienced testers. This suggests that experience in software development and testing does help testers a great deal in avoiding the omission of relevant categories in the absence of a systematic identification technique.

Following on observation 5 above, Tables 3.8 and 3.9 show the percentage increase in the mean numbers of missing categories in each *PC* and the percentage increase in the mean numbers of potential categories in each *PC*, respectively, when the testers work on the next more complex functional unit. (Readers may recall that $\mathbb{U}_{\mathsf{TRADE}}$ is the least complex and $\mathbb{U}_{\mathsf{MEAL}}$ is the most complex.)

Observation 7: When a functional unit becomes more complex, the increase in the mean numbers of missing categories in each *PC* is much more significant for inexperienced testers than experienced testers. Thus, Table 3.8 allows us to draw a conclusion similar to that in observation 6, that is, experience in software development and testing, to some extent, helps testers reduce the occurrence of missing categories in an impromptu identification approach.

By Inexperienced Testers / Experienced Testers:	
Functional Unit	**Percentage Increase in Mean Numbers of Potential Categories in Each *PC***
From $\mathbb{U}_{\mathsf{TRADE}}$ to $\mathbb{U}_{\mathsf{PURCHASE}}$	80% / 85%
From $\mathbb{U}_{\mathsf{PURCHASE}}$ to $\mathbb{U}_{\mathsf{MEAL}}$	41% / 33%

PC = Set of potential categories and choices

Table 3.9: Percentage Increase in Mean Numbers of Potential Categories Identified by Inexperienced and Experienced Testers

Observation 8: Let us compare Tables 3.8 and 3.9 for the percentage increase in the mean numbers of missing and potential categories in each *PC*. Consider the data for inexperienced testers in both tables first. As the functional units become more complex, the percentage increase in the mean numbers of missing categories in each *PC* (3 350% from $\mathbb{U}_{\mathsf{TRADE}}$ to $\mathbb{U}_{\mathsf{PURCHASE}}$ and 420% from $\mathbb{U}_{\mathsf{PURCHASE}}$ to $\mathbb{U}_{\mathsf{MEAL}}$) are much larger than the percentage increase in the mean numbers of potential categories in each *PC* (80% from $\mathbb{U}_{\mathsf{TRADE}}$ to $\mathbb{U}_{\mathsf{PURCHASE}}$ and 41% from $\mathbb{U}_{\mathsf{PURCHASE}}$ to $\mathbb{U}_{\mathsf{MEAL}}$). The phenomenon of consistent percentage increase for missing categories, however, is not applicable to experienced testers when the functional units become more complex. Likewise, in the rightmost column of Table 3.7, the percentages of the mean numbers of missing categories in each *PC* in relation to the mean numbers of potential categories in each *PC* increase with the complexity of the functional units for inexperienced testers but not experienced ones.

Problematic and Non-Problematic Categories

Observation 9: After examining the missing categories, we then analyze the problematic categories and choices identified by experienced testers for the three functional

By Inexperienced Testers / Experienced Testers:						
Functional Unit	Problematic Categories			Non-Problematic Categories		
	Total Numbers	Mean Numbers in Each *PC*	Mean Percentages in All Potential Categories	Total Numbers	Mean Numbers in Each *PC*	Mean Percentages in All Potential Categories
$\mathbb{U}_{\text{TRADE}}$	43 / 5	0.90 / 0.63	16.23% / 9.26%	222 / 49	4.63 / 6.13	83.77% / 90.74%
$\mathbb{U}_{\text{PURCHASE}}$	79 / 12	1.65 / 1.50	16.63% / 11.88%	396 / 89	8.25 / 11.13	83.37% / 88.12%
$\mathbb{U}_{\text{MEAL}}$	158 / 28	3.59 / 3.50	25.69% / 20.90%	457 / 106	10.39 / 13.25	74.31% / 79.10%
Averages		**2.04 / 1.88**	**19.52% / 14.01%**		**7.75 / 10.17**	**80.48% / 85.99%**

PC = Set of potential categories and choices

Table 3.10: Total Numbers, Mean Numbers, and Mean Percentages of Problematic and Non-Problematic Categories Identified by Inexperienced and Experienced Testers

units. It turns out that all these categories and choices can be classified into the problematic types as defined in Section 3.1.3. In other words, no new type of problematic category and choice is found. This suggests that the list of problematic categories and choices in Section 3.1.3 is fairly comprehensive.

Readers may recall from Section 3.1.3 that a potential category $[X]$ for a functional unit is said to be *problematic* if $[X]$ is an irrelevant category, a relevant category with missing choices, or a relevant category with problematic choices. Otherwise, $[X]$ is said to be *non-problematic*. Table 3.10 shows the statistics of problematic and non-problematic categories for each functional unit. Note that only non-problematic categories and their associated choices are useful for test case generation. Let us first focus on problematic categories.

Observation 10: Similarly to observations 1 and 5, the mean numbers of problematic categories in each *PC* and the mean percentages of problematic categories in all potential categories increase with the complexity of the functional units for both groups of subjects. The reason for such an observation is as follows: Recall that observation 1 states that the mean numbers of potential categories identified by each

subject (that is, the mean numbers of potential categories in each *PC*) increase with the complexity of the functional units. In general, more potential categories to be identified would increase the chance of the occurrence of problematic categories. Furthermore, observations 5 and 10 together show clearly that the chance to make mistakes (in terms of missing and problematic categories) in an impromptu identification approach increases with the complexity of the functional units.

Observation 11: Across the three functional units, the mean numbers of problematic categories in each *PC* and the mean percentages of problematic categories in all the potential categories identified by inexperienced testers are consistently larger (by an average of about 9% and 39%, respectively) than those by experienced testers. Thus, experience in software development and testing not only helps testers reduce the chances of omitting relevant categories (see observation 6), but also reduce the chances of identifying problematic categories.

Observation 12: Consider the reduction in the mean numbers of problematic categories in each *PC* for a given functional unit from inexperienced to experienced testers. These reductions are 0.27, 0.15, and 0.09 for $\mathbb{U}_{\mathsf{TRADE}}$, $\mathbb{U}_{\mathsf{PURCHASE}}$, and $\mathbb{U}_{\mathsf{MEAL}}$, respectively. We note that the reductions decrease with the complexity of the functional unit. Thus, although observation 11 finds that testing experience helps reduce the number of problematic categories, this advantage diminishes as the functional units become more complex. In other words, the need for a systematic identification technique for categories and choices is higher for more complex functional units.

Note that observation 12 focuses on the reduction of problematic categories from inexperienced to experienced testers for the *same* functional unit. Let us look at the reduction of problematic categories in another dimension. Table 3.11 shows the percentage increase in the mean numbers of problematic categories in each *PC* and the mean percentages of

By Inexperienced Testers / Experienced Testers:		
Functional Unit	**Percentage Increase in Mean Numbers of Problematic Categories in Each *PC***	**Percentage Increase in Mean Percentages of Problematic Categories in All Potential Categories**
From $\mathbb{U}_{\text{TRADE}}$ to $\mathbb{U}_{\text{PURCHASE}}$	83% / 138%	2% / 28%
From $\mathbb{U}_{\text{PURCHASE}}$ to $\mathbb{U}_{\text{MEAL}}$	118% / 133%	54% / 76%

PC = Set of potential categories and choices

Table 3.11: Percentage Increase in Mean Numbers/Percentages of Problematic Categories Identified by Inexperienced and Experienced Testers

problematic categories in all the potential categories identified by both groups of subjects when they work on the *next* more complex functional unit. From the table, we have observation 13 as follows:

Observation 13: Readers may recall from observation 7 that the increase in the mean numbers of missing categories in each *PC* is much more significant for inexperienced testers than experienced testers when the functional units become more complex. In contrast to observation 7, Table 3.11 shows that the percentage increase in the mean numbers of problematic categories in each *PC* and the mean percentages of problematic categories in all potential categories are larger for experienced testers than inexperienced testers when the functional units become more complex. This observation provides further support to our argument in observation 12 that the contribution of testing experience to the reduction of problematic categories becomes less for more complex functional units.

Now, let us turn to non-problematic categories.

Observation 14: Table 3.10 shows that, for both inexperienced and experienced testers, the mean numbers of non-problematic categories increase with the complexity of the functional units. This is consistent with the trend in observation 1 for potential categories.

Observation 15: Let us compare Tables 3.11 and 3.12 about the percentage increase in the mean numbers of problematic and non-problematic categories in each *PC*. For both groups of subjects, as the functional units become more complex, the percentage increase in the mean numbers of problematic categories in each *PC* (such as 83% from $\mathbb{U}_{\mathsf{TRADE}}$ to $\mathbb{U}_{\mathsf{PURCHASE}}$ for inexperienced testers) are larger than the percentage increase in the mean numbers of non-problematic categories in each *PC* (such as 78% from $\mathbb{U}_{\mathsf{TRADE}}$ to $\mathbb{U}_{\mathsf{PURCHASE}}$ for inexperienced testers). It appears, therefore, that although both inexperienced and experienced testers can identify more non-problematic categories when the functional units become more complex (see observation 14), this advantage is not sufficient to offset the increase in problematic categories at the same time.

Observation 15 also supports the following observation 16:

Observation 16: Following on to observations 14 and 15, Table 3.10 shows that the mean percentages of non-problematic categories in all the potential categories identified by both groups of subjects decrease with the complexity of the functional units. Readers may recall from observation 10 that the mean percentages of problematic categories in all the potential categories identified by both groups of subjects increase with the complexity of the functional units. In other words, given a potential category $[X]$ identified by either inexperienced or experienced testers, the

By Inexperienced Testers / Experienced Testers:		
Functional Unit	**Percentage Increase in Mean Numbers of Non-Problematic Categories in Each *PC***	**Percentage Decrease in Mean Percentages of Non-Problematic Categories in All Potential Categories**
From $\mathbb{U}_{\mathsf{TRADE}}$ to $\mathbb{U}_{\mathsf{PURCHASE}}$	78% / 82%	0.5% / 2.9%
From $\mathbb{U}_{\mathsf{PURCHASE}}$ to $\mathbb{U}_{\mathsf{MEAL}}$	26% / 19%	10.9% / 10.2%

PC = Set of potential categories and choices

Table 3.12: Percentage Increase/Decrease in Mean Numbers/Percentages of Non-Problematic Categories Identified by Inexperienced and Experienced Testers

chance of $[X]$ being a problematic category is higher for a more complex functional unit.

We also postulate from observation 11 that, given a potential category $[X]$ identified by experienced testers, the chance of $[X]$ being non-problematic (and, hence, useful for testing) should be higher than by inexperienced testers. This hypothesis is supported by observation 17 below:

Observation 17: In Table 3.10, the mean number of non-problematic categories in each *PC* and the mean percentage of non-problematic categories in all the potential categories identified by experienced testers are consistently larger (by an average of about 31% and 7%, respectively) than that by inexperienced testers across the three functional units. Once again, experience in software development and testing has a positive effect on the identification of non-problematic categories.

Let us take a closer examination of the statistics of the different types of problematic category. Table 3.13 first shows the total numbers of these problematic types identi-

By Inexperienced Testers / Experienced Testers:						
	Total Numbers of Different Types of Problematic Category					
Functional Unit	**Irrelevant Categories**	**Categories with Missing Choices**	**Categories with Invalid Choices**	**Categories with Overlapping Choices**	**Categories with Combinable Choices**	**Categories with Composite Choices**
$\mathbb{U}_{\mathsf{TRADE}}$	0 / 0	3 / 0	0 / 0	6 / 0	0 / 0	34 / 5
$\mathbb{U}_{\mathsf{PURCHASE}}$	0 / 1	9 / 1	2 / 1	26 / 2	0 / 1	42 / 8
$\mathbb{U}_{\mathsf{MEAL}}$	123 / 14	12 / 2	14 / 4	4 / 1	5 / 2	4 / 7

Table 3.13: Total Numbers of Different Types of Problematic Category Identified by Inexperienced and Experienced Testers

By Inexperienced Testers / Experienced Testers:						
	Mean Numbers of Different Types of Problematic Category in *Each PC*					
Functional Unit	**Irrelevant Categories**	**Categories with Missing Choices**	**Categories with Invalid Choices**	**Categories with Overlapping Choices**	**Categories with Combinable Choices**	**Categories with Composite Choices**
$\mathbb{U}_{\mathsf{TRADE}}$	0.00 / 0.00	0.06 / 0.00	0.00 / 0.00	0.13 / 0.00	0.00 / 0.00	0.71 / 0.63
$\mathbb{U}_{\mathsf{PURCHASE}}$	0.00 / 0.13	0.19 / 0.13	0.04 / 0.13	0.54 / 0.25	0.00 / 0.13	0.88 / 1.00
$\mathbb{U}_{\mathsf{MEAL}}$	2.80 / 1.75	0.27 / 0.25	0.32 / 0.50	0.09 / 0.13	0.11 / 0.25	0.09 / 0.88
Averages	**0.93 / 0.63**	**0.17 / 0.13**	**0.12 / 0.21**	**0.25 / 0.13**	**0.04 / 0.13**	**0.56 / 0.83**

PC = Set of potential categories and choices

Table 3.14: Mean Numbers of Different Types of Problematic Category in Each *PC* Identified by Inexperienced and Experienced Testers

fied by inexperienced and experienced testers, respectively.[7] Table 3.14 then shows the mean numbers of these problematic types in each *PC*. In addition, Tables 3.15 and 3.16 show the percentages of these problematic types in *potential* and *problematic* categories, respectively.

[7] Note that, for each row in Table 3.13 (which corresponds to a functional unit), the sum of the total numbers of different types of problematic category is equal to or greater than the corresponding total number of problematic categories shown in Table 3.10. This is because, for a potential category that is mistakenly identified, it may belong to one or more types of problematic category.

By Inexperienced Testers / Experienced Testers:						
	Percentages of Different Types of Problematic Category in All *Potential* Categories					
Functional Unit	**Irrelevant Categories**	**Categories with Missing Choices**	**Categories with Invalid Choices**	**Categories with Overlapping Choices**	**Categories with Combinable Choices**	**Categories with Composite Choices**
$\mathbb{U}_{\text{TRADE}}$	0.0% / 0.0%	1.1% / 0.0%	0.0% / 0.0%	2.3% / 0.0%	0.0% / 0.0%	12.8% / 9.3%
$\mathbb{U}_{\text{PURCHASE}}$	0.0% / 1.0%	1.9% / 1.0%	0.4% / 1.0%	5.5% / 2.0%	0.0% / 1.0%	8.8% / 7.9%
$\mathbb{U}_{\text{MEAL}}$	20.0% / 10.4%	2.0% / 1.5%	2.3% / 3.0%	0.7% / 0.7%	0.8% / 1.5%	0.7% / 5.2%
Averages	**6.7% / 3.8%**	**1.7% / 0.8%**	**0.9% / 1.3%**	**2.8% / 0.9%**	**0.3% / 0.8%**	**7.4% / 7.5%**

Table 3.15: Percentages of Different Types of Problematic Category in All Potential Categories Identified by Inexperienced and Experienced Testers

By Inexperienced Testers / Experienced Testers:						
	Percentages of Different Types of Problematic Category in All *Problematic* Categories					
Functional Unit	**Irrelevant Categories**	**Categories with Missing Choices**	**Categories with Invalid Choices**	**Categories with Overlapping Choices**	**Categories with Combinable Choices**	**Categories with Composite Choices**
$\mathbb{U}_{\text{TRADE}}$	0.0% / 0.0%	7.0% / 0.0%	0.0% / 0.0%	14.0% / 0.0%	0.0% / 0.0%	79.1% / 100.0%
$\mathbb{U}_{\text{PURCHASE}}$	0.0% / 9.1%	11.4% / 9.1%	2.5% / 9.1%	32.9% / 18.2%	0.0% / 9.1%	53.2% / 66.7%
$\mathbb{U}_{\text{MEAL}}$	77.8% / 50.0%	7.6% / 7.1%	8.9% / 14.3%	2.5% / 3.8%	3.2% / 7.1%	2.5% / 25.0%
Averages	**25.9% / 19.7%**	**8.7% / 5.4%**	**3.8% / 7.8%**	**16.5% / 7.3%**	**1.1% / 5.4%**	**44.9% / 63.9%**

Table 3.16: Percentages of Different Types of Problematic Category in All Problematic Categories Identified by Inexperienced and Experienced Testers

Observation 18: Table 3.14 shows that, with respect to the mean numbers of different types of problematic category in each *PC*, experienced testers have identified *fewer* irrelevant categories, categories with missing choices, and categories with overlapping choices than their inexperienced counterparts, but *more* categories with invalid choices, categories with combinable choices, and categories with composite choices. This indicates that experienced testers are not necessarily better than inexperienced ones in every aspect.

Observation 19: In Tables 3.15 and 3.16, we observe a tendency similar to observation 18. With respect to the percentages of different types of problematic category in all *PC*'s, experienced testers have identified *fewer* irrelevant categories, categories with missing choices, and categories with overlapping choices than their inexperienced counterparts, but *more* categories with invalid choices, categories with combinable choices, and categories with composite choices.

Observation 20: Given a problematic category $[X]$ identified by inexperienced or experienced testers, the chance that $[X]$ is a category with composite choices is the highest. On the other hand, the chance that it is a category with combinable choices is the smallest. This result leads to the following conclusions:

(a) Referring to Table 3.6 again, the mean numbers of choices in each category are 2.37 ($= \frac{23.2}{9.8}$) and 2.43 ($= \frac{29.2}{12.0}$) for inexperienced and experienced testers, respectively. The small numbers of choices per category suggest that all the subjects incline to reduce the total number of choices and, in turn, reduce the total number of complete test frames [8] with a view to saving testing effort. Obviously, fewer choices per category would also mean that the chance of having combinable choices is smaller.

(b) Without the support of a systematic identification technique, it is difficult for the subjects to partition a given category into choices such that all the values in each choice are similar in their effects on the system's behavior or in the type of output they produce. This difficulty results in categories with composite choices.

[8] Recall that a complete test frame is a set of choices such that a test case will be formed whenever a single value is selected from each choice.

3.2.4 Study 2: Effectiveness of Checklist Guideline

Objective and Steps

In Section 3.1.5, we have provided a checklist as a simple guideline for detecting missing/problematic categories and choices despite an impromptu identification approach. The objective of study 2 here is to evaluate the effectiveness of our checklist, in terms of its ability to help testers reduce the occurrence of missing and problematic categories in *PC*'s.

When commencing study 2, we first discuss with the 16 subjects about the missing and problematic categories that they have identified in the first study, and how the checklist can be used to help detect and remove these mistakes. We then ask each subject to perform another identification exercise according to the following scheme:

(a) **Subjects 1 to 8:** Identify from $\mathbb{U}_{\mathsf{MEAL}}$ a set *PC* of categories and their associated choices in an impromptu manner. Then use the checklist as a simple guideline for detecting and removing any mistake from the *PC*, and to refine any categories and choices in the *PC* if necessary. Finally, for every category or choice that remains in the *PC*, state the reason why it should be identified.

(b) **Subjects 9 to 16:** Repeat (a) above for $\mathbb{U}_{\mathsf{TRADE}}$ followed by $\mathbb{U}_{\mathsf{PURCHASE}}$, instead of $\mathbb{U}_{\mathsf{MEAL}}$.

Note that, in study 1 in which our checklist is not used, Subjects 1 to 8 have performed the identification exercise for $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$ only (but not $\mathbb{U}_{\mathsf{MEAL}}$), whereas Subjects 9 to 16 have performed the exercise for $\mathbb{U}_{\mathsf{MEAL}}$ only (but not $\mathbb{U}_{\mathsf{TRADE}}$ and $\mathbb{U}_{\mathsf{PURCHASE}}$). This arrangement prevents the experienced subjects from building up their knowledge of the same functional unit from study 1 and, hence, allows us to measure the effectiveness of our checklist in an objective manner.

Without Checklist / With Checklist:	
Functional Unit	**Total Numbers of Missing Categories**
$\mathbb{U}_{\mathsf{TRADE}}$	5 / 1
$\mathbb{U}_{\mathsf{PURCHASE}}$	5 / 2
$\mathbb{U}_{\mathsf{MEAL}}$	11 / 4
Totals	**21 / 7**

Table 3.17: Total Numbers of Missing Categories with and without Checklist

Findings and Discussions

We first consider missing categories:

Observation 21: Table 3.17 shows the total numbers of missing categories we have detected from the *PC*'s identified by experienced testers before and after using the checklist. From there, we see that the checklist helps reduce the numbers of missing categories for all the three functional units. Considering all the functional units together, the total number of missing categories is reduced by 67%, which is significant.

The following reports our observations related to problematic categories:

Observation 22: Table 3.18 shows the total numbers of different types of problematic category identified by experienced testers with and without the use of the checklist. Except the irrelevant categories identified from $\mathbb{U}_{\mathsf{TRADE}}$, the total numbers for each type of problematic category across the three functional units using the checklist do not exceed those totals when the checklist is not used. Considering all the three functional units together, the checklist is able to reduce the numbers of irrelevant categories, categories with missing choices, categories with invalid choices, categories with overlapping choices, categories with combinable choices,

Without Checklist / With Checklist:						
	Total Numbers of Different Types of Problematic Category					
Functional Unit	Irrelevant Categories	Categories with Missing Choices	Categories with Invalid Choices	Categories with Overlapping Choices	Categories with Combinable Choices	Categories with Composite Choices
$\mathbb{U}_{\text{TRADE}}$	0 / 1	0 / 0	0 / 0	0 / 0	0 / 0	5 / 2
$\mathbb{U}_{\text{PURCHASE}}$	1 / 1	1 / 0	1 / 1	2 / 0	1 / 0	8 / 3
$\mathbb{U}_{\text{MEAL}}$	14 / 9	2 / 0	4 / 1	1 / 0	2 / 0	7 / 3
Totals	**15 / 11**	**3 / 0**	**5 / 2**	**3 / 0**	**3 / 0**	**20 / 8**

Table 3.18: Total Numbers of Different Types of Problematic Category with and without Checklist

and categories with composite choices by about 27%, 100%, 60%, 100%, 100%, and 60%, respectively. Thus, the checklist is most effective in preventing the occurrence of categories with missing/overlapping/combinable choices.

Thus, observations 21 and 22 serve as strong evidence that the checklist is fairly effective to reduce missing and problematic categories. These two observations, however, also reconfirm the need for a systematic identification technique because missing and problematic categories still exist even with the use of the checklist.

Observation 23: Table 3.18 also shows that, among different types of problematic category, the checklist is least effective in reducing the occurrence of irrelevant categories. A plausible reason is that, to avoid the identification of irrelevant categories, testers must determine the influencing factors of the software system under test. This determination task is non-trivial and may sometimes be intangible at the specification stage.

3.2.5 Feedback of Subjects

After the two studies described in Sections 3.2.3 and 3.2.4, we organized a meeting with the 16 experienced subjects, with a view to finding out the following from these experienced testers:

(a) What components in a specification are useful for the identification of categories and choices?

(b) In what ways can the effectiveness of the identification process be improved?

These two issues will be discussed below.

Usefulness of Specification Components

Unlike specifications that are written in formal languages such as Z [81] and Boolean predicates [51, 73], informal specifications are often expressed in many different styles and formats, and contain a large variety of components. Examples of these components are narrative descriptions, use cases, activity diagrams, swimlane diagrams, data flow diagrams (DFD's), and data dictionaries. For a complex informal specification with many different components, software testers may find it difficult to identify categories and choices from the *entire* specification in one single round. Rather, testers may be inclined to decompose the identification process into several steps, each step focusing on only one specification component. We have observed that most of the experienced subjects have followed such an identification approach.

Activity Diagrams and Swimlane Diagrams

Inspired by the above observation, we have asked the experienced subjects what specification components they consider useful for the identification of categories and choices. In response to this question, the majority of these subjects consider activity

diagrams and swimlane diagrams to be very useful for category and choice identification. Basically, an activity diagram or a swimlane diagram represents the actions and decisions that occur as some function is performed. The diagrams use rounded rectangles to denote activities, arrows to represent control flows of activities, diamond icons (corresponding to *decision points*) to depict branching decisions, and solid thick bars to indicate the occurrence of parallel activities. Arrows are used to indicate *alternative threads* that emerge from every decision point depending on the *guard conditions* enclosed in square brackets. The diamond icon can also be used to show where the alternative threads merge again. Intuitively, the decision points and the guard conditions in an activity diagram or a swimlane diagram indicate where and how a software system *behaves differently*. (Readers may recall that a category corresponds to a factor of the software system that affects its *execution behavior*.) Thus, most experienced subjects consider that this characteristic makes activity diagrams and swimlane diagrams very useful for identifying categories and choices. [9]

Example 3.10 below further illustrates the usefulness of activity diagrams:

Example 3.10 (Generation of Daily Meal Schedules)
Refer to Example 3.9 about how the generation of daily meal schedules can be affected by the exceptional flight schedules (EFS's) in the functional unit $\mathbb{U}_{\mathsf{MEAL}}$ of the meal ordering system MOS. (Basically, an EFS allows users to change the estimated time of departure of a flight on a particular date after the master flight schedule of this flight has been defined.)

Consider the partial activity diagram in Figure 3.2, which shows part of the generation process of daily meal schedules. The upper diamond icon represents a decision point associated with two guard conditions, "EFS is Defined" and "EFS is Not Defined". The

[9] The swimlane diagram is a useful variation of the activity diagram. Both types of diagram contain decision points and their associated guard conditions. A swimlane diagram, however, can also indicate which actor (if there are multiple actors involved in a specific function) or analysis class is responsible for the action described by an activity rectangle. Responsibilities are represented as parallel segments that divide the diagram into swimlanes, like the lanes in a swimming pool [65].

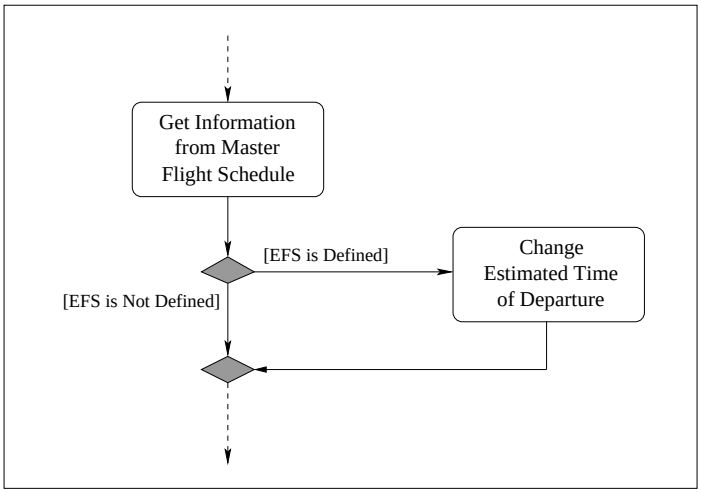


Figure 3.2: Part of an Activity Diagram for the Generation of Daily Meal Schedules

decision point and guard conditions clearly indicate that the category [Defined EFS] should be identified, with |Defined EFS: Yes| and |Defined EFS: No| as its two associated choices. Such identification is based on the rationale that the two guard conditions correspond to different flows of control and, in turn, different execution behavior of MOS. ■

Note that the illustration of the usefulness of activity diagrams in Example 3.10 is also applicable to swimlane diagrams.

Sample Input Screens and Data Dictionaries

About half of the experienced subjects also report that sample input screens and data dictionaries are useful for the identification task, although these specification components are not as good as activity diagrams and swimlane diagrams. Consider sample input screens first. These experienced subjects report that sample input screens, by showing the input elements explicitly, represent a good repository in which input elements affecting the execution behavior of the software system can be identified as categories. By

similar reasons, data dictionaries are also considered to be useful for identification. These experienced subjects also think that the sample input screens are more useful than data dictionaries. This is because the elements in a data dictionary do not necessarily correspond to inputs, unlike those in an input screen. For example, some of the former elements may correspond to system output and, hence, should not be identified as categories. Thus, the experienced subjects need to spend extra effort to identify a subset of the elements in a data dictionary that correspond to system inputs.

The experienced subjects also give the following two reasons why they consider sample input screens and data dictionaries less useful than activity diagrams and swimlane diagrams: Firstly, the usefulness of activity diagrams and swimlane diagrams is largely due to the decision points and the guard conditions, which do not exist in sample input screens and data dictionaries. Secondly, as illustrated in Example 3.10, the guard conditions largely ease the identification of choices for each category (which corresponds to a decision point in an activity diagram or swimlane diagram). On the other hand, in the sample input screens and data dictionaries, even though software testers can determine which input elements should be used for category identification, little information is provided for identifying the choices for each category.

