2007

Transformation of Formally Defined Post-Conditions into Target Language Statements

Swetha Padma Parvathaneni
Wright State University

TRANSFORMATION OF FORMALLY DEFINED POST-CONDITIONS INTO TARGET LANGUAGE STATEMENTS.

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

Swetha Padma Parvathaneni
Bachelor of Technology in Electronics and Communication Engineering,
Acharya Nagarjuna University, INDIA, 2005

2007
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

March 17th, 2007

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Swetha Parvathaneni ENTITLED Transformation of Formally Specified Post-Conditions into Target Language Statements BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

_____

Thomas C. Hartrum, Ph.D.
Thesis Co-Director

_____

Forouzan Golshani, Ph.D.
Department Chair

Committee on Final Examination

_____

Mateen M.Rizki, Ph.D.
Thesis Co-Director

_____

Krishnaprasad Thirunarayan, Ph.D.

_____

Joseph F.Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

ABSTRACT

For years software engineering researchers have been trying to come up with a software synthesis system that can transform a formal specification model into a design model from which executable code can be generated. AFIT Wide Spectrum Object Modeling Environment (AWSOME) is one such formal based software synthesis system. It uses a formal specification language called AWL (AFIT Wide Spectrum Language). In this system formal specifications written in AWL are parsed into an AST (Abstract Syntax Tree). To this AST transforms are applied to take it to a form from which code can be generated.

The intent of this thesis is to demonstrate the transformation of the post-conditions of a method into target language statements. The methods in the classes are specified using pre-conditions and post-conditions. A post condition is a Boolean predicate that must be true after the method has been executed. Transforms are developed to eliminate any post-condition that has set operators. After removing the post-condition from the method, statements that achieve the desired results specified by the post-condition are added to the method body. The result is a design model from which executable code can be generated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

It is my pleasure to thank many people who made this thesis possible. It is difficult to overstate my gratitude to my thesis advisor, Dr. Thomas C Hartrum. This thesis would not have been possible without his support and remarkable patience. I am very thankful to my committee members Dr. Mateen M.Rizki and Dr. Krishnaprasad Thirunarayan for their invaluable feedback on this work. I remain indebted to my family and friends for their support.

CHAPTER 1: INTRODUCTION

Over the years engineers have been trying to come up with a system that can take a language independent formal specification model into an executable language dependent code. Also it would be of great help if the maintenance is performed on the specification instead of on the implementation after it has been optimized, resulting in software which is harder to understand [1]. This puts forth the need to use an automation based paradigm in which we can see a more formalized and computer assisted software engineering process. Specification being the least complex and the closest to the users, they can more easily understand the changes made to it than to the implementation. With each change the revised specification is just re-implemented with the help of computer assistance. The decisions regarding the optimizations to be used are still made by the system analysts and the programmers. Since the changes are made to the specification and it can be re-implemented, we now have a system which is better documented and totally reusable. Aiming to achieve such a system, the Air Force Institute of Technology (AFIT) and Wright State University (WSU) have been developing a transformation system that generates executable code from a formal specification model. In this system, provided with the formal specification of a problem, transforms can be applied to it to obtain an executable code. It starts with a formal specification model that is developed by the application engineer. This model transforms into a design mode after the application of some correctness preserving transforms. Further application of other transforms results in an executable code.

ABOUT AWSOME

AWSOME is short for AFIT Wide Spectrum Object Modeling Environment [1]. It is a transformation system which uses a formal specification model to generate executable Java code. The AFIT transformation system is object-oriented, based in the Unified Modeling Language (UML). This tool is based on a meta-model represented using abstract syntax trees (AST). The formal language AFIT Wide-spectrum Language (AWL) is used to represent the formal specifications. The formal model is parsed into an AST. Then a series of transforms are performed on the AST to change it from a specification AST to a design AST from which executable code can be generated. The goal of ASWOME is to transform any formally correct representation of an object model into executable code. Currently the system has many transforms developed to change the AST from specification model to design model. Manubolu[6] and others [7][8][9][10] have designed transforms which can be applied to set operators present in the declarations but the set operators in the post-conditions, pre-conditions and invariants remain untouched.

PROBLEM STATEMENT

The methods in the classes are specified using pre-conditions and post-conditions. Transforms are to be designed to convert the post-conditions into statements so that we can get a design AST from which executable code can be generated. This thesis intends to transform those post-conditions that have set operators from the analysis model to the

design model. The specification language used is AWL. Post-conditions with sets are first transformed to post-conditions with arrays and then into statements which can be used to generate Java code.

LIMITATIONS OF THE THESIS

This thesis focuses only on the set operators in the post-conditions. Set operators may also be present in the pre-conditions or invariants but these issues are not addressed in this thesis although some of the transforms designed may do so as a side effect. It is assumed that the necessary transforms are already applied to the input AWL file so as to get it into the form required to apply the transforms designed in this thesis. The goal of this thesis is to provide transforms needed to transform the set operators in the post-conditions, but determining when and where to apply these transforms in general requires human interaction. However this research will provide computer help in the form of utility functions. In other words the goal is not to develop a fully automated system.

APPROACH

The approach used for the thesis is an object oriented approach. The initial phase of this approach is the requirement analysis phase in which the existing AWSOME tool is studied and the problem statement is carefully looked at in order to identify the transforms that are to be designed. In the next phase the transforms identified in the analysis phase are designed. These transforms are implemented and tested in the next

phase. The transforms are applied to different test cases and output model obtained is observed for correctness.

DOCUMENT ORGANIZATION

The thesis document starts with the Introduction chapter which briefly describes AWSOME. It presents the problem statement, and the approach involved to solve it is outlined here. The limitations of this thesis are also mentioned in