Data Flow Diagrams

Finally, six experienced subjects also point out that lower-level DFD's are useful for identifying influencing factors as categories and choices. In short, the DFD takes an input-process-output view of a system [65]. That is, data flow into the software, are transformed by processing elements, and resultant data flow out of the software. The flows of data are indicated by labeled arrows, transformations are represented by bubbles, and data stores are denoted by double lines. DFD's are arranged in levels. The highest level data flow model (also called a *level-0* DFD or *context diagram*) represents the

system as a whole. Lower-level DFD's refine higher-level DFD's, providing more details. Pressman [65] argues that, although DFD's are not a formal part of UML, they can be used to complement UML diagrams and provide additional insight into system requirements and flows.

The usefulness of lower-level DFD's for identifying categories and choices is mainly contributed by their data stores. See Figure 3.3, for instance, for a level-1 DFD for generating daily meal schedules in $\mathbb{U}_{\mathsf{MEAL}}$ of MOS. There are four data stores in this DFD. Let us consider the data store "Master Flight Schedules (MFS's)". Intuitively, an MFS indicates an environment condition of $\mathbb{U}_{\mathsf{MEAL}}$ with three possible statuses (namely, an undefined MFS, a defined but empty MFS, and a defined and nonempty MFS), each of which will result in a different execution behavior of the system. Thus, the category [Status of MFS] should be identified with three associated choices |Status of MFS: Not Defined|, |Status of MFS: Defined but Empty|, and |Status of MFS: Defined and Nonempty|. (The first two choices are identified to test how the system behaves under abnormal situations.) Categories and choices can be identified for influencing environment conditions corresponding to the data stores "Exceptional Flight Schedules (EFS's)" and "Exceptional Crew Configuration Records (ECCR's)" in a similar manner.

Readers should note that, although DFD's are useful for identifying influencing *environment conditions* as categories and choices (by virtue of the data stores), they provide little information for identifying influencing *parameters* as categories and choices. Although some data flows in a DFD may correspond to potential parameters, we do not know whether they are influencing by considering the DFD alone. In addition, instead of representing individual data items, such data flows may correspond to groups of data items, some of which may correspond to influencing parameters while the others may not.

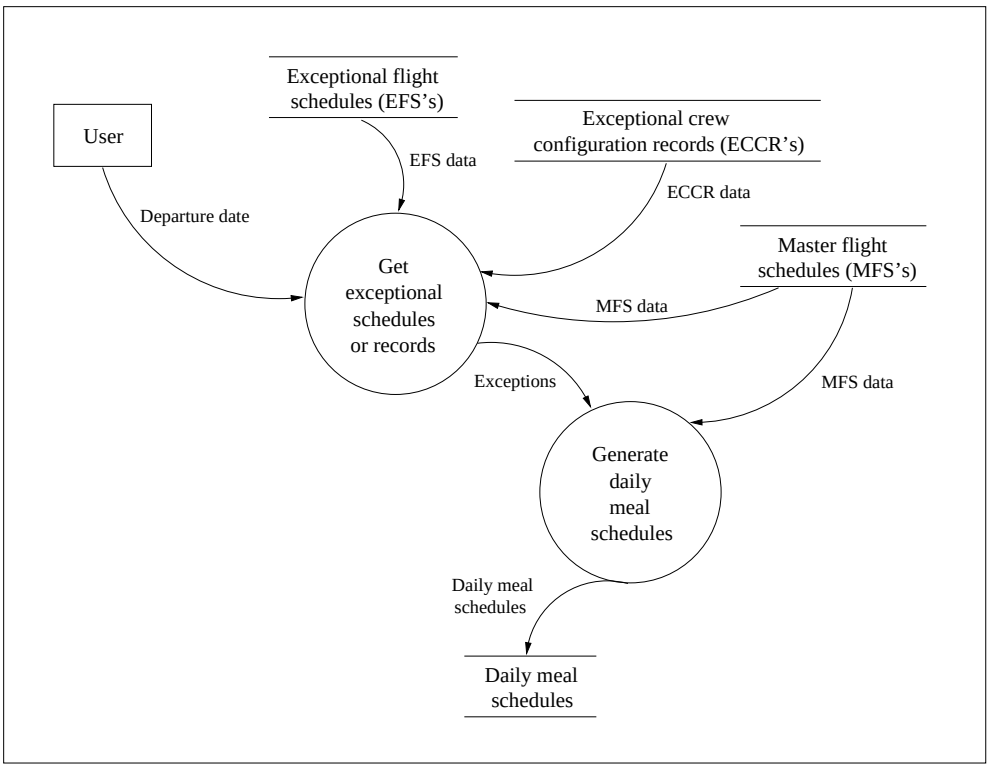


Figure 3.3: A Level-1 Data Flow Diagram for the Generation of Daily Meal Schedules

Improvement to the Impromptu Identification Process

In the discussion meeting with the 16 experienced subjects, we found that 11 of them have also constructed choice relation tables (for CHOC'LATE) [21, 62] or classification trees (for CTM) [10, 11, 20, 39, 44], even though they were not asked to do so. Basically, the construction of such tables or trees represents the next step in CHOC'LATE and CTM after the identification of categories and choices. The main purpose of the tables and trees is to capture the constraints among choices (in CHOC'LATE) or categories (in CTM), through which complete test frames can be subsequently generated (see our earlier discussions in Sections 2.1 and 2.2).

These 11 experienced subjects argue that the identification of categories and choices are closely related to the identification of their constraints and, hence, these two steps

should not be totally separate. More specifically, the subjects need to consider the constraints when identifying categories and choices. Otherwise, the identification task may not be well executed. This point echoes problem 2 of CHOC'LATE and CTM as stated in Section 2.3, and it is illustrated in Example 3.11 below:

Example 3.11 (Exceptional Schedules and Records)
Refer to Example 3.9 about how the generation of daily meal schedules can be affected by the exceptional flight schedules (EFS's) and the exceptional crew configuration records (ECCR's) if they have been defined.

In view of the possible definition of EFS's and ECCR's, any master flight schedule will fall into one of the following situations:

(a) It is associated with an EFS but not an ECCR.

(b) It is associated with an ECCR but not an EFS.

(c) It is associated with an EFS and an ECCR.

(d) It is not associated with any EFS or ECCR.

There are two approaches to identifying categories and choices:

Approach 1: Intuitively, in order to cater for all the above situations, two categories should be identified: [Defined EFS] and [Defined ECCR]. The first category should have two associated choices: |Defined EFS: Yes| and |Defined EFS: No|. Similarly, the second category should have two associated choices: |Defined ECCR: Yes| and |Defined ECCR: No|. With these categories and choices, each of the above four situations can be tested *separately* by selecting one choice in [Defined EFS] and one in [Defined ECCR]. For example, the selection of both |Defined EFS: No| and |Defined ECCR: Yes| will cover situation (b).

Approach 2: Some software testers, however, may argue that we do not need two categories and four choices. The rationale is that such an approach will increase the number of complete test frames B^c's (and in turn the number of test cases *tc*'s) generated and, hence, will require more testing effort. Instead, these testers propose that only one category [Exceptional Schedules/Records Defined], with |Exceptional Schedules/Records Defined: Yes| and |Exceptional Schedules/Records Defined: No| as its two associated choices, should be identified. Note that the choice |Exceptional Schedules/Records Defined: Yes| caters for situations (a), (b), and (c) collectively; whereas |Exceptional Schedules/Records Defined: No| caters for situation (d) only. On one hand, the new proposal will result in fewer B^c's and *tc*'s, and some test effort can be saved. On the other hand, the resulting test suite will be less "refined", in the sense that *tc*'s cannot be generated to cater for situations (a), (b), and (c) on an individual basis.

In order to judge which identification approach should be adopted, the experienced subjects need to consider a question: Which approach will generate a test suite with a better coverage and, hence, a higher chance of revealing failures? The question can only be answered if the subjects know:

(i) What are the other categories and their associated choices to be identified?

(ii) What are the constraints among the identified categories/choices? These constraints determine how choices are combined to form part of any B^c.

(iii) Suppose B_1^c, B_2^c, and B_3^c are any complete test frames containing the following pairs of choices:

- (|Defined EFS: Yes| and |Defined ECCR: No|),
- (|Defined EFS: No| and |Defined ECCR: Yes|), and

- (|Defined EFS: Yes| and |Defined ECCR: Yes|),

respectively. Suppose, further, that B_1^c, B_2^c, and B_3^c only differ in the above choice pairs; all the other choices contained in these three complete test frames are identical (that is, $B_1^c \setminus \{|\text{Defined EFS: Yes}|, |\text{Defined ECCR: No}|\} = B_2^c \setminus \{|\text{Defined EFS: No}|, |\text{Defined ECCR: Yes}|\} = B_3^c \setminus \{|\text{Defined EFS: Yes}|, |\text{Defined ECCR: Yes}|\}$). In this case, are B_1^c, B_2^c, and B_3^c associated with different execution behavior of the system? If yes, then approach 1 should be used to generate a test suite with a better coverage. Otherwise, approach 2 should be used to generate a smaller test suite (that is, with fewer *tc*'s) without jeopardizing its effectiveness of revealing failures.

■

In summary, the experienced subjects suggest that identifying categories and choices and identifying constraints among categories/choices should not be viewed as two totally distinct processes.

3.2.6 Threads to Validity

Our second round of studies has two limitations owing to various settings. Firstly, as compared with our first round of studies [15] in Section 3.1, which involved 44 or more undergraduates or postgraduates, our current studies involve only 16 experienced subjects. The current studies would certainly be better if more experienced subjects participated in it. It is not easy, however, to find a large group of well-experienced software testers who are willing to participate in empirical studies (with or without remuneration). Secondly, one may argue that the subjects may gain in experience after doing one case (such as $\mathbb{U}_{\mathsf{TRADE}}$). We believe that this effect should be minimal. This is because, in both studies 1 and 2, the subjects were advised of their errors only after they have completed all the identification tasks for their assigned specification(s).

3.2.7 Summary

We have described our comparative studies using three commercial specifications and involving inexperienced and experienced software testers. In general, experienced testers have identified more potential categories and choices. They also have smaller number of missing categories and smaller number of problematic categories. At the same time, they have identified larger number of non-problematic categories. These observations thus provide evidence that experience in software development and testing does help improve the quality of the identified *PC* despite an impromptu identification approach. We must, however, point out that the contribution of experience to the reduction of mistakes decreases with the complexity of the functional units. We find from the empirical results that, although experienced testers can identify more non-problematic categories when the functional units become more complex, this advantage is not sufficient to offset the increase in problematic categories at the same time. (This phenomenon also occurs for inexperienced testers.) Thus, software development experience cannot substitute the demand for a systematic identification methodology.

Regarding the types of mistake, experienced testers are not necessarily better than inexperienced ones in every aspect. Firstly, on one hand, the increase in missing categories in each *PC* is larger for inexperienced testers than experienced testers when the functional units become more complex; on the other hand, the increase in problematic categories in each *PC* is smaller for inexperienced testers than experienced testers as the functional units become more complex. Secondly, with respect to all potential/problematic categories, experienced testers have identified fewer irrelevant categories and categories with missing/overlapping choices, but more categories with invalid/combinable/composite choices.

Moreover, we observe that the use of the checklist presented in Section 3.1.5 helps software testers reduce the occurrence of missing categories and problematic categories

of all types. Among the different types of problematic category, the checklist is more effective for reducing the occurrence of categories with missing/overlapping/combinable choices, but least effective in reducing the occurrence of irrelevant categories. Our studies also show that, with the use of the checklist, even software practitioners with substantial years of commercial experience in software development and testing still make a number of mistakes. We can conclude from this latter observation that there is a great demand for a systematic identification technique.

In terms of the usefulness of individual specification components in identifying categories and choices, the experienced subjects favor activity diagrams, swimlane diagrams, sample input screens, and data dictionaries in various extents. Furthermore, the subjects suggest that, in order to improve the effectiveness of an impromptu approach, the identification of categories, choices, and constraints between categories/choices should not be considered separately. With such subjects' feedback, we conclude that the systematic identification technique to be developed must be able to apply to a specification with many different components and should allow the identification of categories, choices, and their constraints to be done together.

Chapter 4

Our Identification Methodology: DESSERT

4.1 Overall Approach of DESSERT

Unlike formal specifications (denoted by $\mathbb{S}$'s) which are written in rigorous specification languages such as Z and Boolean predicates [51, 73], informal $\mathbb{S}$'s are often expressed in many different styles and formats, and contain a large variety of components. These specification components (denoted by $\mathbb{C}$'s) include, for instance, narrative descriptions, use cases, class diagrams, state machines, activity diagrams, swimlane diagrams, data flow diagrams, system flowcharts, and decision tables.

Given the many possible $\mathbb{C}$'s in an informal $\mathbb{S}$, a software tester may feel difficult applying CHOC'LATE for the entire $\mathbb{S}$.[1] Specifically, the main difficulty is related to the construction of a choice relation table (denoted by $\mathcal{T}$), which contains many choices and choice relations that are derived from different $\mathbb{C}$'s. (Once $\mathcal{T}$ is ready, the construction of

[1] We observe that CHOC'LATE and CTM have similar (but not identical) approaches to test case generation. For the rest of the thesis, we shall discuss DESSERT in the context of CHOC'LATE.

test cases from the table is supported by well defined algorithms.) The difficulty is caused by the three problems of test case generation methods as discussed in Section 2.3. Firstly, there is a demand for systematic identification methodologies for categories and choices, especially for informal $\mathbb{S}$'s (problem 1). Secondly, CHOC'LATE assumes that identifying categories and choices and identifying choice relations are two distinct and independent tasks (problem 2). Thirdly, CHOC'LATE assumes that the software tester can construct a single $\mathcal{T}$ for the entire $\mathbb{S}$ without much difficulty, regardless of its size and complexity (problem 3).

In this chapter, we introduce a **D**ivid**E**-and-conquer methodology for identifying categorie**S**, choice**S**, and choic**E** **R**elations for **T**est case generation (abbreviated as DESSERT), with a view to alleviating the above problems. Basically, DESSERT uses the following three-step approach to constructing a $\mathcal{T}$ for each functional unit (note that the first step in CHOC'LATE and CTM is to decompose the software system into several functional units that can be tested independently):

(1) Decompose the entire $\mathbb{S}$ (corresponding to the functional unit selected for testing) into several components $\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_n$ (where $n \geq 1$), with each $\mathbb{C}_i$ ($1 \leq i \leq n$) models part of the dynamic behavior of that unit under test.

(2) Construct a *preliminary* choice relation table τ_i from each $\mathbb{C}_i$.

(3) Consolidate $\tau_1, \tau_2, \ldots, \tau_n$ into a single $\mathcal{T}$. This "divide-and-conquer" approach is particularly useful when the software tester finds $\mathbb{S}$ to be too large and complex for one single round of category, choice, and choice relation identification.

4.2 Construction of Preliminary Choice Relation Tables

Firstly, consider step (1) of DESSERT listed above. There are many ways to decompose a $\mathbb{S}$ into its components, as proposed by software researchers and practitioners. Thus, the

details of such decomposition will not be repeated here. The rest of this section deals with step (2) of DESSERT and Section 4.3 deals with step (3). Limited by the space of this thesis, it is not feasible to discuss in detail how a preliminary choice relation table τ_i is constructed from every $\mathbb{C}_i$. Thus, here we only illustrate how to construct a τ from an activity diagram.

4.2.1 Overview of Activity Diagrams

Activity diagrams, denoted by $\mathcal{A}$'s, provide graphical representations of the flow of interaction within specific scenarios. In its basic form, an $\mathcal{A}$ is a simple and intuitive illustration of what happens in a control flow, what activities can be done in parallel, and whether there are alternative paths through the control flow. As such, an $\mathcal{A}$ can be used to model everything from a high-level business workflow that involves many different use cases, to the details of an individual use case, all the way down to the specific details of an individual method.

An $\mathcal{A}$ starts with a solid circle, representing the *initial activity*. An arrow shows the control flow of activities, which are represented by rounded rectangles labeled for the activities performed. An asterisk on an arrow indicates that the control flow is iterated. The end of a control flow is indicated by a "bull's eye", known as a *final activity*. A single path of execution through an $\mathcal{A}$ is called a *thread*. A diamond represents a *decision point*. Conditions for each arrow out of a decision point (known as an *alternative thread*) are enclosed in brackets, and they are called *guard conditions*. The diamond icon can also be used to show where the alternative threads merge again. A solid thick bar is called a *synchronization bar*. Multiple arrows out of a synchronization bar indicate activities that can be performed in parallel. Multiple arrows into a synchronization bar indicate activities that must all be completed before the next process can begin.

Intuitively, the decision points and guard conditions in $\mathcal{A}$'s indicate where and how

the software system behaves differently. This characteristic makes $\mathcal{A}$'s very useful for deriving information to identify categories, choices, and possibly choice relations for the construction of τ's.

Here, we shall introduce a new concept *complete thread*, which is essential for understanding DESSERT described in this chapter.

Definition 4.1 (Complete Thread)

*In an $\mathcal{A}$, a thread is said to be **complete** if and only if it starts with the initial activity and ends with a final activity.*

4.2.2 Table Construction from Activity Diagrams

We are now ready to present an algorithm (construct_table) for constructing a preliminary choice relation table τ from an activity diagram $\mathcal{A}$. As stated as problem 2 in Section 2.3, identifying choices and their relations (or constraints) are not two totally distinct and independent tasks. On the contrary, when identifying categories and choices, we need to consider choice relations at the same time [15]. Otherwise, problematic categories and choices may be mistakenly identified. Because of this, construct_table does not only help software testers identify categories and choices, but also choice relations as many as possible.

An Algorithm (construct_table) for Constructing a Preliminary Choice Relation Table from an Activity Diagram

Given an activity diagram $\mathcal{A}$ containing one or more guard conditions (denoted by gc_i's where $i \geq 1$), with each gc_i containing one or more subconditions (denoted by $sc_{(i,j)}$'s where $j \geq 1$) separated from the others by the logical operators "AND" or "OR":

(1) Let $F(sc_{(i,j)})$ denote the factor(s) associated with $sc_{(i,j)}$. For every $sc_{(i,j)}$ in every gc_i in $\mathcal{A}$:

   (a) Define the category $[F(sc_{(i,j)})]$ if it has not been defined. Note that the same category may be associated with different $sc_{(i,j)}$'s in the same or different gc_i's.

   (b) Define $sc_{(i,j)}$ as a choice in $[F(sc_{(i,j)})]$ if such a choice has not been defined.

(2) For every pair of overlapping choices $|X : x_i|$ and $|X : x_j|$:

   (a) If $|X : x_i| \subsetneq |X : x_j|$ (that is, $|X : x_i|$ is a subset of $|X : x_j|$ and $|X : x_i| \neq |X : x_j|$), then delete $|X : x_j|$ and define the choice $(|X : x_j| \setminus |X : x_i|)$ if it has not been defined.

   (b) Else if $|X : x_j| \subsetneq |X : x_i|$, then delete $|X : x_i|$ and define the choice $(|X : x_i| \setminus |X : x_j|)$ if it has not been defined.

   (c) Else delete $|X : x_i|$ and $|X : x_j|$ and define the following choices if they have not been defined:

      - $|X : x_i| \setminus |X : x_j|$
      - $|X : x_j| \setminus |X : x_i|$
      - $|X : x_i| \cap |X : x_j|$

   /* By Definition 3.7, overlapping choices must be valid and distinct. Thus, the newly defined choices in step (2) above are nonempty. */

(3) Let $|X : x_1|, |X : x_2|, \ldots, |X : x_n|$ (where $n \geq 1$) denote all choices in $[X]$ identified after step (2), and $E([X])$ denote the input domain relevant to $[X]$. For every category $[X]$ with missing choices, define a new choice $|X : x|$ such that $E([X]) = |X : x| \cup |X : x_1| \cup |X : x_2| \cup \cdots \cup |X : x_n|$ and $|X : x| \cap |X : x_i| = \emptyset$ (where $1 \leq i \leq n$).

(4) Initialize a preliminary choice relation table τ by assigning a null value to every choice relation $|X : x| \mapsto |Y : y|$ in τ.

(5) Assign the relational operator "$\sqsubset$" to every choice relation $|X : x_i| \mapsto |X : x_i|$ in τ.

(6) Assign the relational operator "$\not\sqsubset$ to every choice relation $|X : x_i| \mapsto |X : x_j|$ (where $i \neq j$) in τ.

(7) Let $sc(|X : x|)$ denote the subcondition corresponding to the choice $|X : x|$. For every choice relation $|X : x| \mapsto |Y : y|$ in τ such that $\big([X] \neq [Y]$ and there exists some gc in $\mathcal{A}$ so that gc contains both $sc(|X : x|)$ and $sc(|Y : y|)\big)$, assign the relational operator "$\boxed{\mathbb{P}}$" to $|X : x| \mapsto |Y : y|$.

(8) Let $gc(|X : x|)$ denote the guard condition corresponding to the choice $|X : x|$. For every yet-to-be-defined relational operator in τ corresponding to a choice relation $|X : x| \mapsto |Y : y|$ such that $\big([X] \neq [Y]$, both $gc(|X : x|)$ and $gc(|Y : y|)$ appear in $\mathcal{A}$, and neither $gc(|X : x|)$ nor $gc(|Y : y|)$ is associated with any parallel threads in $\mathcal{A}\big)$, use the following rules to determine the relevant choice relation for $|X : x| \mapsto |Y : y|$:

   (a) Assign the relational operator "$\sqsubset$" to $|X : x| \mapsto |Y : y|$ if, for every complete thread t associated with $gc(|X : x|)$, t is also associated with $gc(|Y : y|)$.

   (b) Assign the relational operator "$\boxed{\mathbb{P}}$" to $|X : x| \mapsto |Y : y|$ if:

      (i) there exists some complete thread t associated with $gc(|X : x|)$ such that t is also associated with $gc(|Y : y|)$; and

      (ii) there exists some complete thread t' associated with $gc(|X : x|)$ such that t' is not associated with $gc(|Y : y|)$.

   (c) Assign the relational operator "$\not\sqsubset$" to $|X : x| \mapsto |Y : y|$ if no complete thread is associated with both $gc(|X : x|)$ and $gc(|Y : y|)$.

Readers may refer to [16] for an example of how to apply the algorithm construct_table. There are two important characteristics of the algorithm. Firstly, construct_table helps identify a set of categories and choices based on the guard conditions *gc*'s appearing in $\mathcal{A}$. Intuitively, each *gc* corresponds to a particular execution behavior of the software and, hence, some categories and choices should be identified with respect to *gc*. Secondly, construct_table will not only identify a set of categories and choices from $\mathcal{A}$, but also determine as many choice relations $|X : x| \mapsto |Y : y|$ in τ as possible. This does not only improve the efficiency of constructing a preliminary $\mathcal{T}$, but also reduce the chance of incorrect manual definition of choice relations. The occurrence of incorrect choice relations will undoubtedly affect the comprehensiveness of the test cases generated by CHOC'LATE.

After the execution of construct_table, all the missing relational operators in τ should be determined by some other means such as the tester's own expertise and judgment. In support of such determination task, CHOC'LATE provides two useful features: (a) consistency checking of manually defined choice relations, and (b) automatic deduction of new choice relations.[2] Readers may refer to [21, 62] for the details of these features.

Certain steps in construct_table warrant additional explanations and discussions:

- **Step (1):** Suppose there exists a decision point in an $\mathcal{A}$, corresponding to the factor "Location of Study". This decision point is associated with two guard conditions "Local On-Campus" and "Overseas Off-Campus" (both are also subconditions by themselves). In step (1) of construct_table, we should define [Location of Study] as a category, with |Location of Study: Local On-Campus| and |Location of Study: Overseas Off-Campus| as its associated choices, if they do not exist. One may argue

[2] Such techniques of consistency checking and automatic deduction assume that the choice relations do not involve overlapping choices.

that we should instead define the entire subconditions as categories with "Yes" as their associated choices. That is, we should define the category [Local On-Campus] with an associated choice |Local On-Campus: Yes|, and the category [Overseas Off-Campus] with an associated choice |Overseas Off-Campus: Yes|. In this approach, however, we have two different categories for the same factor (or decision point), which is obviously counter-intuitive.

In some situations, software testers need to use their judgment to decompose a $sc_{(i,j)}$ into two or more simpler subconditions before applying step (1) of construct_table for category and choice identification. Consider, for instance, the subcondition "$m, n > 7$". This subcondition should first be decomposed into two simpler subconditions "$m > 7$" and "$n > 7$" before applying step (1).

- **Step (7):** For any choice $|X : x|$ in τ, it may be directly defined from a $sc_{(i,j)}$ in a gc_i in step (1)(b) of construct_table, or may be generated in step (2)(a)(ii), (2)(b)(ii), or (3). In the latter case, this generated $|X : x|$ may not have a corresponding $sc(|X : x|)$ in $\mathcal{A}$.

  Let us use an example to explain the rationale of step (7). Suppose we have two guard conditions "$(1 < a < 12)$ AND $(3 < b < 20)$" and "$(1 < a < 12)$ OR $(3 < b < 20)$". They will result in the definition of the choices $|1 < a < 12|$ and $|3 < b < 20|$. Such conditions imply that any $a \in |1 < a < 12|$ and $b \in |3 < b < 20|$ may or may not coexist in the same input to the system under test. Thus, we should assign the relational operator "$\mathbb{E}$" to $|1 < a < 12| \mapsto |3 < b < 20|$.

- **Step (8):** The rules for determining the choice relation for $|X : x| \mapsto |Y : y|$ is based on the rationale that, if $gc(|X : x|)$ and $gc(|Y : y|)$ appear in the same complete thread t in $\mathcal{A}$, then we must select both $|X : x|$ and $|Y : y|$ to form part of some complete test frames, from which test cases can be generated to traverse t for the

purpose of testing.

4.3 Consolidation of Preliminary Choice Relation Tables

This section discusses step (3) of DESSERT (see Section 4.1), which is complicated because of the following problems:

(a) A specification component $\mathbb{C}$ alone may carry incomplete information, because it only constitutes part of $\mathbb{S}$. Consider an example. Suppose that two choices $|X : x|$ and $|Y : y|$ always co-exist in one specification component $\mathbb{C}_1$ but never occur together in another component $\mathbb{C}_2$. One tester may conclude that $|X : x| \sqsubset |Y : y|$ and $|Y : y| \sqsubset |X : x|$ by considering $\mathbb{C}_1$ alone, while another tester may conclude that $|X : x| \not\sqsubset |Y : y|$ and $|Y : y| \not\sqsubset |X : x|$ from $\mathbb{C}_2$ alone. In fact, these four choice relations are all wrong, because $\mathbb{C}_1$ and $\mathbb{C}_2$ together suggest that the correct choice relations should be $|X : x| \boxdot |Y : y|$ and $|Y : y| \boxdot |X : x|$.

(b) Constructing τ_i's separately from individual $\mathbb{C}_i$'s may result in the occurrence of overlapping choices *across* different τ_i's. Such overlapping choices can only be detected when considering different τ_i's together.

(c) When problems (a) and (b) occur together, the situation will become more complicated. We illustrate this situation with an example. Suppose:

   (i) Choices $|X : x_1|$ and $|Y : y|$ are identified from $\mathbb{C}_1$. Then, the choice relations $|X : x_1| \sqsubset |Y : y|$ and $|Y : y| \sqsubset |X : x_1|$ are defined based on $\mathbb{C}_1$ alone.

   (ii) Choices $|X : x_2|$ and $|Y : y|$ are identified from $\mathbb{C}_2$. Then, the choice relations $|X : x_2| \not\sqsubset |Y : y|$ and $|Y : y| \not\sqsubset |X : x_2|$ are defined based on $\mathbb{C}_2$ alone.

   (iii) $|X : x_1| \subsetneq |X : x_2|$ and $|X : x_2| \subsetneq |X : x_1|$.

Let $|X : x_3| = |X : x_1| \cap |X : x_2|$. Based on $\mathbb{C}_1$ alone, we can deduce that $|X : x_3| \sqsubset |Y : y|$ and $|Y : y| \sqsubseteq |X : x_3|$, because $|X : x_3| \subsetneq |X : x_1|$. On the other hand, based on $\mathbb{C}_2$ alone, we can deduce that $|X : x_3| \not\sqsubset |Y : y|$ and $|Y : y| \not\sqsubset |X : x_3|$, because $|X : x_3| \subsetneq |X : x_2|$. Here we have two contradictions, namely, $|X : x_3| \sqsubset |Y : y|$ versus $|X : x_3| \not\sqsubset |Y : y|$ and $|Y : y| \sqsubseteq |X : x_3|$ versus $|Y : y| \not\sqsubset |X : x_3|$. Such contradictions result from the fact that neither $\mathbb{C}_1$ nor $\mathbb{C}_2$ carries complete information of the entire $\mathbb{S}$. In such circumstances, we need to redefine $|X : x_1|$ and $|X : x_2|$ by considering $\mathbb{C}_1$ and $\mathbb{C}_2$ together. Such redefinition will render the previously determined choice relations $|X : x_1| \sqsubset |Y : y|$, $|Y : y| \sqsubset |X : x_1|$, $|X : x_2| \not\sqsubset |Y : y|$, and $|Y : y| \not\sqsubset |X : x_2|$ useless. Thus, the initial effort spent on defining the original choices and choice relations would be wasted. Furthermore, additional choice relations have to be determined based on the newly defined choices. [3]

In view of the above problems, step (3) of DESSERT is decomposed into two substeps (3.1) and (3.2): the first substep deals with problem (a), and the second substep deals with problem (b). (Problem (c) will disappear automatically after problems (a) and (b) have been solved.) These two substeps are discussed below:

4.3.1 Step (3.1) of DESSERT

To alleviate problem (a) mentioned above, we have formulated Proposition 4.1 below, which will be used by the algorithm integrate_table presented after the proposition. First, let us introduce the notation:

- $\mathbb{C}_i$, where $i \geq 1$, denotes a specification component and $\mathbb{D} = \{\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_n\}$, where $n \geq 1$, denotes a subset of all specification components.

[3] Problem (c) above will become even more complicated if $|Y : y|$ in (c)(i) is replaced by $|Y : y_1|$, $|Y : y|$ in (c)(ii) is replaced by $|Y : y_2|$, and $|Y : y_1|$ overlaps with $|Y : y_2|$.

- $|X : x| \mapsto_{\mathbb{D}} |Y : y|$ denotes the choice relation between $|X : x|$ and $|Y : y|$ with respect to $\mathbb{D}$, determined according to the information derived from $\mathbb{C}_1$, $\mathbb{C}_2$, ..., $\mathbb{C}_n$. The three possible choice relations thus determined are denoted by $|X : x| \sqsubset_{\mathbb{D}} |Y : y|$, $|X : x| \sqsubseteq_{\mathbb{D}} |Y : y|$, and $|X : x| \not\sqsubset_{\mathbb{D}} |Y : y|$.

- $|X : x| \mapsto |Y : y|$ denotes the choice relation determined according to the information derived from the *entire* specification $\mathbb{S}$. In particular, when $\mathbb{D}$ contains all the components that constitute the entire $\mathbb{S}$, $|X : x| \mapsto_{\mathbb{D}} |Y : y| = |X : x| \mapsto |Y : y|$.

Using the above notation, we state Proposition 4.1 as follows.

Proposition 4.1 (Choice Relations in Different Sets of Specification Components) *Let* (i) $[X]$ *and* $[Y]$ *be distinct categories,* (ii) $|X : x|$ *and* $|Y : y|$ *be choices, and* (iii) $\mathbb{D}_1$ *and* $\mathbb{D}_2$ *be sets of specification components.*

(a) *If* $|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$ *and* $|X : x| \sqsubset_{\mathbb{D}_2} |Y : y|$, *then* $|X : x| \sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

(b) *If* $|X : x| \sqsubseteq_{\mathbb{D}_1} |Y : y|$ *and* $|X : x| \sqsubseteq_{\mathbb{D}_2} |Y : y|$, *then* $|X : x| \sqsubseteq_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

(c) *If* $|X : x| \not\sqsubset_{\mathbb{D}_1} |Y : y|$ *and* $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$, *then* $|X : x| \not\sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

(d) *If* $|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$ *and* $|X : x| \sqsubseteq_{\mathbb{D}_2} |Y : y|$, *then* $|X : x| \sqsubseteq_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

(e) *If* $|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$ *and* $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$, *then* $|X : x| \sqsubseteq_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

(f) *If* $|X : x| \sqsubseteq_{\mathbb{D}_1} |Y : y|$ *and* $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$, *then* $|X : x| \sqsubseteq_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

Proof of Proposition 4.1 (Choice Relations in Different Sets of Specification Components)

(a)–(c) The proofs follow directly from the definition of "$\sqsubset$", $\sqsubseteq$, and $\not\sqsubset$, respectively.

(d) Suppose $|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$ and $|X : x| \sqsubseteq_{\mathbb{D}_2} |Y : y|$. If we assumed $|X : x| \sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$, then it would mean that, with respect to both $\mathbb{D}_1$ and $\mathbb{D}_2$, any B^c containing $|X : x|$ must also contain $|Y : y|$, which would contradict $|X : x| \sqsubseteq_{\mathbb{D}_2} |Y : y|$. On the other hand, if we assumed $|X : x| \not\sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$, then it would mean that, with respect to both $\mathbb{D}_1$ and $\mathbb{D}_2$, every B^c containing $|X : x|$ would not contain $|Y : y|$. This situation would contradict $|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$ and $|X : x| \sqsubseteq_{\mathbb{D}_2} |Y : y|$. Since "$\sqsubset$", "$\sqsubseteq$", and "$\not\sqsubset$" are exhaustive, we must have $|X : x| \sqsubseteq_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

(e) Suppose $|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$ and $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$. If we assumed $|X : x| \sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$, then it would mean that, with respect to both $\mathbb{D}_1$ and $\mathbb{D}_2$, any B^c containing $|X : x|$ must also contain $|Y : y|$, which would contradict $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$. On the other hand, if we assumed $|X : x| \not\sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$, then it would mean that, with respect to both $\mathbb{D}_1$ and $\mathbb{D}_2$, every B^c containing $|X : x|$ would not contain $|Y : y|$. This situation would contradict $|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$. Since "$\sqsubset$", "$\sqsubseteq$", and "$\not\sqsubset$" are exhaustive, we must have $|X : x| \sqsubseteq_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

(f) Suppose $|X : x| \sqsubseteq_{\mathbb{D}_1} |Y : y|$ and $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$. If we assumed $|X : x| \sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$, then it would mean that, with respect to both $\mathbb{D}_1$ and $\mathbb{D}_2$, any B^c containing $|X : x|$ must also contain $|Y : y|$, which would contradict $|X : x| \sqsubseteq_{\mathbb{D}_1} |Y : y|$ and $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$. On the other hand, if we assumed $|X : x| \not\sqsubset_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$, then it would mean that, with respect to both $\mathbb{D}_1$ and $\mathbb{D}_2$, every B^c containing $|X : x|$ would not contain $|Y : y|$. This situation would contradict $|X : x| \sqsubseteq_{\mathbb{D}_1} |Y : y|$. Since "$\sqsubset$", "$\sqsubseteq$", and "$\not\sqsubset$" are exhaustive, we must have $|X : x| \sqsubseteq_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$. ■

The cases in Proposition 4.1 above are exhaustive and mutually exclusive. They cover all the possible combinations of $|X : x| \mapsto_{\mathbb{D}_1} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y|$. An important property of the proposition is that its "then" parts contain a definite choice relation. Thus, given any pair of choices with their relations identified separately from

$\mathbb{D}_1$ and $\mathbb{D}_2$, Proposition 4.1 can be used to *automatically* deduce the choice relation with respect to both $\mathbb{D}_1$ and $\mathbb{D}_2$ without any further manual definition. [4]

Note that, when $\mathbb{D}_1 \subsetneq \mathbb{D}_2$, we have $\mathbb{D}_1 \cup \mathbb{D}_2 = \mathbb{D}_2$. Furthermore, the conditions ($|X : x| \sqsubset_{\mathbb{D}_1} |Y : y|$ and $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$) in (e) and ($|X : x| \sqsubseteq_{\mathbb{D}_1} |Y : y|$ and $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y|$) in (f) cannot occur together. It follows immediately from the remaining four cases in Proposition 4.1 that we need only consider the choice relation $|X : x| \mapsto_{\mathbb{D}_2} |Y : y|$ when $\mathbb{D}_1 \subsetneq \mathbb{D}_2$. The choice relation $|X : x| \mapsto_{\mathbb{D}_1} |Y : y|$ can be completely ignored.

In addition to the above proposition, we need the following new definition:

Definition 4.2 (Header and Trailer Choices in a Choice Relation)

Given any choice relation $|X : x| \mapsto_{\mathbb{D}} |Y : y|$, $|X : x|$ and $|Y : y|$ are referred to as the ***header*** *and* ***trailer choices*** *(with respect to $|X : x| \mapsto_{\mathbb{D}} |Y : y|$), respectively.*

Example 4.1 (Header and Trailer Choices in a Choice Relation)

Consider the choice relation |Length of LIST: $= 1$| $\sqsubset$ |Status of LIST: Exists| in Example 2.1. With respect to this relation, |Length of LIST: $= 1$| is the header choice and |Status of LIST: Exists| is the trailer choice. ■

We present below an algorithm (integrate_table) for merging two or more τ_i's using Proposition 4.1. The algorithm has the following assumptions:

(a) Every τ_i involves at least two categories.

(b) Before integrate_table starts, all the choice relations in every τ_i have been determined.

[4] For the rest of the thesis, an *automatically* deduced choice relation is simply referred to as a ***deduced*** choice relation. Similarly, a *manually* defined choice relation is simply referred to as a ***defined*** choice relation.

(c) Problematic categories and choices (including overlapping choices) do not exist *within* individual τ_i's.

Assumption (c) can be achieved in the following ways:

- By using the definitions of problematic categories and choices as presented in Section 3.1.3 and the checklist in Section 3.1.5 [15], the software tester can detect and correct the mistakes before integrate_table starts. We strongly recommend this practice when categories and choices are identified from a $\mathbb{C}_i$ in an *ad hoc* manner due to the absence of any appropriate identification algorithm (especially when $\mathbb{C}_i$ is written in a narrative language without a well defined syntax).

- If appropriate identification algorithms (such as construct_table presented in Section 4.2.2) are available, the software tester can use these algorithms to identify categories and choices systematically from $\mathbb{C}_i$'s. In this case, the algorithms should avoid the occurrence of problematic categories and choices (with respect to each individual $\mathbb{C}_i$).

An Algorithm (integrate_table) for Merging Preliminary Choice Relation Tables

Suppose we have n (where $n \geq 2$) preliminary choice relation tables τ_1, τ_2, ..., τ_n to be merged. Let $\mathbb{C}_i$ (where $1 \leq i \leq n$) be any specification component, τ_i be its corresponding preliminary choice relation table, $\mathbb{S}$ be the entire set of $\mathbb{C}_i$'s (that is, $\mathbb{S} = \{\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_n\}$), and $\mathbb{D}$ be any nonempty subset of $\mathbb{S}$.

/* We shall use a linked list L and its associated linked lists to store the result of merging τ_1, τ_2, ..., τ_n. Each element m_j (where $j \geq 1$) of a nonempty L points to an associated, nonempty linked list (denoted by LL_{m_j}). Each element of a nonempty LL_{m_j} stores a choice relation for a pair of choices. Note that each nonempty LL_{m_j} is used to store the choice

relations determined with respect to a given $\mathbb{D}$. Let $\mathbb{D}(LL_{m_j})$ denote the set of specification components used to construct LL_{m_j}. Given any two nonempty LL_{m_j} and LL_{m_k} such that $j \neq k$, we would have $\mathbb{D}(LL_{m_j}) \neq \mathbb{D}(LL_{m_k})$. In addition, let $N(L)$ denote the number of elements m_j's of the linked list L. Thus, $N(L)$ also represents the number of associated linked lists LL_{m_j}'s. */

(1) Initialization of Lists of Choice Relations

initialize L as an empty linked list;

store the contents of τ_1 in LL_{m_1}, with each element of LL_{m_1} storing a choice relation in τ_1 such that $[X] \neq [Y]$;

store the header address of LL_{m_1} in m_1 of L;

/* Note that $\mathbb{D}(LL_{m_1}) = \{\mathbb{C}_1\}$ and $N(L) = 1$ immediately after step (1). In τ_1, the choice relations whose header and trailer choices belong to the *same* categories need not be stored in the associated linked lists. This is because, by Definition 2.4 and/or Corollary 2.1, the relational operator for $|X : x| \mapsto_{\{\mathbb{C}_1\}} |X : x|$ and $|X : x_1| \mapsto_{\{\mathbb{C}_1\}} |X : x_2|$ (where $|X : x_1|$ and $|X : x_2|$ are nonoverlapping) must be "$\sqsubset$" and "$\not\sqsubset$", respectively. Thus, such choice relations can be automatically deduced in the next algorithm refine_table to be introduced in Section 4.3.2 later. By similar reasons, in step (2) below when other τ_i's (where $2 \leq i \leq n$) are processed, the choice relations whose header and trailer choices belong to the *same* categories would not be stored in the associated linked lists. */

(2) Integration of Preliminary Choice Relation Tables

$i := 2$;

while $i \leq n$ **do** /* process τ_2 to τ_n */

while there exists any unprocessed choice relation $|X : x| \mapsto_{\{\mathbb{C}_i\}} |Y : y|$ in τ_i
such that $[X] \neq [Y]$ **do**

select an unprocessed choice relation $|X : x| \mapsto_{\{\mathbb{C}_i\}} |Y : y|$ in τ_i
such that $[X] \neq [Y]$;

found_flg := "N";

$j := 1$;

while $j \leq N(L)$ **and** *found_flg* = "N" **do**

/* search through the associated linked lists */

if LL_{m_j} contains $|X : x| \mapsto_{\mathbb{D}(LL_{m_j})} |Y : y|$ **then**

/* There does not exist more than one associated linked list containing a choice relation between $[X]$ and $[Y]$. Furthermore, if such associated linked list exists, the choice relation between $[X]$ and $[Y]$ will be stored only once there. */

found_flg := "Y";

use Proposition 4.1 to deduce $|X : x| \mapsto_{\mathbb{D}} |Y : y|$
from $|X : x| \mapsto_{\{\mathbb{C}_i\}} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}(LL_{m_j})} |Y : y|$,
where $\mathbb{D} = \{\mathbb{C}_i\} \cup \mathbb{D}(LL_{m_j})$;

if there does not exist any LL_{m_k} (where $k \geq 1$ and $k \neq j$)
such that $\mathbb{D}(LL_{m_k}) = \mathbb{D}$ **then**

initialize LL_{m_k} as an empty linked list;

append the header address of LL_{m_k} to L;

end_if;

append the newly deduced $|X : x| \mapsto_{\mathbb{D}} |Y : y|$ to LL_{m_k};

delete $|X : x| \mapsto_{\mathbb{D}(LL_{m_j})} |Y : y|$ from LL_{m_j};

else

$j := j + 1$;

end_if;

end_while;

if *found_flg* := "N" **then**

/* no associated linked list containing a choice relation between

$[X]$ and $[Y]$ */

if there does not exist any LL_{m_p} (where $p \geq 1$)

such that $\mathbb{D}(LL_{m_p}) = \{\mathbb{C}_i\}$ **then**

initialize LL_{m_p} as an empty linked list;

append the header address of LL_{m_p} to L;

end_if;

append $|X : x| \mapsto_{\{\mathbb{C}_i\}} |Y : y|$ to LL_{m_p};

end_if;

end_while;

while there exists any empty LL_{m_q} **do**

/* delete any empty associated linked list */

delete LL_{m_q};

delete the header address of LL_{m_q} from L;

end_do;

$i := i + 1$;

end_while;

On completion of the algorithm integrate_table, the result of merging τ_1, τ_2, ..., τ_n is stored in L and its associated linked lists. Example 4.2 outlines the major steps of integrate_table, and the structures of the linked lists:

Example 4.2 (Merging Preliminary Choice Relation Tables)
Suppose there are two preliminary choice relation tables τ_1 and τ_2 to be merged. Tables 4.1 and 4.2 show the choice relations stored in τ_1 and τ_2, respectively. Figure 4.1(a) shows the structures of L and its associated LL_{m_1} after executing step (1) of integrate_table, whereas Figure 4.1(b) shows the structures of L and its associated LL_{m_i}'s upon the completion of integrate_table. In Figures 4.1(a) and 4.1(b), $LL_{m_{i,j}}$ (where $i, j \geq 1$) denotes any element of the associated linked list LL_{m_i}. In addition, Table 4.3 shows the choice relations stored in LL_{m_1} in Figure 4.1(a), and Table 4.4 shows the choice relations stored in the three associated linked lists in Figure 4.1(b).

Readers may note that, *within* each associated linked list in Figures 4.1(a) and 4.1(b) (also see Tables 4.3 and 4.4), all the choice relations are determined with respect to the same $\mathbb{D}$. Furthermore, all the choice relations stored in LL_{m_3} (see Figure 4.1(b) and Table 4.4) are deduced in step (2)(a) of integrate_table using Proposition 4.1. ■

4.3.2 Step (3.2) of DESSERT

The integrate_table algorithm is good enough to solve problem (a) as mentioned earlier in the beginning of Section 4.3. Problem (b), however, may still exist after executing integrate_table. Here we discuss our solution to this problem (after problems (a) and (b) have been solved, problem (c) will disappear automatically).

Let us first focus on problem (b) and consider a hypothetical scenario. Suppose $\mathbb{S}$ is decomposed into $\mathbb{C}_1$ and $\mathbb{C}_2$ (where $\mathbb{C}_1 \neq \mathbb{C}_2$) from which their respective τ_1 and τ_2 are

	$\|X : x_1\|$	$\|X : x_2\|$	$\|Y : y_1\|$	$\|Y : y_2\|$	$\|Z : z_1\|$	$\|Z : z_2\|$
$\|X : x_1\|$	$\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\|X : x_2\|$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\|Y : y_1\|$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\|Y : y_2\|$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\|Z : z_1\|$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\|Z : z_2\|$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\sqsubset_{\{\mathbb{C}_1\}}$

Table 4.1: Preliminary Choice Relation Table τ_1

	$\|W : w_1\|$	$\|W : w_2\|$	$\|X : x_1\|$	$\|X : x_2\|$	$\|Y : y_1\|$	$\|Y : y_3\|$	$\|Y : y_4\|$
$\|W : w_1\|$	$\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$
$\|W : w_2\|$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$
$\|X : x_1\|$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$
$\|X : x_2\|$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$
$\|Y : y_1\|$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$
$\|Y : y_3\|$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$
$\|Y : y_4\|$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsupset\!\!\!\!\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$

Table 4.2: Preliminary Choice Relation Table τ_2

constructed. First, distinct categories $[X]$, $[Y]$, and $[Z]$, and their associated choices are identified from $\mathbb{C}_1$. Thereafter, the choice relations among these choices are determined and captured in τ_1 as shown in Table 4.1. Later, categories $[W]$, $[X]$, and $[Y]$, and their associated choices are identified from $\mathbb{C}_2$. Consequently, the choice relations among these choices are determined and captured in τ_2 as shown in Table 4.2. We assume that overlapping choices do not exist *within* τ_1 and τ_2 individually.

Suppose $|Y : y_2|$ in τ_1 overlaps with $|Y : y_3|$ and $|Y : y_4|$ in τ_2 such that $|Y : y_2| = |Y : y_3| \cup |Y : y_4|$. Since $|Y : y_2|$ only exists in τ_1 (but not τ_2), whereas $|Y : y_3|$ and $|Y : y_4|$ only exist in τ_2 (but not τ_1), the software tester may not be aware of these overlapping

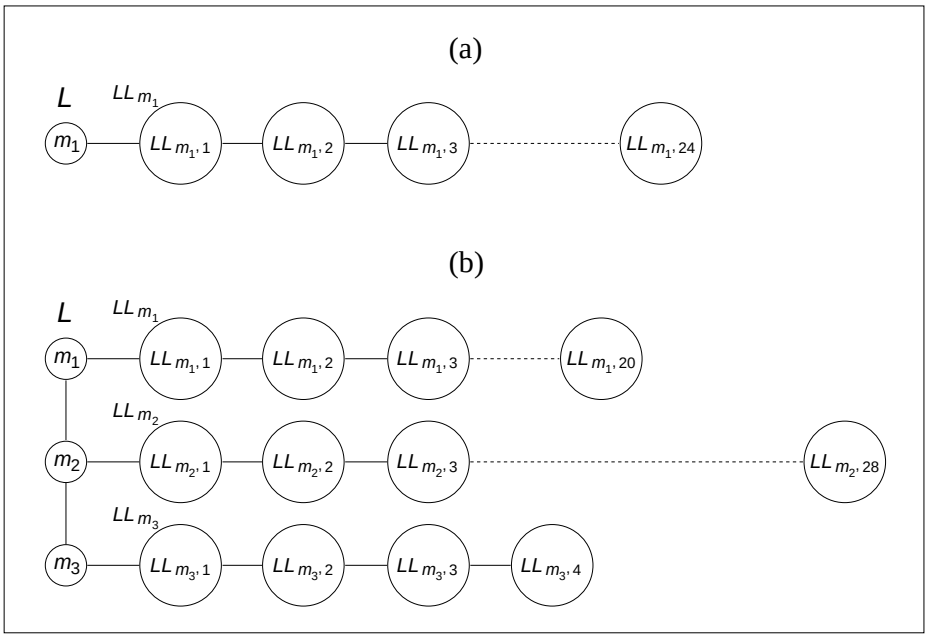


Figure 4.1: Structures of L and its Associated LL_{m_i}'s During and After the Execution of **integrate_table**

choices when constructing τ_1 and τ_2 separately. Note that this hypothetical case involves only one category (that is, $[Y]$) with overlapping choices. Without doubt, the case will become more complicated if $[W]$, $[X]$, and $[Z]$ also contain overlapping choices.

To solve the above problem, a straightforward approach is to replace $|Y : y_2|$ in τ_1 by $|Y : y_3|$ and $|Y : y_4|$, and to *manually* define new choice relations involving $|Y : y_3|$ and $|Y : y_4|$ in τ_1 after replacement. We do not, however, recommend such an approach because the software tester needs to put extra effort in defining the new replacement choice relations, which would mean that the previous effort spent on determining numerous choice relations in τ_1 is wasted (about 31% of the choice relations are affected). It will be desirable if there exists a refinement mechanism which will automatically deduce the new replacement choices (applicable in some other cases) and their choice relations as far as possible.

With this need in mind we have developed a refinement mechanism for overlapping

Associated Linked List of L	**Elements in Associated Linked List**	**Choice Relations Stored in Associated Linked List**	**Elements in Associated Linked List**	**Choice Relations Stored in Associated Linked List**
LL_{m_1}	$LL_{m_1,1}$	$\|X : x_1\| \sqsubset_{\{\mathbb{C}_1\}} \|Y : y_1\|$	$LL_{m_1,13}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_1\|$
	$LL_{m_1,2}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$	$LL_{m_1,14}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_2\|$
	$LL_{m_1,3}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$	$LL_{m_1,15}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$
	$LL_{m_1,4}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$	$LL_{m_1,16}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$
	$LL_{m_1,5}$	$\|X : x_2\| \sqsubset_{\{\mathbb{C}_1\}} \|Y : y_1\|$	$LL_{m_1,17}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_1\|$
	$LL_{m_1,6}$	$\|Y : x_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$	$LL_{m_1,18}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_2\|$
	$LL_{m_1,7}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$	$LL_{m_1,19}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_1\|$
	$LL_{m_1,8}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$	$LL_{m_1,20}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$
	$LL_{m_1,9}$	$\|Y : y_1\| \sqsupset_{\{\mathbb{C}_1\}} \|X : x_1\|$	$LL_{m_1,21}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_1\|$
	$LL_{m_1,10}$	$\|Y : y_1\| \sqsupset_{\{\mathbb{C}_1\}} \|X : x_2\|$	$LL_{m_1,22}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_2\|$
	$LL_{m_1,11}$	$\|Y : y_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$	$LL_{m_1,23}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_1\|$
	$LL_{m_1,12}$	$\|Y : y_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$	$LL_{m_1,24}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$

Table 4.3: Choice Relations Stored in Associated Linked Lists in Figure 4.1(a)

choices and their choice relations. Thereafter, the refined choices and their relations will be processed by CHOC'LATE for test case generation. Before we give the details of our refinement algorithm, we first introduce the following new definitions, corollaries, lemmas, and propositions.

Definition 4.3 (Set of Test Cases Related to a Choice)

Let TC denote the set of all test cases. Given any choice $|X : x|$, we define the ***set of test cases related to*** *$|X : x|$ as $TC(|X : x|) = \{tc \in TC : \ v \in tc \text{ for some } v \in |X : x|\}$.*

Immediately from Definitions 2.1 to 2.3, if $|X : x|$ is valid, $TC(|X : x|) \neq \emptyset$.

Example 4.3 (Set of Test Cases Related to a Choice)

Consider three choices |Integer $M := 1$|, |Integer $N : 5 \leq N \leq 7$|, and |Integer $N : 25 \leq N \leq 26$|. Suppose $TF(|\text{Integer } M := 1|) = \{B_1^c, \ B_2^c\}$, where:

Associated Linked Lists of L	Elements in Associated Linked Lists	Choice Relations Stored in Associated Linked Lists	Associated Linked Lists of L	Elements in Associated Linked Lists	Choice Relations Stored in Associated Linked Lists
LL_{m_1}	$LL_{m_1,1}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$	LL_{m_2}	$LL_{m_2,7}$	$\|W : w_2\| \not\sqsubset_{\{\mathbb{C}_2\}} \|X : x_2\|$
	$LL_{m_1,2}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$		$LL_{m_2,8}$	$\|W : w_2\| \not\sqsubset_{\{\mathbb{C}_2\}} \|Y : y_1\|$
	$LL_{m_1,3}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$		$LL_{m_2,9}$	$\|W : w_2\| \not\sqsubset_{\{\mathbb{C}_2\}} \|Y : y_3\|$
	$LL_{m_1,4}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$		$LL_{m_2,10}$	$\|W : w_2\| \sqsubset_{\{\mathbb{C}_2\}} \|Y : y_4\|$
	$LL_{m_1,5}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$		$LL_{m_2,11}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_1\|$
	$LL_{m_1,6}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$		$LL_{m_2,12}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_2\|$
	$LL_{m_1,7}$	$\|Y : y_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$		$LL_{m_2,13}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|Y : y_3\|$
	$LL_{m_1,8}$	$\|Y : y_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$		$LL_{m_2,14}$	$\|X : x_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|Y : y_4\|$
	$LL_{m_1,9}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_1\|$		$LL_{m_2,15}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_1\|$
	$LL_{m_1,10}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_2\|$		$LL_{m_2,16}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_2\|$
	$LL_{m_1,11}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_1\|$		$LL_{m_2,17}$	$\|X : x_2\| \not\mathbb{C}_{\{\mathbb{C}_2\}} \|Y : y_3\|$
	$LL_{m_1,12}$	$\|Y : y_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Z : z_2\|$		$LL_{m_2,18}$	$\|X : x_2\| \not\sqsubset_{\{\mathbb{C}_2\}} \|Y : y_4\|$
	$LL_{m_1,13}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_1\|$		$LL_{m_2,19}$	$\|Y : y_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_1\|$
	$LL_{m_1,14}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_2\|$		$LL_{m_2,20}$	$\|Y : y_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_2\|$
	$LL_{m_1,15}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_1\|$		$LL_{m_2,21}$	$\|Y : y_3\| \not\mathbb{C}_{\{\mathbb{C}_2\}} \|W : w_1\|$
	$LL_{m_1,16}$	$\|Z : z_1\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$		$LL_{m_2,22}$	$\|Y : y_3\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_2\|$
	$LL_{m_1,17}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_1\|$		$LL_{m_2,23}$	$\|Y : y_3\| \not\sqsubset_{\{\mathbb{C}_2\}} \|X : x_1\|$
	$LL_{m_1,18}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|X : x_2\|$		$LL_{m_2,24}$	$\|Y : y_3\| \not\mathbb{C}_{\{\mathbb{C}_2\}} \|X : x_2\|$
	$LL_{m_1,19}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_1\|$		$LL_{m_2,25}$	$\|Y : y_4\| \not\sqsubset_{\{\mathbb{C}_2\}} \|W : w_1\|$
	$LL_{m_1,20}$	$\|Z : z_2\| \not\sqsubset_{\{\mathbb{C}_1\}} \|Y : y_2\|$		$LL_{m_2,26}$	$\|Y : y_4\| \not\mathbb{C}_{\{\mathbb{C}_2\}} \|W : w_2\|$
LL_{m_2}	$LL_{m_2,1}$	$\|W : w_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|X : x_1\|$		$LL_{m_2,27}$	$\|Y : y_4\| \not\sqsubset_{\{\mathbb{C}_2\}} \|X : x_1\|$
	$LL_{m_2,2}$	$\|W : w_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|X : x_2\|$		$LL_{m_2,28}$	$\|Y : y_4\| \not\sqsubset_{\{\mathbb{C}_2\}} \|X : x_2\|$
	$LL_{m_2,3}$	$\|W : w_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|Y : y_1\|$	LL_{m_3}	$LL_{m_3,1}$	$\|X : x_1\| \not\mathbb{C}_{\{\mathbb{C}_1,\mathbb{C}_2\}} \|Y : y_1\|$
	$LL_{m_2,4}$	$\|W : w_1\| \not\mathbb{C}_{\{\mathbb{C}_2\}} \|Y : y_3\|$		$LL_{m_3,2}$	$\|X : x_2\| \not\mathbb{C}_{\{\mathbb{C}_1,\mathbb{C}_2\}} \|Y : y_1\|$
	$LL_{m_2,5}$	$\|W : w_1\| \not\sqsubset_{\{\mathbb{C}_2\}} \|Y : y_4\|$		$LL_{m_3,3}$	$\|Y : y_1\| \not\mathbb{C}_{\{\mathbb{C}_1,\mathbb{C}_2\}} \|X : x_1\|$
	$LL_{m_2,6}$	$\|W : w_2\| \not\sqsubset_{\{\mathbb{C}_2\}} \|X : x_1\|$		$LL_{m_3,4}$	$\|Y : y_1\| \not\mathbb{C}_{\{\mathbb{C}_1,\mathbb{C}_2\}} \|X : x_2\|$

Table 4.4: Choice Relations Stored in Associated Linked Lists in Figure 4.1(b)

- $B_1^c = \{|\text{Integer } M := 1|,\ |\text{Integer } N : 5 \leq N \leq 7|\}$, and
- $B_2^c = \{|\text{Integer } M := 1|,\ |\text{Integer } N := 25 \leq N \leq 26|\}$.

In this case, $TC(|\text{Integer } M := 1|) = \{tc_1,\ tc_2,\ tc_3,\ tc_4,\ tc_5\}$, where:

- $tc_1 = \{M = 1,\ N = 5\}$,
- $tc_2 = \{M = 1,\ N = 6\}$,
- $tc_3 = \{M = 1,\ N = 7\}$,
- $tc_4 = \{M = 1,\ N = 25\}$, and

- $tc_5 = \{M = 1,\ N = 26\}$. ■

Immediately from Definitions 2.1 and 4.3, we have the following corollary:

Corollary 4.1 (Element of a Choice in a Test Case)

Given any choice $|X : x|$ *and any test case* $tc \in TC(|X : x|)$*, tc contains one and only one element* $v \in |X : x|$.

Corollary 4.1 further results in Corollary 4.2 below:

Corollary 4.2 (Overlapping and Nonoverlapping Choices and their Related Test Cases)

For any choices $|X : x_1|$ *and* $|X : x_2|$*:*

(*a*) $\big(|X : x_1| \cap |X : x_2| \neq \emptyset$*,* $|X : x_1| \not\subseteq |X : x_2|$*, and* $|X : x_2| \not\subseteq |X : x_1|\big)$ *if and only if* $\big(TC(|X : x_1|) \cap TC(|X : x_2|) \neq \emptyset$*,* $TC(|X : x_1|) \not\subseteq TC(|X : x_2|)$*, and* $TC(|X : x_2|) \not\subseteq TC(|X : x_1|)\big)$.

(*b*) $|X : x_1| \subsetneq |X : x_2|$ *if and only if* $TC(|X : x_1|) \subsetneq TC(|X : x_2|)$.

(*c*) $|X : x_1| \cap |X : x_2| = \emptyset$ *if and only if* $TC(|X : x_1|) \cap TC(|X : x_2|) = \emptyset$.

Lemma 4.1 (Choice Relations and Test Cases)

For any choices $|X : x|$ *and* $|Y : y|$*:*

(*a*) $|X : x| \sqsubset |Y : y|$ *if and only if* $TC(|X : x|) \subseteq TC(|Y : y|)$.

(*b*) $|X : x| \sqsubsetneq |Y : y|$ *if and only if* $TC(|X : x|) \not\subseteq TC(|Y : y|)$ *and* $TC(|X : x|) \cap TC(|Y : y|) \neq \emptyset$.

(*c*) $|X : x| \not\sqsubset |Y : y|$ *if and only if* $TC(|X : x|) \cap TC(|Y : y|) = \emptyset$.

Proof of Lemma 4.1 (Choice Relations and Test Cases)

We need only prove parts (a) and (c) of the lemma. Part (b) will follow immediately.

(a) For any choices $|X : x|$ and $|Y : y|$, by Definition 2.4(a), $\big(|X : x| \sqsubset |Y : y|\big) \Leftrightarrow \big(TF(|X : x|) \subseteq TF(|Y : y|)\big) \Leftrightarrow \big($for any B^c, $|X : x| \in B^c$ implies $|Y : y| \in B^c\big) \Leftrightarrow \big($for any tc, $v \in |X : x|$ for some $v \in tc$ implies $v' \in |Y : y|$ for some $v' \in tc\big) \Leftrightarrow \big(TC(|X : x|) \subseteq TC(|Y : y|)\big)$.

(c) For any choices $|X : x|$ and $|Y : y|$, by Definition 2.4(c), $\big(|X : x| \not\sqsubset |Y : y|\big) \Leftrightarrow \big(TF(|X : x|) \cap\ TF(|Y : y|) = \emptyset\big) \Leftrightarrow \big($there does not exist any B^c such that $|X : x|,\ |Y : y| \in B^c\big) \Leftrightarrow \big($there does not exist any tc such that $v \in |X : x|$ for some $v \in tc$ and $v' \in |Y : y|$ for some $v' \in tc\big) \Leftrightarrow \big(TC(|X : x|) \cap TC(|Y : y|) = \emptyset\big)$. ■

Lemma 4.2 (Overlapping Choices and Transitive Property of Choice Relations)
Let (i) $[X]$ *and* $[Y]$ *be distinct categories,* (ii) $|X : x_1|$, $|X : x_2|$, *and* $|Y : y|$ *be choices, and* (iii) $|X : x_1| \subsetneq |X : x_2|$.

(a) *If* $|X : x_2| \sqsubset |Y : y|$, *then* $|X : x_1| \sqsubset |Y : y|$.

(b) *If* $|X : x_2| \not\sqsubset |Y : y|$, *then* $|X : x_1| \not\sqsubset |Y : y|$.

Proof of Lemma 4.2 (Overlapping Choices and Transitive Property of Choice Relations)
Given any distinct categories $[X]$ and $[Y]$, and any choices $|X : x_1|$, $|X : x_2|$, and $|Y : y|$ such that $|X : x_1| \subsetneq |X : x_2|$:

(a) Suppose $|X : x_2| \sqsubset |Y : y|$. By Lemma 4.1(a), we have $TC(|X : x_2|) \subseteq TC(|Y : y|)$. Since $|X : x_1| \subsetneq |X : x_2|$, by Corollary 4.2(b), we also have $TC(|X : x_1|) \subsetneq TC(|X : x_2|)$. Thus, we must have $TC(|X : x_1|) \subsetneq TC(|Y : y|)$, or $|X : x_1| \sqsubset |Y : y|$ by Lemma 4.1(a).

(b) Suppose $|X : x_2| \not\sqsubset |Y : y|$. By Lemma 4.1(c), we have $TC(|X : x_2|) \cap TC(|Y : y|) = \emptyset$. Since $|X : x_1| \subsetneq |X : x_2|$, by Corollary 4.2(b), we also have $TC(|X : x_1|) \subsetneq$

$TC(|X : x_2|)$. Thus, we must have $TC(|X : x_1|) \cap TC(|Y : y|) = \emptyset$, or $|X : x_1| \not\sqsubset |Y : y|$ by Lemma 4.1(c). ■

Proposition 4.2 (First Type of Overlapping Header Choices)

Let (*i*) $[X]$ *and* $[Y]$ *be distinct categories,* (*ii*) $|X : x_1|$, $|X : x_2|$, *and* $|Y : y|$ *be choices,* (*iii*) $|X : x_1| \cap |X : x_2| \neq \emptyset$, (*iv*) $|X : x_1| \not\subseteq |X : x_2|$, *and* (*v*) $|X : x_2| \not\subseteq |X : x_1|$. *Furthermore, let* $|X : x'_1| = |X : x_1| \setminus |X : x_2|$ *and* $|X : x_3| = |X : x_1| \cap |X : x_2|$.

(*a*) *If* $|X : x_1| \sqsubset |Y : y|$, *then* $|X : x'_1| \sqsubset |Y : y|$, $|Y : y| \boxplus |X : x'_1|$, $|X : x_3| \sqsubset |Y : y|$, *and* $|Y : y| \boxplus |X : x_3|$.

(*b*) *If* $|X : x_1| \boxplus |Y : y|$, *then any relational operator for* $|X : x'_1| \mapsto |Y : y|$, $|Y : y| \mapsto |X : x'_1|$, $|X : x_3| \mapsto |Y : y|$, *and* $|Y : y| \mapsto |X : x_3|$ *is possible.*

(*c*) *If* $|X : x_1| \not\sqsubset |Y : y|$, *then* $|X : x'_1| \not\sqsubset |Y : y|$, $|Y : y| \not\sqsubset |X : x'_1|$, $|X : x_3| \not\sqsubset |Y : y|$, *and* $|Y : y| \not\sqsubset |X : x_3|$.

Proof of Proposition 4.2 (First Type of Overlapping Header Choices)

Suppose (i) $[X]$ and $[Y]$ are any distinct categories, (ii) $|X : x_1|$, $|X : x_2|$, and $|Y : y|$ are any choices, (iii) $|X : x_1| \cap |X : x_2| \neq \emptyset$, (iv) $|X : x_1| \not\subseteq |X : x_2|$, and (v) $|X : x_2| \not\subseteq |X : x_1|$. Furthermore, suppose $|X : x'_1| = |X : x_1| \setminus |X : x_2|$ and $|X : x_3| = |X : x_1| \cap |X : x_2|$. In this case, we have $|X : x'_1| \subsetneq |X : x_1|$ and $|X : x_3| \subsetneq |X : x_1|$.

(a) Suppose $|X : x_1| \sqsubset |Y : y|$. By Lemma 4.2(a), we have $|X : x'_1| \sqsubset |Y : y|$ and $|X : x_3| \sqsubset |Y : y|$.

Since $|X : x'_1| \sqsubset |Y : y|$, we can also conclude that $|Y : y| \sqsubset |X : x'_1|$ or $|Y : y| \boxplus |X : x'_1|$, according to Corollary 1(a) that was introduced and proved in [21]. [5]

[5] Corollary 1(a) in [21] states that, given any nonoverlapping choices $|X : x|$ and $|Y : y|$, if $|X : x| \sqsubset |Y : y|$, then $|Y : y| \sqsubset |X : x|$ or $|Y : y| \boxplus |X : x|$.

Now, let us assume $|Y : y| \sqsubset |X : x_1'|$. By Lemma 4.1(a), we have $TC(|Y : y|) \subseteq TC(|X : x_1'|)$. Since $|X : x_3| \sqsubset |Y : y|$, we also have $TC(|X : x_3|) \subseteq TC(|Y : y|)$ by the same lemma. Consequently, we can conclude that $TC(|X : x_3|) \subseteq TC(|X : x_1'|)$, which would contradict Corollary 4.2(c) because $|X : x_1'| \cap |X : x_3| = \emptyset$ implies $TC(|X : x_1'|) \cap TC(|X : x_3|) = \emptyset$. This contradiction shows that $|Y : y| \sqsubset |X : x_1'|$ is not feasible. Thus, we must have $|Y : y| \sqsupset\!\!\!\!\!\!\in |X : x_1'|$. Similarly, we can also prove that $|Y : y| \in\!\!\!\!\!\!\sqsupset |X : x_3|$.

(b) Suppose $|X : x_1| \in\!\!\!\!\!\!\sqsupset |Y : y|$. By Corollary 4.2(a) and Lemma 4.1, the relations among $TC(|X : x_1|)$, $TC(|X : x_2|)$, and $TC(|Y : y|)$ may take one of the possible forms depicted in Figure 4.2, or some other forms not being shown in the figure. It can be seen from Figure 4.2 that the relational operator for $|X : x_1'| \mapsto |Y : y|$, $|Y : y| \mapsto |X : x_1'|$, $|X : x_3| \mapsto |Y : y|$, and $|Y : y| \mapsto |X : x_3|$ may be "$\sqsubset$", "$\in\!\!\!\!\!\!\sqsupset$", or "$\not\sqsubset$".

(c) Suppose $|X : x_1| \not\sqsubset |Y : y|$. By Lemma 4.2(b), we have $|X : x_1'| \not\sqsubset |Y : y|$ and $|X : x_3| \not\sqsubset |Y : y|$. Following directly from Definition 2.4(c), we can also conclude that $|Y : y| \not\sqsubset |X : x_1'|$ and $|Y : y| \not\sqsubset |X : x_3|$. ■

In Proposition 4.2 above, $|X : x_2|$ still overlaps with $|X : x_3|$. More specifically, we have $|X : x_3| \subsetneq |X : x_2|$. In this case, Proposition 4.3 below applies.

Proposition 4.3 (Second Type of Overlapping Header Choices)
Let: (*i*) $[X]$ *and* $[Y]$ *be distinct categories,* (*ii*) $|X : x_1|$, $|X : x_2|$, *and* $|Y : y|$ *be choices, and* (*iii*) $|X : x_1| \subsetneq |X : x_2|$. *Furthermore, let* $|X : x_2'| = |X : x_2| \setminus |X : x_1|$.

(*a*) *If* $|X : x_2| \sqsubset |Y : y|$, *then* $|X : x_1| \sqsubset |Y : y|$, $|Y : y| \in\!\!\!\!\!\!\sqsupset |X : x_1|$, $|X : x_2'| \sqsubset |Y : y|$, *and* $|Y : y| \in\!\!\!\!\!\!\sqsupset |X : x_2'|$.

(*b*) *If* $|X : x_2| \in\!\!\!\!\!\!\sqsupset |Y : y|$, *then any relational operator for* $|X : x_1| \mapsto |Y : y|$, $|Y : y| \mapsto |X : x_1|$, $|X : x_2'| \mapsto |Y : y|$, *and* $|Y : y| \mapsto |X : x_2'|$ *is possible.*

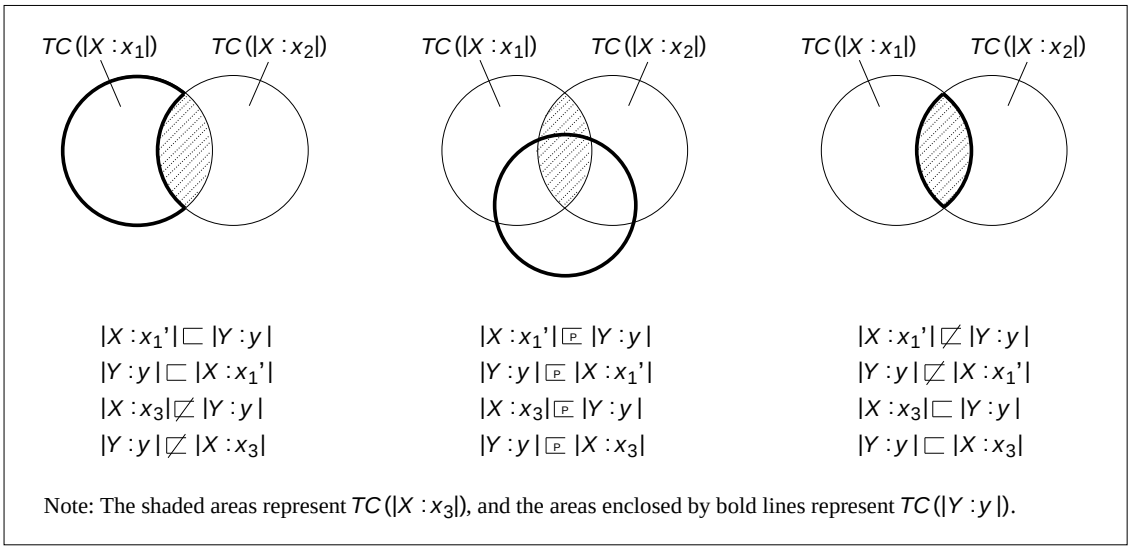


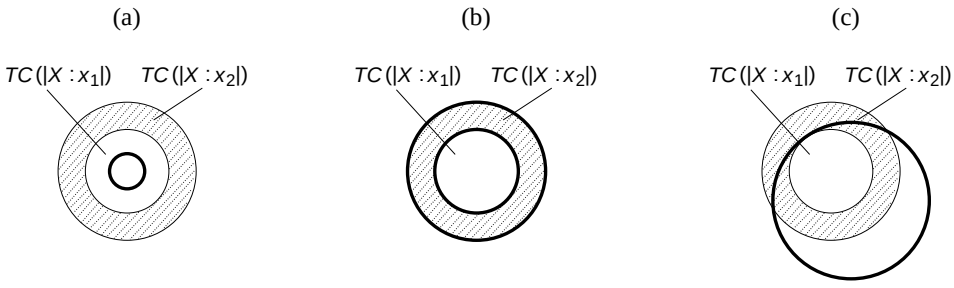
Figure 4.2: Some Possible Relations among $TC(|X : x_1|)$, $TC(|X : x_2|)$, and $TC(|Y : y|)$ When $|X : x_1| \cap |X : x_2| \neq \emptyset$, $|X : x_1| \not\subseteq |X : x_2|$, $|X : x_2| \not\subseteq |X : x_1|$, and $|X : x_1| \sqsubseteq_{\mathbb{P}} |Y : y|$

(*c*) *If* $|X : x_2| \not\sqsubset |Y : y|$, *then* $|X : x_1| \not\sqsubset |Y : y|$, $|Y : y| \not\sqsubset |X : x_1|$, $|X : x_2'| \not\sqsubset |Y : y|$, *and* $|Y : y| \not\sqsubset |X : x_2'|$.

Proof of Proposition 4.3 (Second Type of Overlapping Header Choices)

Suppose (i) $[X]$ and $[Y]$ are any distinct categories, (ii) $|X : x_1|$, $|X : x_2|$, and $|Y : y|$ are any choices, and (iii) $|X : x_1| \subsetneq |X : x_2|$. Furthermore, suppose $|X : x_2'| = |X : x_2| \setminus |X : x_1|$. In this case, we have $|X : x_2'| \subsetneq |X : x_2|$.

(a) Suppose $|X : x_2| \sqsubset |Y : y|$. By Lemma 4.2(a), we have $|X : x_1| \sqsubset |Y : y|$ and $|X : x_2'| \sqsubset |Y : y|$. Since $|X : x_1| \sqsubset |Y : y|$, we can also conclude that $|Y : y| \sqsubset |X : x_1|$ or $|Y : y| \sqsubseteq_{\mathbb{P}} |X : x_1|$, according to Corollary 1(a) that was introduced and proved in [21] (see footnote 5). Now, let us assume $|Y : y| \sqsubset |X : x_1|$. By Lemma 4.1(a), we have $TC(|Y : y|) \subseteq TC(|X : x_1|)$. Since $|X : x_2'| \sqsubset |Y : y|$, we also have $TC(|X : x_2'|) \subseteq TC(|Y : y|)$ by the same lemma. Consequently, we can conclude that $TC(|X : x_2'|) \subseteq TC(|X : x_1|)$, which would contradict Corollary 4.2(c) because $|X : x_1| \cap |X : x_2'| = \emptyset$ implies $TC(|X : x_1|) \cap TC(|X : x_2'|) = \emptyset$. This contradiction shows that $|Y : y| \sqsubset$



(a)	(b)	(c)
$\|X : x_1\| \boxed{\mathbb{P}} \|Y : y\|$	$\|X : x_1\| \not\sqsubset \|Y : y\|$	$\|X : x_1\| \sqsubset \|Y : y\|$
$\|Y : y\| \sqsubset \|X : x_1\|$	$\|Y : y\| \not\sqsubset \|X : x_1\|$	$\|Y : y\| \boxed{\mathbb{P}} \|X : x_1\|$
$\|X : x_2\text{'}\| \not\sqsubset \|Y : y\|$	$\|X : x_2\text{'}\| \sqsubset \|Y : y\|$	$\|X : x_2\text{'}\| \boxed{\mathbb{P}} \|Y : y\|$
$\|Y : y\| \not\sqsubset \|X : x_2\text{'}\|$	$\|Y : y\| \sqsubset \|X : x_2\text{'}\|$	$\|Y : y\| \boxed{\mathbb{P}} \|X : x_2\text{'}\|$

Note: In Figures 5.3(a) and 5.3(c), the inner and outer circles enclosed by thin lines represent $TC(|X : x_1|)$ and $TC(|X : x_2|)$, respectively. Also, in these two figures, the circles enclosed by bold lines represent $TC(|Y : y|)$.
In Figure 5.3(b), the inner and outer circles enclosed by bold lines represent $TC(|X : x_1|)$ and $TC(|X : x_2|)$, respectively; the area between the two bold lines (the shaded area) represents $TC(|Y : y|)$.
In Figures 5.3(a), 5.3(b), and 5.3(c), the shaded areas represent $TC(|X : x_2\text{'}|)$.

Figure 4.3: Some Possible Relations among $TC(|X : x_1|)$, $TC(|X : x_2|)$, and $TC(|Y : y|)$ When $|X : x_1| \subsetneq |X : x_2|$ and $|X : x_2| \boxed{\mathbb{P}} |Y : y|$

$|X : x_1|$ is not feasible. Thus, we must have $|Y : y| \boxed{\mathbb{P}} |X : x_1|$. Similarly, we can also prove that $|Y : y| \boxed{\mathbb{P}} |X : x_2'|$.

(b) Suppose $|X : x_2| \boxed{\mathbb{P}} |Y : y|$. By Corollary 4.2(b) and Lemma 4.1, the relations among $TC(|X : x_1|)$, $TC(|X : x_2|)$, and $TC(|Y : y|)$ may take one of the possible forms depicted in Figure 4.3, or some other forms not being shown in the figure. It can be seen from Figure 4.3 that the relational operator for $|X : x_1| \mapsto |Y : y|$, $|Y : y| \mapsto |X : x_1|$, $|X : x_2'| \mapsto |Y : y|$, and $|Y : y| \mapsto |X : x_2'|$ may be "$\sqsubset$", "$\boxed{\mathbb{P}}$", or "$\not\sqsubset$".

(c) Suppose $|X : x_2| \not\sqsubset |Y : y|$. By Lemma 4.2(b), we have $|X : x_1| \not\sqsubset |Y : y|$ and $|X : x_2'| \not\sqsubset |Y : y|$. Following directly from Definition 2.4(c), we can also conclude that $|Y : y| \not\sqsubset |X : x_1|$ and $|Y : y| \not\sqsubset |X : x_2'|$. ■

Proposition 4.4 (First Type of Overlapping Trailer Choices)

Let: (i) $[X]$ *and* $[Y]$ *be distinct categories,* (ii) $|X : x|$, $|Y : y_1|$, *and* $|Y : y_2|$ *be choices,* (iii) $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, (iv) $|Y : y_1| \not\subseteq |Y : y_2|$, *and* (v) $|Y : y_2| \not\subseteq |Y : y_1|$. *Furthermore, let* $|Y : y_1'| = |Y : y_1| \setminus |Y : y_2|$ *and* $|Y : y_3| = |Y : y_1| \cap |Y : y_2|$.

(a) *If* $|X : x| \sqsubset |Y : y_1|$, *then any relational operator for* $|X : x| \mapsto |Y : y_1'|$, $|Y : y_1'| \mapsto |X : x|$, $|X : x| \mapsto |Y : y_3|$, *and* $|Y : y_3| \mapsto |X : x|$ *is possible.*

(b) *If* $|X : x| \sqsubseteq |Y : y_1|$, *then:*

- $|X : x| \sqsubseteq |Y : y_1'|$ *or* $|X : x| \not\sqsubset |Y : y_1'|$,
- *any relational operator for* $|Y : y_1'| \mapsto |X : x|$ *is possible,*
- $|X : x| \sqsubseteq |Y : y_3|$ *or* $|X : x| \not\sqsubset |Y : y_3|$, *and*
- *any relational operator for* $|Y : y_3| \mapsto |X : x|$ *is possible.*

(c) *If* $|X : x| \not\sqsubset |Y : y_1|$, *then* $|X : x| \not\sqsubset |Y : y_1'|$, $|Y : y_1'| \not\sqsubset |X : x|$, $|X : x| \not\sqsubset |Y : y_3|$, *and* $|Y : y_3| \not\sqsubset |X : x|$.

Proof of Proposition 4.4 (First Type of Overlapping Trailer Choices)

Suppose (i) $[X]$ and $[Y]$ are any distinct categories, (ii) $|X : x|$, $|Y : y_1|$, and $|Y : y_2|$ are any choices, (iii) $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, (iv) $|Y : y_1| \not\subseteq |Y : y_2|$, and (v) $|Y : y_2| \not\subseteq |Y : y_1|$. Furthermore, suppose $|Y : y_1'| = |Y : y_1| \setminus |Y : y_2|$ and $|Y : y_3| = |Y : y_1| \cap |Y : y_2|$. In this case, we have $|Y : y_1'| \subsetneq |Y : y_1|$ and $|Y : y_3| \subsetneq |Y : y_1|$.

(a) Suppose $|X : x| \sqsubset |Y : y_1|$. By Corollary 4.2(a) and Lemma 4.1, the relations among $TC(|X : x|)$, $TC(|Y : y_1|)$, and $TC(|Y : y_2|)$ may take one of the possible forms depicted in Figure 4.4, or some other forms not being shown in the figure. It can be seen from Figure 4.4 that the relational operator for $|X : x| \mapsto |Y : y_1'|$, $|Y : y_1'| \mapsto |X : x|$, $|X : x| \mapsto |Y : y_3|$, and $|Y : y_3| \mapsto |X : x|$ may be "$\sqsubset$", "$\sqsubseteq$", or "$\not\sqsubset$".

(b) Suppose $|X : x| \boxplus |Y : y_1|$. By Lemma 4.1(b), we have $TC(|X : x|) \not\subseteq TC(|Y : y_1|)$. In addition, since $|Y : y'_1| \subsetneq |Y : y_1|$, by Corollary 4.2(b), we have $TC(|Y : y'_1|) \subsetneq TC(|Y : y_1|)$. In turn, we can conclude that $TC(|X : x|) \not\subseteq TC(|Y : y'_1|)$. Now, let us assume $|X : x| \sqsubset |Y : y'_1|$. Here we have a contradiction because $|X : x| \sqsubset |Y : y'_1|$ implies $TC(|X : x|) \subseteq TC(|Y : y'_1|)$ according to Lemma 4.1(a). Thus, we must have $|X : x| \boxplus |Y : y'_1|$ or $|X : x| \not\sqsubset |Y : y'_1|$. Furthermore, these possible choice relations for $|X : x| \mapsto |Y : y'_1|$ cannot be narrowed down any further as is evident from Figure 4.5 (note that the three forms, representing the different combinations of choice relations, in the figure are not exhaustive). Similarly, we can also prove that $|X : x| \boxplus |Y : y_3|$ or $|X : x| \not\sqsubset |Y : y_3|$, and these possible choice relations for $|X : x| \mapsto |Y : y_3|$, as shown in Figure 4.5, cannot be reduced further. Figure 4.5 also shows that any relational operator for $|Y : y'_1| \mapsto |X : x|$ and $|Y : y_3| \mapsto |X : x|$ is possible.

(c) Suppose $|X : x| \not\sqsubset |Y : y_1|$. Following directly from Definition 2.4(c), we can also conclude that $|Y : y_1| \not\sqsubset |X : x|$. By Lemma 4.2(b), we have $|Y : y'_1| \not\sqsubset |X : x|$ and $|Y : y_3| \not\sqsubset |X : x|$, which in turn conclude that $|X : x| \not\sqsubset |Y : y'_1|$ and $|X : x| \not\sqsubset |Y : y_3|$.

■

In Proposition 4.4 above, $|Y : y_2|$ still overlaps with $|Y : y_3|$. More specifically, we have $|Y : y_3| \subsetneq |Y : y_2|$. In this case, Proposition 4.5 below applies.

Proposition 4.5 (Second Type of Overlapping Trailer Choices)

Let: (*i*) $[X]$ *and* $[Y]$ *be distinct categories,* (*ii*) $|X : x|$, $|Y : y_1|$, *and* $|Y : y_2|$ *be choices, and* (*iii*) $|Y : y_1| \subsetneq |Y : y_2|$. *Furthermore, let* $|Y : y'_2| = |Y : y_2| \setminus |Y : y_1|$.

(*a*) *If* $|X : x| \sqsubset |Y : y_2|$, *then any relational operator for* $|X : x| \mapsto |Y : y_1|$, $|Y : y_1| \mapsto |X : x|$, $|X : x| \mapsto |Y : y'_2|$, *and* $|Y : y'_2| \mapsto |X : x|$ *is possible.*

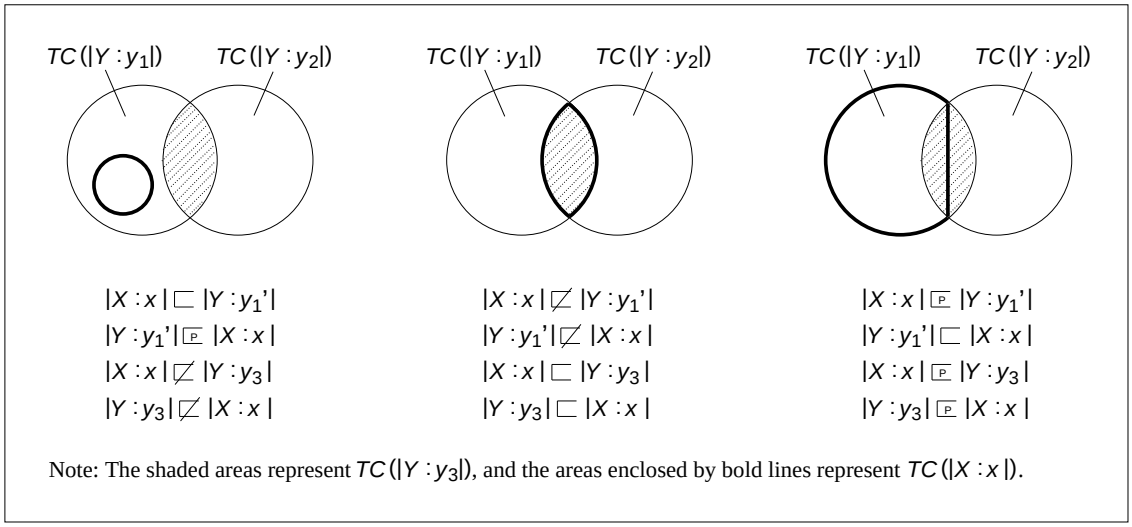


Figure 4.4: Some Possible Relations among $TC(|X : x|)$, $TC(|Y : y_1|)$, and $TC(|Y : y_2|)$ When $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, $|Y : y_1| \not\subseteq |Y : y_2|$, $|Y : y_2| \not\subseteq |Y : y_1|$, and $|X : x| \sqsubset |Y : y_1|$

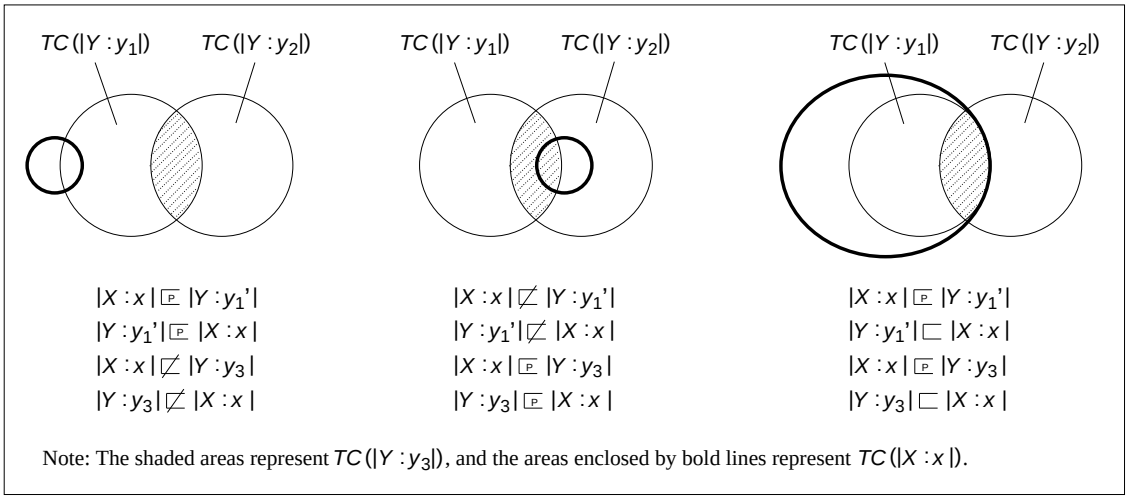


Figure 4.5: Some Possible Relations among $TC(|X : x|)$, $TC(|Y : y_1|)$, and $TC(|Y : y_2|)$ When $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, $|Y : y_1| \not\subseteq |Y : y_2|$, $|Y : y_2| \not\subseteq |Y : y_1|$, and $|X : x| \not\sqsubseteq_P |Y : y_1|$

(*b*) *If* $|X : x| \boxplus |Y : y_2|$*, then:*

- $|X : x| \boxplus |Y : y_1|$ *or* $|X : x| \not\boxplus |Y : y_1|$*,*
- *any relational operator for* $|Y : y_1| \mapsto |X : x|$ *is possible,*
- $|X : x| \boxplus |Y : y_2'|$ *or* $|X : x| \not\boxplus |Y : y_2'|$*, and*
- *any relational operator for* $|Y : y_2'| \mapsto |X : x|$ *is possible.*

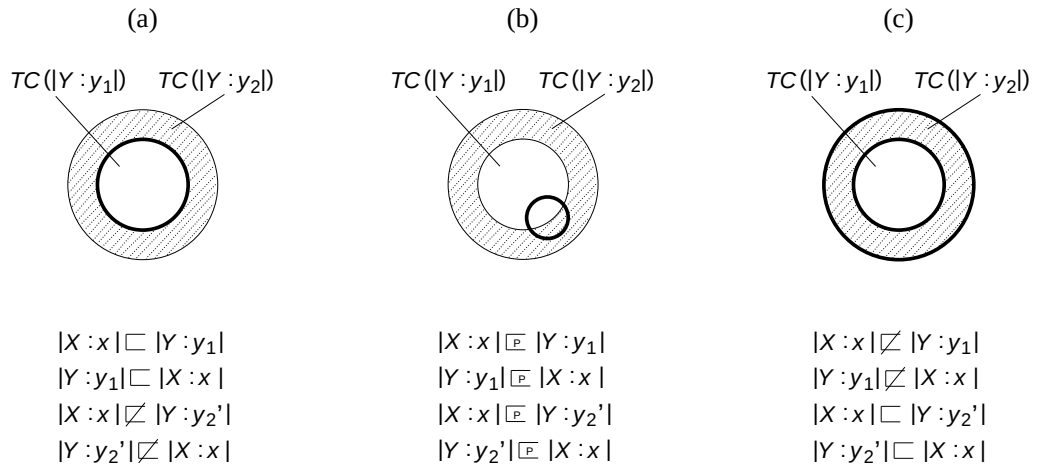
(*c*) *If* $|X : x| \not\boxplus |Y : y_2|$*, then* $|X : x| \not\boxplus |Y : y_1|$, $|Y : y_1| \not\boxplus |X : x|$, $|X : x| \not\boxplus |Y : y_2'|$*, and* $|Y : y_2'| \not\boxplus |X : x|$.

Proof of Proposition 4.5 (Second Type of Overlapping Trailer Choices)

Suppose (i) $[X]$ and $[Y]$ are any distinct categories, (ii) $|X : x|$, $|Y : y_1|$, and $|Y : y_2|$ are any choices, and (iii) $|Y : y_1| \subsetneq |Y : y_2|$. Furthermore, suppose $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$. In this case, we have $|Y : y_2'| \subsetneq |Y : y_2|$.

(a) Suppose $|X : x| \sqsubset |Y : y_2|$. By Corollary 4.2(b) and Lemma 4.1, the relations among $TC(|X : x|)$, $TC(|Y : y_1|)$, and $TC(|Y : y_2|)$ may take one of the possible forms depicted in Figure 4.6, or some other forms not being shown in the figure. It can be seen from Figure 4.6 that the relational operator for $|X : x| \mapsto |Y : y_1|$, $|Y : y_1| \mapsto |X : x|$, $|X : x| \mapsto |Y : y_2'|$, and $|Y : y_2'| \mapsto |X : x|$ may be "$\sqsubset$", "$\boxplus$", or "$\not\boxplus$".

(b) Suppose $|X : x| \boxplus |Y : y_2|$. By Lemma 4.1(b), we have $TC(|X : x|) \not\subseteq TC(|Y : y_2|)$. In addition, since $|Y : y_1| \subsetneq |Y : y_2|$, by Corollary 4.2(b), we have $TC(|Y : y_1|) \subsetneq TC(|Y : y_2|)$. In turn, we can conclude that $TC(|X : x|) \not\subseteq TC(|Y : y_1|)$. Now, let us assume $|X : x| \sqsubset |Y : y_1|$. Here we have a contradiction because $|X : x| \sqsubset |Y : y_1|$ implies $TC(|X : x|) \subseteq TC(|Y : y_1|)$ according to Lemma 4.1(a). Thus, we must have $|X : x| \boxplus |Y : y_1|$ or $|X : x| \not\boxplus |Y : y_1|$. Furthermore, these possible choice relations for $|X : x| \mapsto |Y : y_1|$ cannot be narrowed down any further as is evident



Note: In Figure 5.6(a), the inner circle enclosed by a bold line represents both $TC(|X : x|)$ and $TC(|Y : y_1|)$; the outer circle enclosed by a thin line represents $TC(|Y : y_2|)$.
In Figure 5.6(b), the inner and outer circles enclosed by thin lines represent $TC(|Y : y_1|)$ and $TC(|Y : y_2|)$, respectively; the small circle enclosed by a bold line represents $TC(|X : x|)$.
In Figure 5.6(c), the inner and outer circles enclosed by bold lines represent $TC(|Y : y_1|)$ and $TC(|Y : y_2|)$, respectively; the area between the two bold lines (the shaded area) represents $TC(|X : x|)$.
In Figures 5.6(a), 5.6(b), and 5.6(c), the shaded areas represent $TC(|Y : y_2'|)$.

Figure 4.6: Some Possible Relations among $TC(|X : x|)$, $TC(|Y : y_1|)$, and $TC(|Y : y_2|)$ When $|Y : y_1| \subsetneq |Y : y_2|$ and $|X : x| \sqsubset |Y : y_2|$

from Figure 4.7 (note that the three forms, representing the different combinations of choice relations, in the figure are not exhaustive). Similarly, we can also prove that $|X : x| \sqsubseteq_P |Y : y_2'|$ or $|X : x| \not\sqsubset |Y : y_2'|$, and these possible choice relations for $|X : x| \mapsto |Y : y_2'|$, as shown in Figure 4.7, cannot be reduced further. Figure 4.7 also shows that any relational operator for $|Y : y_1| \mapsto |X : x|$ and $|Y : y_2'| \mapsto |X : x|$ is possible.

(c) Suppose $|X : x| \not\sqsubset |Y : y_2|$. Following directly from Definition 2.4(c), we can also conclude that $|Y : y_2| \not\sqsubset |X : x|$. By Lemma 4.2(b), we have $|Y : y_1| \not\sqsubset |X : x|$ and $|Y : y_2'| \not\sqsubset |X : x|$. ■

Readers may note that Propositions 4.2 to 4.5 can also be applied to choice relations

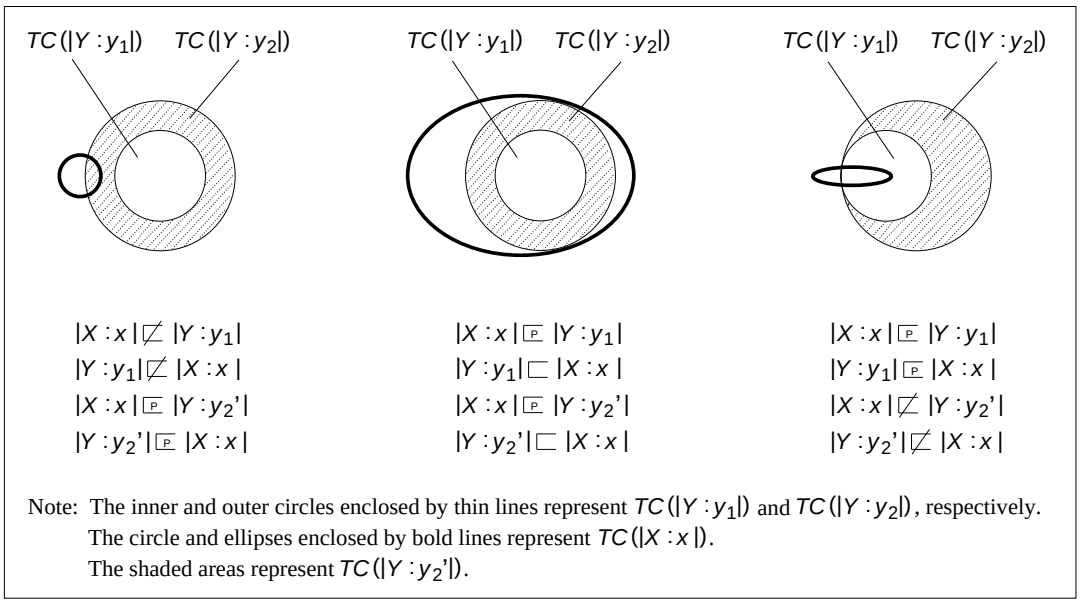


Figure 4.7: Some Possible Relations among $TC(|X : x|)$, $TC(|Y : y_1|)$, and $TC(|Y : y_2|)$ When $|Y : y_1| \subsetneq |Y : y_2|$ and $|X : x| \sqsubseteq_{\mathbb{P}} |Y : y_2|$

determined with respect to any $\mathbb{D}$ other than the entire $\mathbb{S}$. To explain how these four propositions can be *recursively* applied for refining overlapping choices and their relations, we first consider a pair of choice relations involving overlapping choices. Let $\mathbb{D}_1$ and $\mathbb{D}_2$ be two sets of specification components. Suppose we identify a pair of choices $|X : x_1|$ and $|Y : y_1|$ (where $[X] \neq [Y]$) from $\mathbb{D}_1$ and define their relation $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$. Suppose we identify another pair of choices $|X : x_2|$ and $|Y : y_2|$ (where $[X] \neq [Y]$) from $\mathbb{D}_2$ and define their relation $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$. In particular, $|X : x_1|$ may be identical to or overlap with $|X : x_2|$, and $|Y : y_1|$ may be identical to or overlap with $|Y : y_2|$. Table 4.5 lists all the possible combinations of $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ that involve overlapping choices. The rightmost column in the table shows the corresponding refinement rule for each combination. These refinement rules are discussed below:

Refinement Rule 1

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y|$ such that (i) $[X]$ and $[Y]$ are distinct

Combinations	Choice Relations	Overlapping Choices	Corresponding Refinement Rules
1	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y\|$	$\|X : x_1\| \cap \|X : x_2\| \neq \emptyset$, $\|X : x_1\| \not\subseteq \|X : x_2\|$, $\|X : x_2\| \not\subseteq \|X : x_1\|$	1
2	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y\|$	$\|X : x_1\| \subsetneq \|X : x_2\|$	2
3	$\|X : x\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|Y : y_1\| \cap \|Y : y_2\| \neq \emptyset$, $\|Y : y_1\| \not\subseteq \|Y : y_2\|$, $\|Y : y_2\| \not\subseteq \|Y : y_1\|$	3
4	$\|X : x\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|Y : y_1\| \subsetneq \|Y : y_2\|$	4
5	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \cap \|X : x_2\| \neq \emptyset$, $\|X : x_1\| \not\subseteq \|X : x_2\|$, $\|X : x_2\| \not\subseteq \|X : x_1\|$, $\|Y : y_1\| \cap \|Y : y_2\| = \emptyset$	5
6	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \subsetneq \|X : x_2\|$, $\|Y : y_1\| \cap \|Y : y_2\| = \emptyset$	6
7	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \cap \|X : x_2\| = \emptyset$, $\|Y : y_1\| \cap \|Y : y_2\| \neq \emptyset$, $\|Y : y_1\| \not\subseteq \|Y : y_2\|$, $\|Y : y_2\| \not\subseteq \|Y : y_1\|$	7
8	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \cap \|X : x_2\| = \emptyset$, $\|Y : y_1\| \subsetneq \|Y : y_2\|$	8
9	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \cap \|X : x_2\| \neq \emptyset$, $\|X : x_1\| \not\subseteq \|X : x_2\|$, $\|X : x_2\| \not\subseteq \|X : x_1\|$, $\|Y : y_1\| \cap \|Y : y_2\| \neq \emptyset$, $\|Y : y_1\| \not\subseteq \|Y : y_2\|$, $\|Y : y_2\| \not\subseteq \|Y : y_1\|$	9
10	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \cap \|X : x_2\| \neq \emptyset$, $\|X : x_1\| \not\subseteq \|X : x_2\|$, $\|X : x_2\| \not\subseteq \|X : x_1\|$, $\|Y : y_1\| \subsetneq \|Y : y_2\|$	10
11	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \subsetneq \|X : x_2\|$, $\|Y : y_1\| \cap \|Y : y_2\| \neq \emptyset$, $\|Y : y_1\| \not\subseteq \|Y : y_2\|$, $\|Y : y_2\| \not\subseteq \|Y : y_1\|$	11
12	$\|X : x_1\| \mapsto_{\mathbb{D}_1} \|Y : y_1\|$, $\|X : x_2\| \mapsto_{\mathbb{D}_2} \|Y : y_2\|$	$\|X : x_1\| \subsetneq \|X : x_2\|$, $\|Y : y_1\| \subsetneq \|Y : y_2\|$	12

Table 4.5: Possible Combinations of Two Choice Relations Involving Overlapping Choices

categories, (ii) $|X : x_1| \cap |X : x_2| \neq \emptyset$, (iii) $|X : x_1| \not\subseteq |X : x_2|$, and (iv) $|X : x_2| \not\subseteq |X : x_1|$. Let $|X : x_1'| = |X : x_1| \setminus |X : x_2|$, $|X : x_2'| = |X : x_2| \setminus |X : x_1|$, and $|X : x_3| = |X : x_1| \cap |X : x_2|$. Follow steps (a) and (b) below to determine new choice relations $|X : x_1'| \mapsto_{\mathbb{D}_1} |Y : y|$, $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y|$, $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y|$, and $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$ if they have not been determined: [6]

(a) For $i = 1$ and 2, do the following:

If $|X : x_i| \sqsubset_{\mathbb{D}_i} |Y : y|$ or $|X : x_i| \not\sqsubset_{\mathbb{D}_i} |Y : y|$, apply Proposition 4.2(a) or 4.2(c), respectively, to deduce the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i}$

[6] In Refinement Rules 1 to 12, new choice relations may be (automatically) deduced or (manually) defined.

$|Y : y|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y|$.[7] On the other hand, if $|X : x_i| \sqsubseteq_{\mathbb{D}_i} |Y : y|$, following Proposition 4.2(b), we need to define the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y|$.[8] Then, replace $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y|$ by the newly determined $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y|$.

(b) Apply Proposition 4.1 to deduce the relational operator for $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$ from $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y|$ (the last two choice relations are newly determined in step (a)). Then, replace $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y|$ by the newly deduced $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

Refinement Rule 2

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y|$ such that (i) $[X]$ and $[Y]$ are distinct categories and (ii) $|X : x_1| \subsetneq |X : x_2|$. Let $|X : x_2'| = |X : x_2| \setminus |X : x_1|$. Follow steps (a) and (b) below to determine new choice relations $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y|$, and $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$ if they have not been determined:

(a) If $|X : x_2| \sqsubset_{\mathbb{D}_2} |Y : y|$ or $|X : x_2| \not\sqsubset_{\mathbb{D}_2} |Y : y|$, apply Proposition 4.3(a) or 4.3(c), respectively, to deduce the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y|$. On the other hand, if $|X : x_2| \sqsubseteq_{\mathbb{D}_2} |Y : y|$, following Proposition 4.3(b), we need to define the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y|$. Then, replace $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y|$ by the newly determined $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y|$.

[7] Consider, for instance, $|X : x_i| \sqsubset_{\mathbb{D}_i} |Y : y|$. In this case, by Proposition 4.2(a), $|X : x_i'| \sqsubset_{\mathbb{D}_i} |Y : y|$ and $|X : x_3| \sqsubset_{\mathbb{D}_i} |Y : y|$ can be deduced.

[8] One may argue that Proposition 4.2 should also be used in step (a) to determine the new choice relations $|Y : y| \mapsto_{\mathbb{D}_i} |X : x_i'|$ and $|Y : y| \mapsto_{\mathbb{D}_i} |X : x_3|$. This is not necessary because a $\mathcal{T}$ or τ is arranged symmetrically to either side and, hence, if $|X : x| \mapsto |Y : y|$ (or $|X : x| \mapsto_{\mathbb{D}_i} |Y : y|$) has been determined in $\mathcal{T}$ (or τ_i), $|Y : y| \mapsto |X : x|$ (or $|Y : y| \mapsto_{\mathbb{D}_i} |X : x|$) would also have been determined in $\mathcal{T}$ (or τ_i) (see Tables 4.1 and 4.2 for example). Thus, for example, $|Y : y| \mapsto_{\mathbb{D}_1} |X : x_1'|$ and $|Y : y| \mapsto_{\mathbb{D}_1} |X : x_3|$ will still be determined when we apply Refinement Rule 3 to $|Y : y| \mapsto_{\mathbb{D}_1} |X : x_1|$ and $|Y : y| \mapsto_{\mathbb{D}_2} |X : x_2|$ (corresponding to Combination 3 in Table 4.5). Similar situations occur in Refinement Rules 2 to 12 to be described later.

(b) Apply Proposition 4.1 to deduce the relational operator for $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$ from $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y|$ (the last choice relation is newly determined in step (a)). Then, replace $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y|$ by the newly deduced $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y|$.

Refinement Rule 3

Suppose $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, (iii) $|Y : y_1| \not\subseteq |Y : y_2|$, and (iv) $|Y : y_2| \not\subseteq |Y : y_1|$. Let $|Y : y_1'| = |Y : y_1| \setminus |Y : y_2|$, $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$, and $|Y : y_3| = |Y : y_1| \cap |Y : y_2|$. Follow steps (a) and (b) below to determine new choice relations $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_1'|$, $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_3|$, $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_3|$, and $|X : x| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$ if they have not been determined:

(a) For $i = 1$ and 2, do the following:

> If $|X : x| \sqsubset_{\mathbb{D}_i} |Y : y_i|$ or $|X : x| \not\sqsubset_{\mathbb{D}_i} |Y : y_i|$, apply Proposition 4.4(a) or 4.4(c), respectively, to deduce the relational operators for $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_3|$. On the other hand, if $|X : x| \sqsubseteq_{\mathbb{D}_i} |Y : y_i|$, following Proposition 4.4(b), we need to define the relational operators for $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_3|$. During the definition process, according to Proposition 4.4(b), the relational operators for $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_3|$ must be "$\sqsubseteq$ " or "$\not\sqsubset$". Then, replace $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_i|$ by the newly determined $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_3|$.

(b) Apply Proposition 4.1 to deduce the relational operator for $|X : x| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$ from $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_3|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_3|$ (the last two choice relations are newly determined in step (a)). Then, replace $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_3|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_3|$ by the newly deduced $|X : x| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$.

Refinement Rule 4

Suppose $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories and (ii) $|Y : y_1| \subsetneq |Y : y_2|$. Let $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$. Follow steps (a) and (b) below to determine new choice relations $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_1|$, $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, and $|X : x| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$ if they have not been determined:

(a) If $|X : x| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.5(c) to deduce the relational operators for $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. On the other hand, if $|X : x| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.5(a) or 4.5(b), respectively, we need to define the relational operators for $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. If $|X : x| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, according to Proposition 4.5(b), during the definition process, we need to take into account that the relational operators for $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ must be "$\sqsubseteq$" or "$\not\sqsubset$". Then, replace $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_2'|$.

(b) Apply Proposition 4.1 to deduce the relational operator for $|X : x| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$ from $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_1|$ (the last choice relation is newly determined in step (a)). Then, replace $|X : x| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x| \mapsto_{\mathbb{D}_2} |Y : y_1|$ by the newly deduced $|X : x| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$.

Refinement Rule 5

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \cap |X : x_2| \neq \emptyset$, (iii) $|X : x_1| \not\subseteq |X : x_2|$, (iv) $|X : x_2| \not\subseteq |X : x_1|$, and (v) $|Y : y_1| \cap |Y : y_2| = \emptyset$. Let $|X : x_1'| = |X : x_1| \setminus |X : x_2|$, $|X : x_2'| = |X : x_2| \setminus |X : x_1|$, and $|X : x_3| = |X : x_1| \cap |X : x_2|$. Determine new choice relations $|X : x_1'| \mapsto_{\mathbb{D}_1} |Y : y_1|$, $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_1|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$, and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2|$ if they have not been determined as follows:

For $i = 1$ and 2, do the following:

If $|X : x_i| \sqsubset_{\mathbb{D}_i} |Y : y_i|$ or $|X : x_i| \not\sqsubset_{\mathbb{D}_i} |Y : y_i|$, apply Proposition 4.2(a) or 4.2(c), respectively, to deduce the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$. On the other hand, if $|X : x_i| \boxminus_{\mathbb{D}_i} |Y : y_i|$, following Proposition 4.2(b), we need to define the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$. Then, replace $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_i|$ by the newly determined $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$.

Refinement Rule 6

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \subsetneq |X : x_2|$, and (iii) $|Y : y_1| \cap |Y : y_2| = \emptyset$. Let $|X : x_2'| = |X : x_2| \setminus |X : x_1|$. Determine new choice relations $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$ if they have not been determined as follows:

If $|X : x_2| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_2| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.3(a) or 4.3(c), respectively, to deduce the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$. On the other hand, if $|X : x_2| \boxminus_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.3(b), we need to define the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$. Then, replace $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$.

Refinement Rule 7

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \cap |X : x_2| = \emptyset$, (iii) $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, (iv) $|Y : y_1| \not\subseteq |Y : y_2|$, and (v) $|Y : y_2| \not\subseteq |Y : y_1|$. Let $|Y : y_1'| = |Y : y_1| \setminus |Y : y_2|$, $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$, and $|Y : y_3| = |Y : y_1| \cap |Y : y_2|$. Determine new choice relations $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1'|$, $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$, $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_3|$ if they have not been determined as follows:

For $i = 1$ and 2, do the following:

> If $|X : x_i| \sqsubset_{\mathbb{D}_i} |Y : y_i|$ or $|X : x_i| \not\sqsubset_{\mathbb{D}_i} |Y : y_i|$, apply Proposition 4.4(a) or 4.4(c), respectively, to deduce the relational operators for $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_3|$. On the other hand, if $|X : x_i| \sqsubseteq_{\mathbb{D}_i} |Y : y_i|$, following Proposition 4.4(b), we need to define the relational operators for $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_3|$. During the definition process, according to Proposition 4.4(b), the relational operators for $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_3|$ must be "$\sqsubseteq$" or "$\not\sqsubset$". Then, replace $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_i|$ by the newly determined $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_3|$.

Refinement Rule 8

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \cap |X : x_2| = \emptyset$, and (iii) $|Y : y_1| \subsetneq |Y : y_2|$. Let $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$. Determine new choice relations $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ if they have not been determined as follows:

> If $|X : x_2| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.5(c) to deduce the relational operators for $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. On the other hand, if $|X : x_2| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_2| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.5(a) or 4.5(b), respectively, we need to define the relational operators for $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. If $|X : x_2| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, according to Proposition 4.5(b), during the definition process, we need to take into account that the relational operators for $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ must be "$\sqsubseteq$" or "$\not\sqsubset$". Then, replace $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2'|$.

Refinement Rule 9

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \cap |X : x_2| \neq \emptyset$, (iii) $|X : x_1| \not\subseteq |X : x_2|$, (iv) $|X : x_2| \not\subseteq |X : x_1|$, (v) $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, (vi) $|Y : y_1| \not\subseteq |Y : y_2|$, and (vii) $|Y : y_2| \not\subseteq |Y : y_1|$. Let $|X : x_1'|$ $= |X : x_1| \setminus |X : x_2|$, $|X : x_2'| = |X : x_2| \setminus |X : x_1|$, $|X : x_3| = |X : x_1| \cap |X : x_2|$, $|Y : y_1'| =$ $|Y : y_1| \setminus |Y : y_2|$, $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$, and $|Y : y_3| = |Y : y_1| \cap |Y : y_2|$. Follow steps (a) to (d) below to determine new choice relations $|X : x_1'| \mapsto_{\mathbb{D}_1} |Y : y_1|$, $|X : x_1'| \mapsto_{\mathbb{D}_1} |Y : y_1'|$, $|X : x_1'| \mapsto_{\mathbb{D}_1} |Y : y_3|$, $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_1|$, $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_1'|$, $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_3|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_3|$, $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_3|$, and $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2}$ $|Y : y_3|$ if they have not been determined:

(a) For $i = 1$ and 2, do the following:

> If $|X : x_i| \sqsubset_{\mathbb{D}_i} |Y : y_i|$ or $|X : x_i| \not\sqsubset_{\mathbb{D}_i} |Y : y_i|$, apply Proposition 4.2(a) or 4.2(c), respectively, to deduce the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i}$ $|Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$. On the other hand, if $|X : x_i| \sqsubseteq_{\mathbb{D}_i} |Y : y_i|$, following Proposition 4.2(b), we need to define the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$. Then, replace $|X : x_i|$ $\mapsto_{\mathbb{D}_i} |Y : y_i|$ by the newly determined $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i}$ $|Y : y_i|$.

(b) Consider $|X : x_1'| \sqsubset_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2'| \sqsubset_{\mathbb{D}_2} |Y : y_2|$, which are newly determined in step (a). For $i = 1$ and 2, do the following:

> If $|X : x_i'| \sqsubset_{\mathbb{D}_i} |Y : y_i|$ or $|X : x_i'| \not\sqsubset_{\mathbb{D}_i} |Y : y_i|$, apply Proposition 4.4(a) or 4.4(c), respectively, to deduce the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i}$ $|Y : y_i'|$ and $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_3|$. On the other hand, if $|X : x_i'| \sqsubseteq_{\mathbb{D}_i} |Y : y_i|$, following Proposition 4.4(b), we need to define the relational operators

for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_3|$. During the definition process, according to Proposition 4.4(b), the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_3|$ must be "$\sqsubseteq$" or "$\not\sqsubseteq$". Then, replace $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ by the newly determined $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_3|$.

(c) Now, consider $|X : x_3| \sqsubset_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_3| \sqsubset_{\mathbb{D}_2} |Y : y_2|$, which are newly determined in step (a). For $i = 1$ and 2, do the following:

If $|X : x_3| \sqsubset_{\mathbb{D}_i} |Y : y_i|$ or $|X : x_3| \not\sqsubset_{\mathbb{D}_i} |Y : y_i|$, apply Proposition 4.4(a) or 4.4(c), respectively, to deduce the relational operators for $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_3|$. On the other hand, if $|X : x_3| \sqsubseteq_{\mathbb{D}_i} |Y : y_i|$, following Proposition 4.4(b), we need to define the relational operators for $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_3|$. During the definition process, according to Proposition 4.4(b), the relational operators for $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_3|$ must be "$\sqsubseteq$" or "$\not\sqsubseteq$". Then, replace $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$ by the newly determined $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i'|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_3|$.

(d) Finally, consider $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_3|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_3|$, which are newly determined in step (c). Apply Proposition 4.1 to both choice relations to deduce the relational operator for $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$. Then, replace $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_3|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_3|$ by the newly deduced $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$.

Refinement Rule 10

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \cap |X : x_2| \neq \emptyset$, (iii) $|X : x_1| \not\subseteq |X : x_2|$, (iv) $|X : x_2| \not\subseteq |X : x_1|$, and (v) $|Y : y_1| \subsetneq |Y : y_2|$. Let $|X : x_1'| = |X : x_1| \setminus |X : x_2|$, $|X : x_2'| = |X : x_2| \setminus$

$|X : x_1|$, $|X : x_3| = |X : x_1| \cap |X : x_2|$, and $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$. Follow steps (a) to (d) below to determine new choice relations $|X : x_1'| \mapsto_{\mathbb{D}_1} |Y : y_1|$, $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_1|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_1|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_1|$, $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, and $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$ if they have not been determined:

(a) For $i = 1$ and 2, do the following:

If $|X : x_i| \sqsubset_{\mathbb{D}_i} |Y : y_i|$ or $|X : x_i| \not\sqsubset_{\mathbb{D}_i} |Y : y_i|$, apply Proposition 4.2(a) or 4.2(c), respectively, to deduce the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$. On the other hand, if $|X : x_i| \sqsubseteq_{\mathbb{D}_i} |Y : y_i|$, following Proposition 4.2(b), we need to define the relational operators for $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$. Then, replace $|X : x_i| \mapsto_{\mathbb{D}_i} |Y : y_i|$ by the newly determined $|X : x_i'| \mapsto_{\mathbb{D}_i} |Y : y_i|$ and $|X : x_3| \mapsto_{\mathbb{D}_i} |Y : y_i|$.

(b) Consider $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$, which is newly determined in step (a). If $|X : x_2'| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.5(c) to deduce the relational operators for $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. On the other hand, if $|X : x_2'| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_2'| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.5(a) or 4.5(b), respectively, we need to define the relational operators for $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. If $|X : x_2'| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, according to Proposition 4.5(b), during the definition process, we need to take into account that the relational operators for $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ must be "$\sqsubseteq$" or "$\not\sqsubset$". Then, replace $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$.

(c) Now, consider $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2|$, which is newly determined in step (a). If $|X : x_3| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.5(c) to deduce the relational operators for

$|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. On the other hand, if $|X : x_3| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_3| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.5(a) or 4.5(b), respectively, we need to define the relational operators for $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. If $|X : x_3| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, according to Proposition 4.5(b), during the definition process, we need to take into account that the relational operators for $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ must be "$\sqsubseteq$" or "$\not\sqsubset$". Then, replace $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_2'|$.

(d) Finally, consider $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_1|$, which are newly determined in steps (a) and (c), respectively. Apply Proposition 4.1 to both choice relations to deduce the relational operator for $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$. Then, replace $|X : x_3| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_3| \mapsto_{\mathbb{D}_2} |Y : y_1|$ by the newly deduced $|X : x_3| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$.

Refinement Rule 11

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \subsetneq |X : x_2|$, (iii) $|Y : y_1| \cap |Y : y_2| \neq \emptyset$, (iv) $|Y : y_1| \not\subseteq |Y : y_2|$, and (v) $|Y : y_2| \not\subseteq |Y : y_1|$. Let $|X : x_2'| = |X : x_2| \setminus |X : x_1|$, $|Y : y_1'| = |Y : y_1| \setminus |Y : y_2|$, $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$, and $|Y : y_3| = |Y : y_1| \cap |Y : y_2|$. Follow steps (a) to (e) below to determine new choice relations $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1'|$, $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$, $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_3|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_3|$, and $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$ if they have not been determined:

(a) If $|X : x_2| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_2| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.3(a) or 4.3(c), respectively, to deduce the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$. On the other hand, if $|X : x_2| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.3(b),

we need to define the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$. Then, replace $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$.

(b) If $|X : x_1| \sqsubset_{\mathbb{D}_1} |Y : y_1|$ or $|X : x_1| \not\sqsubset_{\mathbb{D}_i} |Y : y_1|$, apply Proposition 4.4(a) or 4.4(c), respectively, to deduce the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1'|$ and $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$. On the other hand, if $|X : x_1| \sqsubseteq_{\mathbb{D}_1} |Y : y_1|$, following Proposition 4.4(b), we need to define the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1'|$ and $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$. During the definition process, according to Proposition 4.4(b), the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1'|$ and $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$ must be "$\sqsubseteq$" or "$\not\sqsubset$". Then, replace $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ by the newly determined $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1'|$ and $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$.

(c) Consider $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$, which is newly determined in step (a). If $|X : x_1| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_1| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.4(a) or 4.4(c), respectively, to deduce the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_3|$. On the other hand, if $|X : x_1| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.4(b), we need to define the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_3|$. During the definition process, according to Proposition 4.4(b), the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_3|$ must be "$\sqsubseteq$" or "$\not\sqsubset$". Then, replace $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_3|$.

(d) Now, consider $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$, which is newly determined in step (a). If $|X : x_2'| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_2'| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.4(a) or 4.4(c), respectively, to deduce the relational operators for $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_3|$. On the other hand, if $|X : x_2'| \sqsubseteq_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.4(b), we need to define the relational operators for $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_3|$. During the definition process, according to Proposition 4.4(b),

the relational operators for $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_3|$ must be "$\boxplus$" or "$\not\sqsubset$". Then, replace $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_3|$.

(e) Finally, consider $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_3|$, which are newly determined in steps (b) and (c), respectively. Apply Proposition 4.1 to both choice relations to deduce the relational operator for $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$. Replace $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_3|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_3|$ by $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_3|$.

Refinement Rule 12

Suppose $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ such that (i) $[X]$ and $[Y]$ are distinct categories, (ii) $|X : x_1| \subsetneq |X : x_2|$, and (iii) $|Y : y_1| \subsetneq |Y : y_2|$. Let $|X : x_2'| = |X : x_2| \setminus |X : x_1|$ and $|Y : y_2'| = |Y : y_2| \setminus |Y : y_1|$. Follow steps (a) to (d) below to determine new choice relations $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_1|$, $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_1|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$, $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2'|$, and $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$ if they have not been determined:

(a) If $|X : x_2| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_2| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.3(a) or 4.3(c), respectively, to deduce the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$. On the other hand, if $|X : x_2| \boxplus_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.3(b), we need to define the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$. Then, replace $|X : x_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ and $|X : x_2'| \mapsto_{\mathbb{D}_2} |Y : y_2|$.

(b) Consider $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$, which is newly determined in step (a). If $|X : x_1| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.5(c) to deduce the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2'|$. On the other hand, if $|X : x_1| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x_1| \boxplus_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.5(a) or 4.5(b), respectively, we need to define the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and

$|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y'_2|$. If $|X : x_1| \boxplus_{\mathbb{D}_2} |Y : y_2|$, according to Proposition 4.5(b), during the definition process, we need to take into account that the relational operators for $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y'_2|$ must be "$\boxplus$" or "$\not\sqsubset$". Then, replace $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y'_2|$.

(c) Now, consider $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$, which is newly determined in step (a). If $|X : x'_2| \not\sqsubset_{\mathbb{D}_2} |Y : y_2|$, apply Proposition 4.5(c) to deduce the relational operators for $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y'_2|$. On the other hand, if $|X : x'_2| \sqsubset_{\mathbb{D}_2} |Y : y_2|$ or $|X : x'_2| \boxplus_{\mathbb{D}_2} |Y : y_2|$, following Proposition 4.5(a) or 4.5(b), respectively, we need to define the relational operators for $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y'_2|$. If $|X : x'_2| \boxplus_{\mathbb{D}_2} |Y : y_2|$, according to Proposition 4.5(b), during the definition process, we need to take into account that the relational operators for $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y'_2|$ must be "$\boxplus$" or "$\not\sqsubset$". Then, replace $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y_2|$ by the newly determined $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y_1|$ and $|X : x'_2| \mapsto_{\mathbb{D}_2} |Y : y'_2|$.

(d) Apply Proposition 4.1 to deduce the relational operator for $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$ from $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_1|$ (the last choice relation is newly determined in step (b)). Then, replace $|X : x_1| \mapsto_{\mathbb{D}_1} |Y : y_1|$ and $|X : x_1| \mapsto_{\mathbb{D}_2} |Y : y_1|$ by $|X : x_1| \mapsto_{\mathbb{D}_1 \cup \mathbb{D}_2} |Y : y_1|$.

The above refinement rules show that new choice relations (involving more "fine grain" choices) can be automatically deduced as far as possible by applying Propositions 4.2 to 4.5. Even in some situations where automatic deductions are not possible, a certain degree of consistency checking of manually defined choice relations can be provided. For example, in step (a) of Refinement Rule 3 involving the application of Proposition 4.4(b) when $|X : x| \boxplus_{\mathbb{D}_i} |Y : y_i|$, we know that the relational operators for

$|X : x| \mapsto_{\mathbb{D}_i} |Y : y'_i|$ and $|X : x| \mapsto_{\mathbb{D}_i} |Y : y_3|$ must be "$\sqsubseteq$" or "$\not\sqsubseteq$" (but not "$\sqsubset$"). The features of automatic deduction and consistency checking greatly contribute to the effectiveness of determining choice relations. [9]

So far, we have illustrated how Propositions 4.2 to 4.5 can be recursively applied to refine a pair of choice relations with overlapping choices. We now extend our refinement mechanism to handle more than two choice relations as follows:

An Algorithm (refine_table) for Refining Choice Relations with Overlapping Choices

We follow the notation used in the algorithm integrate_table. Given a linked list L with n elements $m_1, m_2, \ldots, m_n$, where $n \geq 1$, and given n associated, nonempty linked lists $LL_{m_1}, LL_{m_2}, \ldots, LL_{m_n}$, perform the following steps to refine the overlapping choices and their choice relations stored in every such associated linked list:

(1) Initialization of Lists of Choice Relations

(a) Initialize P, Q, and R as empty linked lists.

/* Each element of a nonempty P, Q, and R stores a choice relation during the refinement process. Thus, P, Q, and R have the same structure as any LL_{m_i}'s (where $1 \leq i \leq n$). Furthermore, after step (2) of this algorithm, all the remaining choice relations do not involve any overlapping choices, and these relations are stored in P. */

(b) Set $P = LL_{m_1}$.

[9] Readers should not confuse our automatic deduction and consistency check techniques described above with the similar techniques developed for CHOC'LATE by Chen et al. [21, 62]. While our automatic deduction and consistency check techniques are specifically developed for overlapping choices, the techniques by Chen et al. [21, 62] apply to nonoverlapping choices only.

(2) Refinement of Choice Relations with Overlapping Choices

For $i = 2$ to n, do the following:

(a) Append a copy of the contents of LL_{m_i} to P.

/* Note that, for every choice relation in LL_{m_i}, the header and trailer choices belong to different categories. */

/* Step (2)(b) below deals with the refinement of choice relations involving overlapping header choices. */

(b) Repeat this step if there exist any overlapping ***header*** choices in P:

(i) Select a header choice $|X : x_a|$ from P so that there exists another header choice $|X : x_b|$ in P satisfying the following properties:

- $|X : x_a|$ overlaps with $|X : x_b|$, and
- $|X : x_a| \not\subseteq |X : x_b|$.

(ii) Move all the ***choice relations*** with $|X : x_a|$ as their header choices from P to Q.

/* All the choice relations in Q will be processed in step (2)(b)(iii) below using Propositions 4.2 and 4.3 (if applicable) and the results will be temporarily stored in R. */

(iii) For every unprocessed choice relation $|X : x_a| \mapsto_{\mathbb{D}_j} |Y : y_c|$ in Q, do the following:

With respect to $|X : x_a|$ and $|X : x_b|$ in step (2)(b)(i) above, if Proposition 4.2 applies, then use this proposition to refine the choice relation chosen, with a view to determining new choice relations (with respect to $\mathbb{D}_j$) if they do not exist in R. Only those new choice relations with header choices $|X : x'|$ and $|X : x''|$

(where $|X : x'| = |X : x_a| \setminus |X : x_b|$ and $|X : x''| = |X : x_a| \cap |X : x_b|$) need to be determined (see also footnote 8 for explanations). [10]

Alternatively, with respect to $|X : x_a|$ and $|X : x_b|$ in step (2)(b)(i) above, if Proposition 4.3 applies, then use this proposition to refine the choice relation chosen, with a view to determining new choice relations (with respect to $\mathbb{D}_j$) if they do not exist in R. Only those new choice relations with header choices $|X : x_b|$ and $|X : x'|$ (where $|X : x'| = |X : x_a| \setminus |X : x_b|$) need to be determined (see also footnote 8 for explanations). [11]

During the determination of new choice relations, whenever possible, perform automatic deductions of new choice relations according to Propositions 4.2 and 4.3. Perform consistency checks for all new, manually defined choice relations using the two propositions. If inconsistency is detected, alert the users about the problem and prompt them to redo this step (2)(b)(iii) immediately. Store all newly determined choice relations in R.

(iv) Append all the choice relations stored in R to P.

/* The choice relations in R may still contain other overlapping header choices. Thus, such choice relations are appended to P for the next round of refinement if applicable. */

(v) Delete the contents of Q and R.

(vi) For every pair of choice relations $|X : x| \mapsto_{\mathbb{D}_u} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}_v} |Y : y|$

[10] $|X : x_a|$, $|X : x_b|$, $|X : x'|$, and $|X : x''|$ here correspond to $|X : x_1|$, $|X : x_2|$, $|X : x'_1|$, and $|X : x_3|$ in Proposition 4.2, respectively.

[11] $|X : x_a|$, $|X : x_b|$, and $|X : x'|$ here correspond to $|X : x_2|$, $|X : x_1|$, and $|X : x'_2|$ in Proposition 4.3, respectively.

in P:

/* The above scenario is not possible at the start of step (2)(b) because of the algorithm integrate_table which has been executed. This scenario, however, may become possible after executing steps (2)(b)(ii)–(iv) of this algorithm refine_table. */

Apply Proposition 4.1 to deduce $|X : x| \mapsto_{\mathbb{D}_u \cup \mathbb{D}_v} |Y : y|$ from $|X : x| \mapsto_{\mathbb{D}_u} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}_v} |Y : y|$. Replace $|X : x| \mapsto_{\mathbb{D}_u} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}_v} |Y : y|$ by $|X : x| \mapsto_{\mathbb{D}_u \cup \mathbb{D}_v} |Y : y|$ in P.

/* Step (2)(c) below deals with the refinement of choice relations involving overlapping trailer choices. */

(c) Repeat this step if there exist any overlapping ***trailer*** choices in P:

(i) Select a trailer choice $|Y : y_a|$ from P so that there exists another trailer choice $|Y : y_b|$ in P satisfying the following properties:

- $|Y : y_a|$ overlaps with $|Y : y_b|$, and
- $|Y : y_a| \not\sqsubseteq |Y : y_b|$.

(ii) Move all the ***choice relations*** with $|Y : y_a|$ as their trailer choices from P to Q.

/* All the choice relations in Q will be processed in step (2)(c)(iii) below using Propositions 4.4 and 4.5 (if applicable) and the results will be temporarily stored in R. */

(iii) For every unprocessed choice relation $|X : x_c| \mapsto_{\mathbb{D}_j} |Y : y_a|$ in Q, do the following:

With respect to $|Y : y_a|$ and $|Y : y_b|$ in step (2)(c)(i) above, if Proposition 4.4 applies, then use this proposition to refine the

choice relation chosen, with a view to determining new choice relations (with respect to $\mathbb{D}_j$) if they do not exist in R. Only those new choice relations with trailer choices $|Y : y'|$ and $|Y : y''|$ (where $|Y : y'| = |Y : y_a| \setminus |Y : y_b|$ and $|Y : y''| = |Y : y_a| \cap |Y : y_b|$) need to be determined (see also footnote 8 for explanations). [12]

Alternatively, with respect to $|Y : y_a|$ and $|Y : y_b|$ in step (2)(c)(i) above, if Proposition 4.5 applies, then use this proposition to refine the choice relation chosen, with a view to determining new choice relations (with respect to $\mathbb{D}_j$) if they do not exist in R. Only those new choice relations with trailer choices $|Y : y_b|$ and $|Y : y'|$ (where $|Y : y'| = |Y : y_a| \setminus |Y : y_b|$) need to be determined (see also footnote 8 for explanations). [13]

During the determination of new choice relations, whenever possible, perform automatic deductions of new choice relations according to Propositions 4.4 and 4.5. Perform consistency checks for all new, manually defined choice relations using the two propositions. If inconsistency is detected, alert the users about the problem and prompt them to redo this step (2)(c)(iii) immediately. Store all newly determined choice relations in R.

(iv) Append all the choice relations stored in R to P.

/* The choice relations in R may still contain other overlapping trailer choices. Thus, such choice relations are appended to P for the next round

[12] $|Y : y_a|$, $|Y : y_b|$, $|Y : y'|$, and $|Y : y''|$ here correspond to $|Y : y_1|$, $|Y : y_2|$, $|Y : y'_1|$, and $|Y : y_3|$ in Proposition 4.4, respectively.

[13] $|Y : y_a|$, $|Y : y_b|$, and $|Y : y'|$ here correspond to $|Y : y_2|$, $|Y : y_1|$, and $|Y : y'_2|$ in Proposition 4.5, respectively.

of refinement if applicable. */

(v) Delete the contents of Q and R.

(vi) For every pair of choice relations $|X : x| \mapsto_{\mathbb{D}_u} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}_v} |Y : y|$ in P:

/* The above scenario is not possible at the start of step (2)(c) because of the algorithm integrate_table which has been executed. This scenario, however, may become possible after executing steps (2)(c)(ii)–(iv) of this algorithm refine_table. */

Apply Proposition 4.1 to deduce $|X : x| \mapsto_{\mathbb{D}_u \cup \mathbb{D}_v} |Y : y|$ from $|X : x| \mapsto_{\mathbb{D}_u} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}_v} |Y : y|$. Replace $|X : x| \mapsto_{\mathbb{D}_u} |Y : y|$ and $|X : x| \mapsto_{\mathbb{D}_v} |Y : y|$ by $|X : x| \mapsto_{\mathbb{D}_u \cup \mathbb{D}_v} |Y : y|$ in P.

(3) Construction of Choice Relation Table

(a) Initialize $\mathcal{T}$ as an empty table.

(b) For every choice relation $|X : x| \mapsto_{\mathbb{D}} |Y : y|$ in P:

(i) If there does not exist any row and column in $\mathcal{T}$ that corresponds to $|X : x|$ in $|X : x| \mapsto_{\mathbb{D}} |Y : y|$, then add such a row and column to $\mathcal{T}$.

(ii) If there does not exist any row and column in $\mathcal{T}$ that corresponds to $|Y : y|$ in $|X : x| \mapsto_{\mathbb{D}} |Y : y|$, then add such a row and column to $\mathcal{T}$.

(iii) Store the relational operator of $|X : x| \mapsto_{\mathbb{D}} |Y : y|$ in the corresponding element of $\mathcal{T}$.

(c) Apply the choice relation table construction algorithm provided by CHOC'LATE (see [21, 62] for details) for determining all the yet-to-be-defined relational operators in $\mathcal{T}$.

At the completion of refine_table, a $\mathcal{T}$ is constructed in which all its choice relations (which do not involve overlapping choices) have been determined. Thereafter, $\mathcal{T}$ will be processed by CHOC'LATE for test case generation [21, 62].

The basic rationale of step (2) of refine_table has been highlighted by means of the 12 refinement rules discussed earlier. Example 4.4 below gives further explanations:

Example 4.4 (Refining Choice Relations with Overlapping Choices)
Refer to Example 4.2. Table 4.6 depicts the interim choice relation table $\mathcal{T}_1$ constructed after executing step (3)(b) of refine_table. After a close examination of Table 4.6, we have the following observations:

(a) All the choice relations in Table 4.6 do not involve overlapping choices (before executing refine_table, $|Y : y_2|$ overlaps with $|Y : y_3|$ and $|Y : y_4|$).

(b) When step (3)(c) commences, CHOC'LATE will first automatically assign the relational operators "$\sqsubset$" and "$\not\sqsubset$" to every choice relation $|M : m| \mapsto_{\mathbb{D}} |M : m|$ and $|M : m_1| \mapsto_{\mathbb{D}} |M : m_2|$ (where $[M]$ is any category, $|M : m|$, $|M : m_1|$, and $|M : m_2|$ are any choices, and $|M : m_1| \neq |M : m_2|$), respectively, in $\mathcal{T}_1$. Eight yet-to-be-defined choice relations, whose header and trailer choices belong to *different* categories, will remain after this process. All of them involve ($|W : w_1|$ or $|W : w_2|$) and ($|Z : z_1|$ or $|Z : z_2|$) as their header or trailer choices. These choice relations occur owing to the fact that category $[Z]$ and its associated choices are identified from $\mathbb{C}_1$ only (but not $\mathbb{C}_2$), whereas category $[W]$ and its associated choices are identified from $\mathbb{C}_2$ only (but not $\mathbb{C}_1$). Thus, τ_1 and τ_2 (see Tables 4.1 and 4.2) do not contain any choice relations $|W : w_i| \mapsto_{\{\mathbb{C}_k\}} |Z : z_j|$ and $|Z : z_j| \mapsto_{\{\mathbb{C}_k\}} |W : w_i|$ (where $i, j, k = 1$ or 2) before executing refine_table.

All these eight yet-to-be-defined relations will be determined in step (3)(c) of refine_table, by applying the table construction algorithm provided by CHOC'LATE.

When defining choice relations, the testers may need additional information from sources other than $\mathbb{S}$ such as end users and software designers.

(c) In Table 4.6, all the choice relations $|X : x_i| \mapsto_{\{\mathbb{C}_1,\mathbb{C}_2\}} |Y : y_j|$ and $|Y : y_j| \mapsto_{\{\mathbb{C}_1,\mathbb{C}_2\}} |X : x_i|$ are automatically deduced in step (2)(b)(vi) or (2)(c)(vi) of refine_table. These choice relations occur because categories $[X]$ and $[Y]$ and their associated choices are identified from both $\mathbb{C}_1$ and $\mathbb{C}_2$. As a result, both τ_1 and τ_2 contain choice relations $|X : x_m| \mapsto_{\{\mathbb{C}_k\}} |Y : y_n|$ and $|Y : y_n| \mapsto_{\{\mathbb{C}_k\}} |X : x_m|$ ($k = 1$ in τ_1 and $k = 2$ in τ_2); some of these $|Y : y_n|$'s may be overlapping choices (more specifically, $|Y : y_2| = |Y : y_3| \cup |Y : y_4|$). During the execution of refine_table, $|X : x_m| \mapsto_{\{\mathbb{C}_k\}} |Y : y_n|$ and $|Y : y_n| \mapsto_{\{\mathbb{C}_k\}} |X : x_m|$ (or the choice relations refined from $|X : x_m| \mapsto_{\{\mathbb{C}_k\}} |Y : y_n|$ and $|Y : y_n| \mapsto_{\{\mathbb{C}_k\}} |X : x_m|$ in step (2)(b)(iii) or 2(c)(iii) of the algorithm) are converted by step (2)(b)(vi) or 2(c)(vi) into $|X : x_i| \mapsto_{\{\mathbb{C}_1,\mathbb{C}_2\}} |Y : y_j|$ or $|Y : y_j| \mapsto_{\{\mathbb{C}_1,\mathbb{C}_2\}} |X : x_i|$, in which no overlapping choices are involved.

(d) In Table 4.6, all the choice relations with $|Z : z_1|$ or $|Z : z_2|$ as their header or trailer choices (but not both) are determined with respect to $\{\mathbb{C}_1\}$ only. This is because $|Z : z_1|$ and $|Z : z_2|$ are not identified from $\mathbb{C}_2$ initially and, hence, both choices do not appear as header or trailer choices in all the choice relations in τ_2 (see Table 4.2) before the execution of refine_table. Consequently, steps (2)(b)(vi) and (2)(c)(vi) of the algorithm will not deduce any choice relation with respect to $\{\mathbb{C}_1, \mathbb{C}_2\}$ and has $|Z : z_1|$ or $|Z : z_2|$ as its header or trailer choice.

Similarly, since $|W : w_1|$ and $|W : w_2|$ are not identified from $\mathbb{C}_1$ initially and, hence, both choices do not appear as header or trailer choices in all the choice relations in τ_1 (see Table 4.1) before the execution of refine_table. Consequently, steps (2)(b)(vi) and (2)(c)(vi) of the algorithm will not deduce any choice relation with respect to $\{\mathbb{C}_1, \mathbb{C}_2\}$ and has $|W : w_1|$ or $|W : w_2|$ as its header or trailer choice. This explains why, in Table 4.6, all the choice relations with $|W : w_1|$ or $|W : w_2|$

	$\lvert W : w_1\rvert$	$\lvert W : w_2\rvert$	$\lvert X : x_1\rvert$	$\lvert X : x_2\rvert$	$\lvert Y : y_1\rvert$	$\lvert Y : y_3\rvert$	$\lvert Y : y_4\rvert$	$\lvert Z : z_1\rvert$	$\lvert Z : z_2\rvert$
$\lvert W : w_1\rvert$			$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\mathbb{E}_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$		
$\lvert W : w_2\rvert$			$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\sqsubset_{\{\mathbb{C}_2\}}$		
$\lvert X : x_1\rvert$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$			$\mathbb{E}_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\lvert X : x_2\rvert$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$			$\mathbb{E}_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\mathbb{E}_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\lvert Y : y_1\rvert$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\mathbb{E}_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\mathbb{E}_{\{\mathbb{C}_1,\mathbb{C}_2\}}$				$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\lvert Y : y_3\rvert$	$\mathbb{E}_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\mathbb{E}_{\{\mathbb{C}_1,\mathbb{C}_2\}}$				$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\lvert Y : y_4\rvert$	$\not\sqsubset_{\{\mathbb{C}_2\}}$	$\mathbb{E}_{\{\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1,\mathbb{C}_2\}}$	$\not\sqsubset_{\{\mathbb{C}_1,\mathbb{C}_2\}}$				$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$
$\lvert Z : z_1\rvert$			$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$		
$\lvert Z : z_2\rvert$			$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$	$\not\sqsubset_{\{\mathbb{C}_1\}}$		

Table 4.6: Interim Choice Relation Table $\mathcal{T}_1$ Constructed after Step (3)(b) of **refine_table**

as their header or trailer choices (but not both) are determined with respect to $\{\mathbb{C}_2\}$ only.

(e) All the choice relations in Table 4.6 can be considered as determined with respect to the entire $\mathbb{S}$. This is explained as follows. On completion of step (2) of refine_table, all the choice relations determined with respect to $\{\mathbb{C}_1, \mathbb{C}_2\}$ can be considered as those that are determined with respect to $\mathbb{S}$, because $\mathbb{C}_1$ and $\mathbb{C}_2$ together constitute $\mathbb{S}$. On the other hand, for all the choice relations determined with respect to $\{\mathbb{C}_1\}$ or $\{\mathbb{C}_2\}$ only, they are still effectively the same as those relations determined with respect to $\mathbb{S}$. Consider, for instance, $|W : w_1| \not\sqsubset_{\{\mathbb{C}_2\}} |X : x_1|$ in Table 4.6. As mentioned in observation (d) above, $\mathbb{C}_1$ does not contain any information leading to the identification of category $[W]$ and its associated choices. Thus, $|W : w_1| \not\sqsubset |X : x_1|$ would have been determined if the tester takes into account the entire $\mathbb{S}$ when determining $|W : w_1| \mapsto |X : x_1|$. ■

4.4 Case Study 1

4.4.1 Setting of Study

We have evaluated the effectiveness of DESSERT based on the specification (denoted by $\mathbb{S}_{\mathsf{VAS}}$) of a real-life commercial software system, which is known as visitor administration system and denoted by VAS. The system is now in use in an international airline company, which prefers to remain anonymous and, hence, will only be referred to as AIR in this thesis. The main purposes of VAS are to provide systematic and efficient registration, authorization, access control, and reporting of visitor activities in the AIR headquarter.

VAS has two subsystems: visitor request administration (VRA) and visitor card management (VCM). The first subsystem VRA is a Web enabled application that runs on an AIR's Intranet Web server. Every staff member authorized for making visitor requests shall have access to VRA. The second subsystem VCM is installed on every reception counter's desktop computer that is Windows-based and comes with a contactless SmartCard reader terminal. In general, VCM has similar functionality as VRA, thus allowing a reception operator to issue access cards to non-anticipated visitors via his/her desktop computer in which VCM is installed.

To register a visitor for an anticipated visit to AIR, the staff member concerned shall make a request in VRA. Information such as the staff member's and visitor's particulars, and the visitation details are to be entered as part of the request. If the visit is: (a) not on a weekend or a public holiday, (b) within office hours, (c) spans a day or less, and (d) involves an access area within the default zone, the request will go into the receptionist log. Otherwise, the request is considered as exceptional and it will go into the endorsement log, awaiting AIR Security's approval. A request will also go into the endorsement log if the visitor is blacklisted in VAS. If this happens, AIR Security can waive or reject

the visitor request. If AIR Security approves an exceptional request or waives a request involving a blacklisted visitor, this request will then be moved from the endorsement log to the receptionist log. Later, when the visitor arrives at any reception counter, the reception operator will search through the receptionist log for a corresponding visitor request record. If found, that record will be edited by the operator before issuing a visitor access card.

The specification $\mathbb{S}_{\mathsf{VAS}}$ contains various $\mathbb{C}$'s such as narrative descriptions of the system, state machines, activity diagrams, data flow diagrams, sample input screens and outputs, and a data dictionary. Because of this characteristic, $\mathbb{S}_{\mathsf{VAS}}$ lends itself to be a very good specification document for our case studies. In order to protect the identity of AIR, we have slightly amended the original specification before our studies commence. The majority of the content of $\mathbb{S}_{\mathsf{VAS}}$, however, has remained intact.

VAS provides many functions such as "Process Visitor Requests", "Endorse Visitor Requests", "Issue Visitor Access Cards", "Process Overdue Access Cards", and "Update Visitor Blacklist". Our studies mainly focus on the function "Process Visitor Requests", which is a core feature of VAS. We have recruited a volunteer for our studies, who is known as Participant *X* in this thesis. This participant has a postgraduate degree in IT and several years of commercial experience in software development. The following subsections describe our studies and the results in detail.

4.4.2 Constructing a Preliminary Choice Relation Table from an Activity Diagram

We found one activity diagram (denoted by $\mathcal{A}_{\mathsf{REQUEST}}$) in $\mathbb{S}_{\mathsf{VAS}}$ which is related to the processing of visitor requests. We have given Participant *X* a copy of $\mathcal{A}_{\mathsf{REQUEST}}$ and an one-page executive summary of $\mathbb{S}_{\mathsf{VAS}}$ (instead of the entire $\mathbb{S}_{\mathsf{VAS}}$), and have asked him to construct a preliminary choice relation table (denoted by $\tau(\mathcal{A}_{\mathsf{REQUEST}})$) from

	$\mathcal{A}_{\mathsf{REQUEST}}$	$\mathcal{D}_{\mathsf{REQUEST}}$	$\mathcal{S}_{\mathsf{REQUEST}}$	$\mathcal{N}_{\mathsf{REQUEST}}$
Number of Potential Categories	5	3	3	11
Number of Potential Choices	10	6	6	24
Number (Percentage) of Missing Categories	0 (0%)	0 (0%)	0 (0%)	1 (9%)
Number (Percentage) of Problematic Categories	0 (0%)	3 (100%)	2 (67%) *	3 (27%)
Number (Percentage) of Missing Choices	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Number (Percentage) of Problematic Choices	0 (0%)	3 (50%)	0 (0%)	3 (13%)

(*) These two problematic categories are irrelevant.

Table 4.7: Number of Potential/Missing/Problematic Categories and Choices for $\mathbb{S}_{\mathsf{VAS}}$ (Before Correction)

$\mathcal{A}_{\mathsf{REQUEST}}$, using the algorithm construct_table in Section 4.2.2. This arrangement ensures that $\tau(\mathcal{A}_{\mathsf{REQUEST}})$ is constructed without using the information in other $\mathbb{C}$'s. We observe that it is difficult to construct $\tau(\mathcal{A}_{\mathsf{REQUEST}})$ without some basic background information of VAS. This explains why an executive summary of $\mathbb{S}_{\mathsf{VAS}}$ is given to Participant X. As a precaution, we have told Participant X to construct $\tau(\mathcal{A}_{\mathsf{REQUEST}})$ solely based on $\mathcal{A}_{\mathsf{REQUEST}}$. Our subsequent checking of $\tau(\mathcal{A}_{\mathsf{REQUEST}})$ has confirmed that this is the case.

$\mathcal{A}_{\mathsf{REQUEST}}$ contains five decision points; each of them is associated with a unique factor of VAS and two guard conditions. Furthermore, each of these 10 ($= 5 \times 2$) guard conditions is a subcondition by itself. An example of the decision points is "Type of Visitor", with "Normal" and "Blacklisted" as its two associated guard conditions (or subconditions). According to the algorithm construct_table, these two subconditions will result in the definition of the potential category [Type of Visitor] and its two associated potential choices |Type of Visitor: Normal| and |Type of Visitor: Blacklisted|.

A close examination of $\tau(\mathcal{A}_{\mathsf{REQUEST}})$ reveals that Participant X has defined five potential categories; each of them is associated with two potential choices. With respect to $\mathcal{A}_{\mathsf{REQUEST}}$, no missing/problematic categories and choices are detected in $\tau(\mathcal{A}_{\mathsf{REQUEST}})$

(see the first and second leftmost columns of Table 4.7). In other words, all the potential categories are relevant and all the potential choices are valid. Furthermore, all the 100 choice relations in $\tau(\mathcal{A}_{\mathsf{REQUEST}})$ are correctly determined by construct_table (with respect to $\mathcal{A}_{\mathsf{REQUEST}}$), among which 10, 10, and 80 come from steps (5), (6), and (7) of the algorithm, respectively. Thus, yet-to-be-defined choice relations do not exist at the completion of construct_table.

4.4.3 Constructing Preliminary Choice Relation Tables from Other Specification Components

In relation to visitor request processing, we found one data flow diagram (denoted by $\mathcal{D}_{\mathsf{REQUEST}}$), one state machine (denoted by $\mathcal{S}_{\mathsf{REQUEST}}$), and one section of narrative descriptions (denoted by $\mathcal{N}_{\mathsf{REQUEST}}$) in $\mathbb{S}_{\mathsf{VAS}}$. We have repeated the study of $\mathcal{A}_{\mathsf{REQUEST}}$ for each of these three $\mathbb{C}$'s but with two exceptions. Firstly, defining potential categories and choices from $\mathcal{D}_{\mathsf{REQUEST}}$, $\mathcal{S}_{\mathsf{REQUEST}}$, and $\mathcal{N}_{\mathsf{REQUEST}}$ by Participant X is done in an ad hoc manner without the support of any systematic methodology. Secondly, immediately after such definition, we would first check the potential categories and choices to see if there are any mistakes, and ensure that none of them is defined using the information in other $\mathbb{C}$'s and the executive summary of $\mathbb{S}_{\mathsf{VAS}}$. After correcting the mistakes (if any), Participant X would then proceed to construct the corresponding τ's, which are denoted by $\tau(\mathcal{D}_{\mathsf{REQUEST}})$, $\tau(\mathcal{S}_{\mathsf{REQUEST}})$, and $\tau(\mathcal{N}_{\mathsf{REQUEST}})$, respectively.

After checking the potential categories and choices defined from $\mathcal{D}_{\mathsf{REQUEST}}$, $\mathcal{S}_{\mathsf{REQUEST}}$, and $\mathcal{N}_{\mathsf{REQUEST}}$ by Participant X, we observe a missing category, and some problematic categories and choices, with respect to these $\mathbb{C}$'s (see the three rightmost columns of Table 4.7). This observation is consistent with the results reported in Chapter 3 that mistakes are likely to occur when defining categories and choices in an ad hoc manner. (We shall omit the detailed analysis of these mistakes because it is not the main focus of

	$\mathcal{A}_{\mathsf{REQUEST}}$	$\mathcal{D}_{\mathsf{REQUEST}}$	$\mathcal{S}_{\mathsf{REQUEST}}$	$\mathcal{N}_{\mathsf{REQUEST}}$
Number of Relevant Categories	5	3	1	12
Number of Valid Choices	10	9	2	30

Table 4.8: Number of Relevant Categories and Valid Choices for $\mathbb{S}_{\mathsf{VAS}}$ (After Correction)

the chapter.) We have then discussed with Participant *X* about the mistakes and helped him make the necessary corrections. Table 4.8 shows the number of relevant categories and valid choices for the three $\mathbb{C}$'s after the correction. Note that some of these categories and choices defined from different $\mathbb{C}$'s are identical. After counting, we found 12 relevant categories and 32 valid choices which are all distinct.

When we check the relevant categories and valid choices again, we observe that three ($= \frac{3}{12} \times 100\% = 25\%$) relevant categories defined from one $\mathbb{C}$ cannot be defined from any of the other three $\mathbb{C}$'s. An example is the relevant category [Number of Visitors (*N*)], with its associated valid choices |Number of Visitors (*N*): $N = 0$|, |Number of Visitors (*N*): $0 < N <$ Maximum Limit|, and |Number of Visitors (*N*): $N =$ Maximum Limit|, which can only be defined from $\mathcal{N}_{\mathsf{REQUEST}}$. We further observe that none of the 12 relevant categories can be defined from all the four $\mathbb{C}$'s. These observations thus support our argument put forward in Section 4.3 that an individual $\mathbb{C}$ may carry incomplete information of the software system.

At this stage we have encountered a problem. After the correction, only one relevant category has been defined from $\mathcal{S}_{\mathsf{REQUEST}}$ (see Table 4.8). [14] This phenomenon contradicts an assumption of the algorithm integrate_table (see Section 4.3.1), that every τ must involve two or more relevant categories. To solve this problem, we have asked Participant *X* to reconsider $\mathcal{D}_{\mathsf{REQUEST}}$ and $\mathcal{S}_{\mathsf{REQUEST}}$ together as one single $\mathbb{C}$ (denoted by

[14] The situation of having only one relevant category defined from a $\mathbb{C}$ is fairly uncommon.

$\mathcal{M}_{\mathsf{REQUEST}}$), from which the corresponding τ (denoted by $\tau(\mathcal{M}_{\mathsf{REQUEST}})$) is constructed. With this approach, four relevant categories and 11 valid choices are defined from $\mathcal{M}_{\mathsf{REQUEST}}$.

Afterward, Participant *X* has constructed $\tau(\mathcal{M}_{\mathsf{REQUEST}})$ and $\tau(\mathcal{N}_{\mathsf{REQUEST}})$ by defining the relation between every pair of valid choices. During the process, he has defined a few incorrect choice relations and subsequent corrections have been made.

4.4.4 Consolidating Preliminary Choice Relation Tables

Here we discuss how $\tau(\mathcal{A}_{\mathsf{REQUEST}})$, $\tau(\mathcal{M}_{\mathsf{REQUEST}})$, and $\tau(\mathcal{N}_{\mathsf{REQUEST}})$ are consolidated together. Prior to consolidation, the dimensions of $\tau(\mathcal{A}_{\mathsf{REQUEST}})$, $\tau(\mathcal{M}_{\mathsf{REQUEST}})$, and $\tau(\mathcal{N}_{\mathsf{REQUEST}})$ are 10×10, 11×11, and 30×30, respectively.

Firstly, Participant *X* applies the algorithm integrate_table in Section 4.3.1 to integrate the three τ's. During the integration process, Participant *X* found 44 pairs of choice relations to which Proposition 4.1 can be applied. Examples of such pairs of choice relations are |Duration of Visit: $>$ 1 Day| $\not\boxtimes_{\{\mathcal{A}_{\mathsf{REQUEST}}\}}$ |Dates of Visit: Exclude Weekends and Public Holidays| and |Duration of Visit: $>$ 1 Day| $\boxtimes_{\{\mathcal{N}_{\mathsf{REQUEST}}\}}$ |Dates of Visit: Exclude Weekends and Public Holidays|. Consider the first choice relation. Its relational operator is "$\not\boxtimes$" because no complete thread in $\mathcal{A}_{\mathsf{REQUEST}}$ (see Figure 4.8 for a partial activity diagram) is associated with both the guard conditions "$>$ 1 Day" and "Exclude Weekends and Public Holidays". Now consider the second choice relation. According to $\mathcal{N}_{\mathsf{REQUEST}}$, the duration and the dates of visit are entered into VAS as separate inputs. Furthermore, for a visitor request which spans more than a day, it can include or exclude weekends and public holidays. This explains why the relational operator for the second choice relation is "$\boxtimes$". Applying Proposition 4.1(f) to this pair of choice relations results in |Duration of Visit: $>$ 1 Day| $\boxtimes_{\{\mathcal{A}_{\mathsf{REQUEST}},\mathcal{N}_{\mathsf{REQUEST}}\}}$ |Dates of Visit: Exclude Weekends and Public Holidays|. At the completion of integrate_table, 856 choice relations are stored

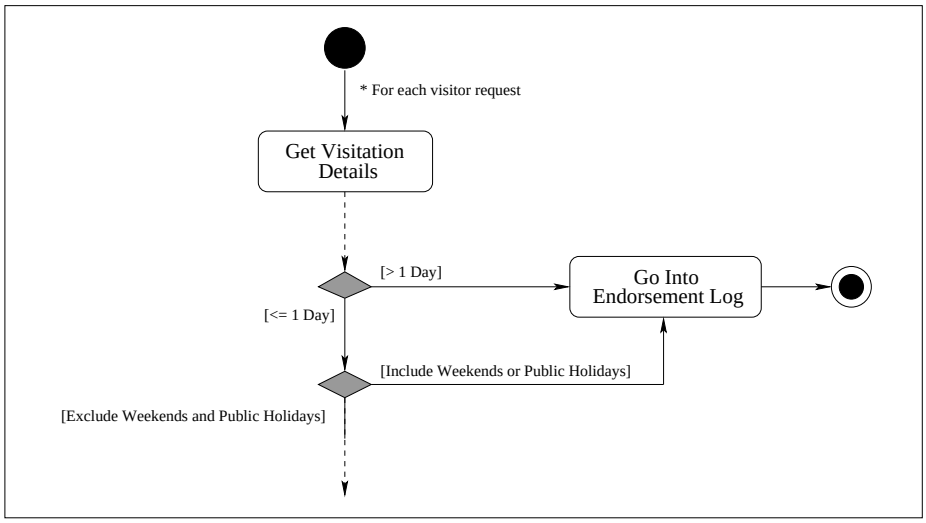


Figure 4.8: A Partial Activity Diagram $\mathcal{A}_{\mathsf{REQUEST}}$

in the linked list L and its associated linked lists. These choice relations involve a total of 32 distinct choices. Despite the large number of choice relations in the linked lists, no manual effort is required in this process because integrate_table can be fully automated.

By examining the choice relations stored in the linked lists, we found four pairs of overlapping choices. For each of these pairs, the two overlapping choices are defined from different $\mathbb{C}$'s and, hence, are not detected by Participant X in the early stages of the study. Below explains how these overlapping choices occur:

(a) In $\mathcal{A}_{\mathsf{REQUEST}}$, Participant X has found one decision point which is associated with two guard conditions "Within Office Hours" and "Outside Office Hours" (fulfilling the second guard condition will cause a visitor request to be stored in the endorsement log). With respect to these two guard conditions, Participant X has defined the category [Period of Visit], with |Period of Visit: Within Office Hours| and |Period of Visit: Outside Office Hours| as its associated choices. On the other hand, $\mathcal{N}_{\mathsf{REQUEST}}$ states that the starting and ending times of visit are entered into VAS as separate inputs. This information causes Participant X to define the category

[Period of Visit] with three associated choices, namely |Period of Visit: Within Office Hours|, |Period of Visit: Partially Outside Office Hours|, and |Period of Visit: Completely Outside Office Hours|. Since |Period of Visit: Outside Office Hours| = |Period of Visit: Partially Outside Office Hours| ∪ |Period of Visit: Completely Outside Office Hours|, we have the following two pairs of overlapping choices:

(i) |Period of Visit: Outside Office Hours| and |Period of Visit: Partially Outside Office Hours|; and

(ii) |Period of Visit: Outside Office Hours| and |Period of Visit: Completely Outside Office Hours|.

(b) $\mathcal{M}_{\mathsf{REQUEST}}$ (more specifically, $\mathcal{S}_{\mathsf{REQUEST}}$) indicates that there are two states of a previously issued visitor access card, namely "Returned" and "Not Yet Returned". This information results in the definition of the category [Previous Access Card] with |Previous Access Card: Returned| and |Previous Access Card: Not Yet Returned| as its two associated choices. On the other hand, $\mathcal{N}_{\mathsf{REQUEST}}$ states that VAS will process the visitor request for a *repeated* visitor differently, depending on the return status of the previously issued access card: (i) returned on time, (ii) returned late, and (iii) not yet returned. This information results in the definition of three choices |Previous Access Card: Returned On Time|, |Previous Access Card: Returned Late|, and |Previous Access Card: Not Yet Returned| for the category [Previous Access Card]. Since |Previous Access Card: Returned| = |Previous Access Card: Returned On Time| ∪ |Previous Access Card: Returned Late|, we have the following two pairs of overlapping choices:

(i) |Previous Access Card: Returned| and |Previous Access Card: Returned On Time|; and

(ii) |Previous Access Card: Returned| and |Previous Access Card: Returned Late|.

Altogether, we found 250 choice relations which involve overlapping choices. The numbers of choice relations associated with overlapping choices are: |Period of Visit: Outside Office Hours| (16), |Period of Visit: Partially Outside Office Hours| (54), |Period of Visit: Completely Outside Office Hours| (54), |Previous Access Card: Returned| (18), |Previous Access Card: Returned On Time| (54), and |Previous Access Card: Returned Late| (54).

Participant X thereafter applies the algorithm refine_table in Section 4.3.2 to refine these 250 choice relations involving overlapping choices. After step (2) of refine_table, none of the above overlapping choices exists. After step (3)(b) of refine_table, a choice relation table (denoted by $\mathcal{T}_{\mathsf{REQUEST}}$) for the function "Process Visitor Requests" of VAS, with a dimension of 30×30, is constructed. At this stage, among the 900 choice relations in $\mathcal{T}_{\mathsf{REQUEST}}$, 822 have been determined and 78 are yet-to-be-defined. All of the latter 78 choice relations, however, are deduced using the automatic deduction technique provided by CHOC'LATE [21, 62] in step (3)(c) of refine_table. Thus, no further manual definition of choice relations is needed. The resultant $\mathcal{T}_{\mathsf{REQUEST}}$, with all its 900 choice relations completely determined, can now be further processed by CHOC'LATE to generate a set of B^c's for testing the function "Process Visitor Requests" [21, 62].

4.5 Case Study 2

4.5.1 Setting of Study

In addition to the case study described in Section 4.4, we have performed another study based on the specification (denoted by $\mathbb{S}_{\mathsf{CODE}}$) of a computerized online directory enquiry system (denoted by CODE) used in a large telecom company. The system supports a maximum of 116 online inquiry terminals and handles more than 60 000 inquiries a day. CODE has four main components: a time-sharing executive module, an input-output

monitor, a language interpreter, and a tree-type database manager. An inquiry input consists of a primary name plus no more than two secondary names. Inquiries can be made in the following modes:

(a) Complete names: Full spelling of a name is used to search for entries.

(b) Incomplete names: For example, "JOHN$-$" can be used to search for entries such as "JOHN", "JOHNNY", or "JOHNSON".

(c) Alternative spellings: For example, "LAM$/$" can be used to search for entries such as "LAM", "LIM", or "LIN" (the slash symbol at the end of an entry indicates search by alternative spellings).

(d) Abbreviations: For example, "HK" instead of "HONG KONG" and "KLN" instead of "KOWLOON".

(e) Initials: For example, "SF" instead of "SAU FUN".

The specification $\mathbb{S}_{\mathsf{CODE}}$ is written primarily in an informal manner and contains narrative descriptions of the system, activity diagrams, sample input screens and outputs. This characteristic makes $\mathbb{S}_{\mathsf{CODE}}$ very useful for evaluating the effectiveness of DESSERT. The subject who performs this study is Participant X; the same person involved in our first study described in Section 4.4.

Initially, we planned to decompose $\mathbb{S}_{\mathsf{CODE}}$ into four $\mathbb{C}$'s: narrative descriptions of the system (denoted by $\mathcal{N}_{\mathsf{ENQUIRY}}$), an activity diagram for processing primary names (denoted by $\mathcal{A}_{\mathsf{P\text{-}ENQUIRY}}$), an activity diagram for processing secondary names (denoted by $\mathcal{A}_{\mathsf{S\text{-}ENQUIRY}}$), and sample input screens and outputs (denoted by I_{ENQUIRY}). Later we found that $\mathcal{N}_{\mathsf{ENQUIRY}}$ often makes reference to I_{ENQUIRY}. Thus, it would be more appropriate to consider both $\mathcal{N}_{\mathsf{ENQUIRY}}$ and I_{ENQUIRY} as one $\mathbb{C}$ to be denoted by $\mathcal{M}_{\mathsf{ENQUIRY}}$. In the end, we have three $\mathbb{C}$'s for table construction.

4.5.2 Constructing Preliminary Choice Relation Tables from Activity Diagrams

We have asked Participant X to construct the preliminary choice relation tables (denoted by $\tau(\mathcal{A}_{\text{P-ENQUIRY}})$ and $\tau(\mathcal{A}_{\text{S-ENQUIRY}})$) corresponding to $\mathcal{A}_{\text{P-ENQUIRY}}$ and $\mathcal{A}_{\text{S-ENQUIRY}}$, using the algorithm construct_table in Section 4.2.2.

After the table construction process, we found that Participant X has defined two potential categories and six potential choices for $\mathcal{A}_{\text{P-ENQUIRY}}$. Whereas for $\mathcal{A}_{\text{S-ENQUIRY}}$, Participant X has defined three potential categories and eight potential choices. We have detected no missing or problematic category/choice in $\tau(\mathcal{A}_{\text{P-ENQUIRY}})$ and $\tau(\mathcal{A}_{\text{S-ENQUIRY}})$, with respect to $\mathcal{A}_{\text{P-ENQUIRY}}$ and $\mathcal{A}_{\text{S-ENQUIRY}}$. In other words, all the potential categories are relevant and all the potential choices are valid. This result clearly demonstrates the effectiveness of the algorithm construct_table. Note the difference in the number of relevant categories and valid choices defined for $\mathcal{A}_{\text{P-ENQUIRY}}$ and $\mathcal{A}_{\text{S-ENQUIRY}}$. A reason for the difference is that some inquiry modes as mentioned in Section 4.5.1 are restricted to primary or secondary name processing only. For example, search by initials is only possible for secondary names.

4.5.3 Constructing Preliminary Choice Relation Table from Other Specification Components

Now, we turn to $\mathcal{M}_{\text{ENQUIRY}}$. We have asked Participant X to define a set of potential categories and potential choices in an ad hoc manner. Two potential categories, namely [Number of Secondary Names] and [Inquiry Mode], have been defined. The first potential category has two associated potential choices: |Number of Secondary Names: $= 0$| and |Number of Secondary Names: $= 1$ or 2|. The second potential category has five associated potential choices: |Inquiry Mode: Complete Name|, |Inquiry Mode: Incomplete

Name|, |Inquiry Mode: Alternative Spelling|, |Inquiry Mode: Abbreviation|, and |Inquiry Mode: Initials|.

With the use of our identification checklist provided in Section 3.1.5, we found that [Inquiry Mode] is a composite category because not all inquiry modes are possible for both primary and secondary name searches. To alleviate this problem, we decompose [Inquiry Mode] into the following two relevant categories:

- [Inquiry Mode for Primary Name]: Its four associated valid choices are |Inquiry Mode for Primary Name: Complete Name|, |Inquiry Mode for Primary Name: Incomplete Name|, |Inquiry Mode for Primary Name: Alternative Spelling|, and |Inquiry Mode for Primary Name: Abbreviation|.

- [Inquiry Mode for Secondary Name]: Its three associated valid choices are |Inquiry Mode for Secondary Name: Complete Name|, |Inquiry Mode for Secondary Name: Alternative Spelling|, and |Inquiry Mode for Secondary Name: Initials|.

After making the above correction, Participant X then proceeds to construct a preliminary choice relation table (denoted by $\tau(\mathcal{M}_{\mathsf{ENQUIRY}})$) for $\mathcal{M}_{\mathsf{ENQUIRY}}$, by determining the choice relation between every pair of valid choices.

Table 4.9 shows the number of relevant categories and valid choices for the three $\mathbb{C}$'s after the correction. We found that some relevant categories and valid choices defined from different $\mathbb{C}$'s are identical. After counting, we found five relevant categories and 15 valid choices which are all distinct. Furthermore, all these five relevant categories cannot be defined from any of the three $\mathbb{C}$'s alone. This observation is consistent with our earlier argument that an individual $\mathbb{C}$ may carry incomplete information of the software system.

	$\mathcal{A}_{\text{P-ENQUIRY}}$	$\mathcal{A}_{\text{S-ENQUIRY}}$	$\mathcal{M}_{\text{ENQUIRY}}$
Number of Relevant Categories	2	3	3
Number of Valid Choices	6	8	9

Table 4.9: Number of Relevant Categories and Valid Choices for $\mathbb{S}_{\text{CODE}}$ (After Correction)

4.5.4 Consolidating Preliminary Choice Relation Tables

Before consolidating $\tau(\mathcal{A}_{\text{P-ENQUIRY}})$, $\tau(\mathcal{A}_{\text{S-ENQUIRY}})$, and $\tau(\mathcal{M}_{\text{ENQUIRY}})$, the dimensions of these tables are 6×6, 8×8, and 9×9, respectively. Participant X first applies the algorithm integrate_table in Section 4.3.1 to integrate the three τ's. We found six pairs of choice relations to which Proposition 4.1 can be applied. Examples are |Number of Secondary Names: $= 0$| $\not\subseteq_{\{\mathcal{A}_{\text{S-ENQUIRY}}\}}$ |Inquiry Mode for Secondary Name: Complete Name| and |Number of Secondary Names: $= 0$| $\not\subseteq_{\{\mathcal{M}_{\text{ENQUIRY}}\}}$ |Inquiry Mode for Secondary Name: Complete Name|. Applying Proposition 4.1(c) to this pair of choice relations results in |Number of Secondary Names: $= 0$| $\not\subseteq_{\{\mathcal{A}_{\text{S-ENQUIRY}},\mathcal{M}_{\text{ENQUIRY}}\}}$ |Inquiry Mode for Secondary Name: Complete Name|. Note that integrate_table is fully automatic and, hence, no manual effort is required in this process. At the completion of integrate_table, 104 choice relations are stored in the linked list L and its associated linked lists. These choice relations involve a total of 15 distinct choices.

Regarding the choice relations stored in the linked lists, we found two pairs of overlapping choices. The two overlapping choices in each pair are defined from different $\mathbb{C}$'s and, hence, are not detected by Participant X when considering each $\mathbb{C}$ alone. The following explains how these overlapping choices occur:

(a) It is stated in $\mathcal{M}_{\text{ENQUIRY}}$ that an inquiry input consists of a primary name plus no more than two secondary names. In view of this information, Participant X has defined the category [Number of Secondary Names], with |Number of Secondary

Names: $= 0|$ and |Number of Secondary Names: $= 1$ or $2|$ as its associated choices. Note that $\mathcal{M}_{\text{ENQUIRY}}$ does not indicate that the CODE system will execute differently between one and two secondary names.

(b) Later, Participant X has found a decision point with three associated guard conditions "Number of Secondary Names $= 0$", "Number of Secondary Names $= 1$", and "Number of Secondary Names $= 2$" in $\mathcal{A}_{\text{S-ENQUIRY}}$.[15] The last two guard conditions clearly indicate that CODE executes differently between one and two secondary names. Thus, with respect to $\mathcal{A}_{\text{S-ENQUIRY}}$, Participant X has defined the category [Number of Secondary Names] with three associated choices, namely |Number of Secondary Names: $= 0|$, |Number of Secondary Names: $= 1|$, and |Number of Secondary Names: $= 2|$.

In (a) and (b) above, |Number of Secondary Names: $= 1$ or $2|$ overlaps with |Number of Secondary Names: $= 1|$ and |Number of Secondary Names: $= 2|$. This is because |Number of Secondary Names: $= 1| \subsetneq$ |Number of Secondary Names: $= 1$ or $2|$ and |Number of Secondary Names: $= 2| \subsetneq$ |Number of Secondary Names: $= 1$ or $2|$.

Altogether, we found 34 choice relations which involve overlapping choices. The numbers of choice relations associated with overlapping choices are: |Number of Secondary Names: $= 1|$ (10), |Number of Secondary Names: $= 2|$ (10), and |Number of Secondary Names: $= 1$ or $2|$ (14).

Participant X then applies the algorithm refine_table in Section 4.3.2 to refine these 34 choice relations involving overlapping choices. After step (2) of refine_table, none of these overlapping choices exists. In addition, after step (3)(c) of refine_table, a choice relation table (denoted by $\mathcal{T}_{\text{ENQUIRY}}$) for the CODE system, with a dimension of 14×14, is constructed. The resultant $\mathcal{T}_{\text{ENQUIRY}}$, with all its 196 ($= 14 \times 14$) choice relations

[15] If an inquiry input contains two secondary names, CODE will check whether both secondary names are associated with the same inquiry mode. If not, CODE will reject the inquiry input.

completely determined, can now be further processed by CHOC'LATE to generate a set of B^c's for testing the CODE system [21, 62].

4.6 Summary

In this chapter, we have introduced a **D**ivid**E**-and-conquer methodology for identifying categorie**S**, choice**S**, and choic**E** **R**elations for **T**est case generation (abbreviated as DESSERT). The purpose is to alleviate the major problems of CHOC'LATE and CTM, thereby improving the effectiveness of testing. The divide-and-conquer approach of DESSERT should appeal to software practitioners because the methodology can be effectively applied to large commercial software systems whose specifications are often complex and contain many different types of components ($\mathbb{C}$'s).

We have discussed in detail how to:

(a) construct preliminary choice relation tables (τ's) from activity diagrams,

(b) consolidate τ's constructed from different $\mathbb{C}$'s into a choice relation table ($\mathcal{T}$),

(c) correct inconsistent relations for the same pair of choices due to incomplete information of individual $\mathbb{C}$'s,

(d) refine choice relations involving overlapping choices defined from different $\mathbb{C}$'s, and

(e) apply consistency checks and automatic deductions in the construction of $\mathcal{T}$.

The theoretical backbone and techniques underlying these procedures have also been discussed.

We have also performed two case studies to evaluate DESSERT, using the specifications (containing many different $\mathbb{C}$'s) of real-life commercial software systems. The

positive results of the studies have confirmed that DESSERT provides a systematic approach to define a comprehensive set of categories, choices, and choice relations; the set so defined in turn forms the basis for constructing a $\mathcal{T}$. Once a $\mathcal{T}$ is constructed, it will then be processed by CHOC'LATE for test case generation.

Chapter 5

Conclusions

5.1 Summary of Results and Contributions

This thesis focuses on two closely related black-box testing methods, namely the choice relation framework (CHOC'LATE) [21, 62] and the classification-tree methodology (CTM) [10, 11, 20, 39, 40]. In both methods, software testers first identify a set of categories and their associated choices based on the influencing factors of the software system (or its functional units) under test. Constraints among the choices (in CHOC'LATE) and the categories (in CTM) are then identified. Such constraints are used to guide the generation of complete test frames by means of some predefined algorithms. Finally, complete test frames are used to generate test cases.

Without doubt, the chance of detecting software faults depends on the comprehensiveness of the generated test cases, which in turn depends on how well the categories and the choices are identified. Despite of the importance of categories and choices, we observe that a systematic method for their identification from informal specifications ($\mathbb{S}$'s) does not exist. As a result, the identification process is often performed in an ad hoc, impromptu manner. We argue that an impromptu identification approach cannot assure the quality of

the identified categories and choices.

To verify our argument, we have performed two rounds of empirical studies, involving inexperienced and experienced testers, and with the use of three commercial $\mathbb{S}$'s. There are several important observations from the studies. Among these observations, three are worth mentioning here. Firstly, both inexperienced and experienced testers have made various types of mistake when they use an impromptu approach to identifying categories and choices from informal $\mathbb{S}$'s [15]. This observation confirms the great demand for a systematic identification technique for informal $\mathbb{S}$'s. Secondly, although experience in software development and testing does help improve the quality of the identified categories and choices under an impromptu identification approach, the contribution of experience to the reduction of mistakes decreases with the complexity of $\mathbb{S}$'s. Thirdly, experienced testers are not necessarily better than inexperienced ones in every aspect. The latter two observations together suggest that software development cannot substitute the demand for a systematic identification methodology.

In our empirical studies, we have asked the experienced subjects about the usefulness of individual specification components ($\mathbb{C}$'s) for identifying categories and choices. The subjects favor activity diagrams, swimlane diagrams, sample input screens, and data dictionaries in various extent. They also suggest that, in order to improve the effectiveness of an impromptu approach, the identification of categories, choices, and constraints between categories/choices should not be considered separately.

Inspired by the above observations and findings, we have developed a **D**ivid**E**-and-conquer methodology for identifying categorie**S**, choice**S**, and choic**E** **R**elations for **T**est case generation (abbreviated as DESSERT).[1] A major merit of DESSERT is that it can be applied to complex $\mathbb{S}$'s with many different types of component. We have also performed

[1] There are three algorithms in DESSERT, namely construct_table, integrate_table, and refine_table. The first algorithm construct_table only works with activity diagrams, while the other two algorithms do not have this limitation.

two case studies using commercial $\mathbb{S}$'s written primarily in an informal manner, with a view to evaluating the effectiveness of DESSERT. Both studies have confirmed the usefulness of DESSERT in identifying categories and choices from informal $\mathbb{S}$'s, particularly when the $\mathbb{S}$'s are complex and contain many $\mathbb{C}$'s of different types.

It is well known that, in the commercial sector, the black-box testing approach is more common than the white-box approach. This is because of the shortage of experienced users of the automated tools needed for white-box testing [24], and the inaccessibility of the source codes of the off-the-shelf software packages under test. Furthermore, informal $\mathbb{S}$'s are more common than formal $\mathbb{S}$'s in industry [60, 82]. Therefore, those studies on white-box testing, as well as black-box testing based on formal $\mathbb{S}$'s, do not have a widespread application in the commercial sector. There is an urgent need for black-box testing techniques that can be applied to informal $\mathbb{S}$'s. In this regard, CHOC'LATE and CTM satisfy such a need because they can be applied to informal $\mathbb{S}$'s. With our work on category and choice identification via DESSERT, the applicability and the effectiveness of CHOC'LATE and CTM can be further increased.

5.2 Future work

Two possible directions of research that are of great significance are systematic identification techniques for $\mathbb{C}$'s other than activity diagrams, and the application of CHOC'LATE to requirements inspection.

In Chapter 4, we have provided a technique for identifying categories and choices from activity diagrams. In addition to activity diagrams, a typical $\mathbb{S}$ often contains other types of $\mathbb{C}$ such as use cases, class diagrams, state machines, data flow diagrams, system flowcharts, and decision tables. Thus, the need for identification techniques for these $\mathbb{C}$'s is strong. In this regard, the results of our empirical studies (particularly the different

types of problematic category and choice) discussed in Chapter 3 and our identification technique for activity diagrams in Chapter 4, will provide useful information that will shed light on the development of identification techniques for other $\mathbb{C}$'s.

Research into how to apply CHOC'LATE to requirements inspection, is another path for further study. Without doubt, the quality of a requirements specification has a great impact on the quality of the software developed. Therefore, a requirements specification should be complete, correct, consistent, and unambiguous. Otherwise, defects may remain undetected, resulting in the delivery of a faulty software system to the users. Motivated by this, some researchers have developed a perspective-based reading (PBR) technique to help identify defects in requirements specifications [4, 71]. Consequently, we have proposed a "problem-driven" approach for supporting PBR, involving the use of CTM [17, 18]. We have also done a case study using a commercial specification, with a view to determining the viability of the approach. The results of our study is promising. Because CHOC'LATE and CTM are two closely related black-box testing methods, we conjecture that CHOC'LATE may also be applied for supporting PBR. Again this is a potentially fruitful area for future research.

Bibliography

[1] R. L. Baber, *The Spine of Software: Designing Provably Correct Software — Theory and Practice*. Chichester, UK: Wiley, 1987.

[2] R. L. Baber, *Error-free Software: Know-how and Know-why of Program Correctness*. Chichester, UK: Wiley, 1991.

[3] R. Bache and M. Müllerburg, "Measures of testability as a basis for quality assurance," *Software Engineering Journal*, vol. 5, no. 2, pp. 86–92, 1990.

[4] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgard, and M. V. Zelkowitz, "The empirical investigation of perspective-based reading," *Empirical Software Engineering: An International Journal*, vol. 1, no. 2, pp. 133–164, 1996.

[5] B. Beizer, *Software Testing Techniques*. New York, NY: Van Nostrand Reinhold, 1990.

[6] J. M. Bieman and J. L. Schultz, "Estimating the number of test cases required to satisfy the all-du-paths testing criterion," *Proceedings of the ACM SIGSOFT '89 3rd Symposium on Software Testing, Analysis, and Verification* (*TAV3*), pp. 179–186. New York, NY: ACM Press, 1989.

[7] B. W. Boehm, "Industrial software metrics top 10 list," *IEEE Software*, vol. 4, no. 5, pp. 84–85, 1987.

[8] B. W. Boehm and V. R. Basili, "Software defect reduction top 10 list," *IEEE Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[9] L. Bougé, N. Choquet, L. Fribourg, and M. C. Gaudel, "Test sets generation from algebraic specifications using logic programming," *Journal of Systems and Software*, vol. 6, no. 4, pp. 343–360, 1986.

[10] A. Cain, T. Y. Chen, D. Grant, P.-L. Poon, **S.-F. Tang**, and T. H. Tse, "ADDICT: a prototype system for automated test data generation using the integrated classification-tree methodology," *Proceedings of the 1st ACIS International Conference on Software Engineering Research and Applications* (*SERA 2003*), pp. 76–81. Mt. Pleasant, MI: International Association for Computer and Information Science, 2003.

[11] A. Cain, T. Y. Chen, D. Grant, P.-L. Poon, **S.-F. Tang**, and T. H. Tse, "An automated test data generation system based on the integrated classification-tree methodology," *Software Engineering Research and Applications*, Lecture Notes in Computer Science, vol. 3026, pp. 225–238. Berlin: Springer, 2004.

[12] H. Y. Chen, T. H. Tse, F. T. Chan, and T. Y. Chen, "In black and white: an integrated approach to class-level testing of object-oriented programs," *ACM Transactions on Software Engineering and Methodology*, vol. 7, no. 3, pp. 250–295, 1998.

[13] T. Y. Chen and M. F. Lau, "Two test data selection strategies towards testing of boolean specifications," *Proceedings of the 21st Annual International Computer Software and Applications Conference* (*COMPSAC 97*), pp. 608–611. Los Alamitos, CA: IEEE Computer Society Press, 1997.

[14] T. Y. Chen, P.-L. Poon, and **S.-F. Tang**, "A systematic method for auditing user acceptance tests," *IS Audit and Control Journal*, vol. 5, pp. 31–36, 1998.

[15] T. Y. Chen, P.-L. Poon, **S.-F. Tang**, and T. H. Tse, "On the identification of categories and choices for specification-based test case generation," *Information and Software Technology*, vol. 46, no. 13, pp. 887–898, 2004.

[16] T. Y. Chen, P.-L. Poon, **S.-F. Tang**, and T. H. Tse, "Identification of categories and choices in activity diagrams," *Proceedings of the 5th International Conference on Quality Software* (*QSIC 2005*), pp. 55–63. Los Alamitos, CA: IEEE Computer Society Press, 2005.

[17] T. Y. Chen, P.-L. Poon, **S.-F. Tang**, T. H. Tse, and Y. T. Yu, "Towards a problem-driven approach to perspective-based reading," *Proceedings of the 7th IEEE International Symposium on High-Assurance Systems Engineering* (*HASE 2002*), pp. 221–229. Los Alamitos, CA: IEEE Computer Society Press, 2002.

[18] T. Y. Chen, P.-L. Poon, **S.-F. Tang**, T. H. Tse, and Y. T. Yu, "Applying testing to requirements inspection for software quality assurance," *Information Systems Control Journal*, vol. 6, pp. 50–56, 2006.

[19] T. Y. Chen, P.-L. Poon, **S.-F. Tang**, and Y. T. Yu, "White on black: a white-box-oriented approach for selecting black-box-generated test cases," *Proceedings of the 1st Asia-Pacific Conference on Quality Software* (*APAQS 2000*), pp. 275–284. Los Alamitos, CA: IEEE Computer Society Press, 2000.

[20] T. Y. Chen, P.-L. Poon, and T. H. Tse, "An integrated classification-tree methodology for test case generation," *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 6, pp. 647–679, 2000.

[21] T. Y. Chen, P.-L. Poon, and T. H. Tse, "A choice relation framework for supporting category-partition test case generation," *IEEE Transactions on Software Engineering*, vol. 29, no. 7, pp. 577–593, 2003.

[22] L. A. Clarke, "How do we improve software quality and how do we show that it matters?" *ACM Computing Surveys*, vol. 28, no. 4, p. 203, 1996.

[23] L. A. Clarke, J. Hassell, and D. J. Richardson, "A closer look at domain testing," *IEEE Transactions on Software Engineering*, vol. SE-8, no. 4, pp. 380–390, 1982.

[24] R. A. DeMillo, W. M. McCracken, R. J. Martin, and J. F. Passafiume, *Software Testing and Evaluation*. Menlo Park, CA: Benjamin/Cummings, 1987.

[25] J. Dick and A. Faivre, "Automating the generation and sequencing of test cases from model-based specifications," *Proceedings of FME '93: Industrial-Strength Formal Methods*, Lecture Notes in Computer Science, vol. 670, pp. 268–284. Berlin: Springer, 1993.

[26] E. P. Doolan, "Experience with Fagan's inspection method," *Software: Practice and Experience*, vol. 22, no. 2, pp. 173–182, 1992.

[27] R. H. Dunn and R. S. Ullman, *TQM for Computer Software*. New York, NY: McGraw-Hill, 1994.

[28] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *IEEE Transactions on Software Engineering*, vol. 10, no. 4, pp. 438–444, 1984.

[29] J. W. Duran and J. Wiorkowski, "Quantifying software validity by sampling," *IEEE Transactions on Reliability*, vol. 29, no. 2, pp. 141–144, 1980.

[30] W. R. Elmendorf, "Functional analysis using cause-effect graphs," *Proceedings of SHARE XLIII*, pp. 567–577. New York, NY: SHARE, 1974.

[31] R. Ferguson and B. Korel, "The chaining approach for software test data generation," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 1, pp. 63–86, 1996.

[32] L. M. Foreman and S. H. Zweben, "A study of the effectiveness of control and data flow testing strategies," *Journal of Systems and Software*, vol. 21, no. 3, pp. 215–228, 1993.

[33] A. F. Frank, L. S. Buchwald, and F. H. Lewski, "Software inspections: an effective verification process," *IEEE Software*, vol. 6, no. 3, pp. 31–36, 1989.

[34] D. P. Freedman and G. M. Weinberg, *Handbook of Walkthroughs, Inspections and Technical Reviews*. New York, NY: Dorset House, 1990.

[35] D. Gelperin and B. Hetzel, "The growth of software testing," *Communications of the ACM*, vol. 31, no. 6, pp. 687–695, 1988.

[36] T. Gilb and D. Graham, *Software Inspection*. Wokingham, UK: Addison Wesley, 1993.

[37] A. L. Goel, "Software reliability models: assumptions, limitations, and applicability," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, pp. 1411–1423, 1985.

[38] A. Griffiths and R. Morello, "The use of formal methods in a commercial V & V consultancy," *Proceedings of the Asia-Pacific Software Engineering Conference* (*APSEC 96*), pp. 184–193. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[39] M. Grochtmann and K. Grimm, "Classification trees for partition testing," *Software Testing, Verification and Reliability*, vol. 3, no. 2, pp. 63–82, 1993.

[40] M. Grochtmann, J. Wegener, and K. Grimm, "Test case design using classification trees and the classification-tree editor CTE," *Proceedings of the 8th International Software Quality Week* (*QW 95*). San Francisco, CA: Software Research Institute, 1995.

[41] M. Grottke and K. S. Trivedi, "Fighting bugs: remove, retry, replicate, and rejuvenate," *IEEE Computer*, vol. 40, no. 2, pp. 107–109, 2007.

[42] R. Hamlet, "Random testing," *Encyclopedia of Software Engineering*, (J. J. Marciniak, Ed.), pp. 970–978. New York, NY: Wiley, 1994.

[43] R. G. Hamlet, "An essay on software testing for quality assurance — editor's introduction," *Annals of Software Engineering*, vol. 4, no. 0, pp. 1–9, 1997.

[44] R. M. Hierons, M. Harman, and H. Singh, "Automatically generating information from a Z specification to support the classification tree method," *Proceedings of the 3rd International Conference of B and Z Users*, Lecture Notes in Computer Science, vol. 2651, pp. 388–407. Berlin: Springer, 2003.

[45] W. E. Howden, "A functional approach to program testing and analysis," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 10, pp. 997–1005, 1986.

[46] W. E. Howden, "Good enough versus high assurance software testing and evaluation methods," *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium* (*HASE 98*), pp. 166–175. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[47] J. C. King, "Program correctness: on inductive assertion methods," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 5, pp. 465–479, 1980.

[48] B. Korel, "Automated test data generation for programs with procedures," *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA 96*), ACM SIGSOFT Software Engineering Notes, vol. 21, no. 3, pp. 209–215. New York, NY: ACM Press, 1996.

[49] F.-C. Kuo, K.-Y. Sim, C.-A. Sun, **S.-F. Tang**, and Z. Q. Zhou, "Enhanced random testing for programs with high dimensional input domains," *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering* (*SEKE 2007*), pp. 135–140. Skokie, IL: Knowledge Systems Institute, 2007.

[50] J. W. Laski and B. Korel, "A data flow oriented program testing strategy," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 3, pp. 347–354, 1983.

[51] M. F. Lau and Y. T. Yu, "An extended fault class hierarchy for specification-based testing," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 3, pp. 247–276, 2005.

[52] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and J. M. Voas, "Estimating the probability of failure when testing reveals no failures," *IEEE Transactions on Software Engineering*, vol. 18, no. 1, pp. 33–43, 1992.

[53] J. D. Musa, "Operational profiles in software-reliability engineering," *IEEE Software*, vol. 10, no. 2, pp. 14–32, 1993.

[54] J. Musa, G. Fuoco, N. Irving, D. Kropfl, and B. Juhlin, "The operational profile," *Handbook of Software Reliability Engineering*, (M. R. Lyu, Ed.), pp. 167–216. New York, NY: McGraw-Hill, 1996.

[55] G. J. Myers, *The Art of Software Testing*. Hoboken, NJ: Wiley, 2004.

[56] National Research Council, *Computers at Risk: Safe Computing in the Information Age*. Washington DC: National Academies Press, 1991.

[57] P. G. Neumann, "The computer-related risk of the year: weak links and correlated events," *Proceedings of the 15th Annual International Computer Software and Applications Conference* (*COMPSAC 91*), pp. 5–8. Los Alamitos, CA: IEEE Computer Society Press, 1991.

[58] S. C. Ntafos, "A comparison of some structural testing strategies," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 868–874, 1988.

[59] Ö. Oskarsson and R. L. Glass, *An ISO 9000 Approach to Building Quality Software*. Upper Saddle River, NJ: Prentice Hall, 1995.

[60] A. M. Paradkar, K.-C. Tai, and M. A. Vouk, "Specification-based testing using cause-effect graphs," *Annals of Software Engineering*, vol. 4, pp. 133–157, 1997.

[61] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis (Eds.), *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison Wesley, 1995.

[62] P.-L. Poon, **S.-F. Tang**, T. H. Tse, and T. Y. Chen, "CHOC'LATE: a CHOiCe reLATion framEwork for specification-based testing," *Communications of the ACM* (accepted for publication).

[63] A. A. Porter, "Assessing software review meetings: results of a comparative analysis of two experimental studies," *IEEE Transactions on Software Engineering*, vol. 23, no. 3, pp. 129–145, 1997.

[64] A. A. Porter, H. P. Siy, C. A. Toman, and L. G. Votta, "An experiment to assess the cost-benefits of code inspections in large scale software development," *IEEE Transactions on Software Engineering*, vol. 23, no. 6, pp. 329–346, 1997.

[65] R. S. Pressman, *Software Engineering: a Practitioner's Approach*. New York, NY: McGraw-Hill, 2005.

[66] S. C. Reid, "An empirical analysis of equivalence partitioning, boundary value analysis and random testing," *Proceedings of the 4th International Software Metrics Symposium* (*METRICS 97*), pp. 64–73. Los Alamitos, CA: IEEE Computer Society Press, 1997.

[67] M. Roper, *Software Testing*. New York, NY: McGraw-Hill, 1994.

[68] G. W. Russell, "Experience with inspection in ultralarge-scale development," *IEEE Software*, vol. 8, no. 1, pp. 25–31, 1991.

[69] J. W. Sanders and E. Curran, *Software Quality: a Framework for Success in Software Development and Support*. Wokingham, UK: Addison Wesley, 1994.

[70] T. Shepard, M. Lamb, and D. Kelly, "More testing should be taught," *Communications of the ACM*, vol. 44, no. 6, pp. 103–108, 2001.

[71] F. Shull, I. Rus, and V. R. Basili, "How perspective-based reading can improve requirements inspections," *IEEE Computer*, vol. 33, no. 7, pp. 73–79, 2000.

[72] H. Singh, M. Conrad, and S. Sadeghipour, "Test case design based on Z and the classification-tree method," *Proceedings of the 1st International Conference on Formal Engineering Methods* (*ICFEM 97*), pp. 81–90. Los Alamitos, CA: IEEE Computer Society Press, 1997.

[73] K.-C. Tai, "Theory of fault-based predicate testing for computer programs," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 552–562, 1996.

[74] S. A. Vilkomir, K. Kapoor, and J. P. Bowen, "Tolerance of control-flow testing criteria," *Proceedings of the 27th Annual International Computer Software and Applications Conference* (*COMPSAC 2003*), pp. 182–187. Los Alamitos, CA: IEEE Computer Society Press, 2003.

[75] J. Wang, R. Hao, and J. Wu, "TUGEN: an automatic test suite generator integrating data-flow and control-flow methods," *Digital Technology — Spanning the Universe: Proceedings of the IEEE International Conference on Communications* (*ICC 98*), pp. 286–290. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[76] G. M. Weinberg and D. P. Freedman, "Reviews, walkthroughs, and inspections," *IEEE Transactions on Software Engineering*, vol. 10, no. 1, pp. 68–72, 1984.

[77] L. J. White and E. I. Cohen, "A domain strategy for computer program testing," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 3, pp. 247–257, 1980.

[78] J. A. Whittaker, "Stochastic software testing," *Annals of Software Engineering*, vol. 4, no. 0, pp. 115–131, 1997.

[79] M. Wood, M. Roper, A. Brooks, and J. Miller, "Comparing and combining software defect detection techniques: a replicated empirical study," *Proceedings of the Joint 6th European Software Engineering Conference and 5th ACM SIGSOFT International Symposium on Foundation of Software Engineering* (*ESEC'97/FSE-5*), ACM SIGSOFT Software Engineering Notes, vol. 22, no. 6, pp. 262–277. New York, NY: ACM Press, 1997.

[80] M. R. Woodward, D. Hedley, and M. A. Hennell, "Experience with path analysis and testing of programs," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 2, pp. 278–286, 1980.

[81] J. B. Wordsworth, *Software Development with Z: a Practical Approach to Formal Methods in Software Engineering*. Wokingham, UK: Addison Wesley, 1992.

[82] E. Yourdon, *Managing the System Life Cycle*. Englewood Cliffs, NJ: Yourdon Press, 1988.

[83] Y. T. Yu, **S.-F. Tang**, P.-L. Poon, and T. Y. Chen, "Improving the cost-effectiveness of a test suite for user acceptance tests," *Information Systems Control Journal*, vol. 6, pp. 32–37, 2000.

[84] Y. T. Yu, **S.-F. Tang**, P.-L. Poon, and T. Y. Chen, "A study of path-based strategy for selecting black-box generated test cases," *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 2, pp. 113–138, 2001.

[85] S. J. Zeil, "Selectivity of data-flow and control-flow path criteria," *Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis*, pp. 216–222. Los Alamitos, CA: IEEE Computer Society Press, 1988.

[86] M. V. Zelkowitz, "A functional correctness model of program verification," *IEEE Computer*, vol. 23, no. 11, pp. 30–39, 1990.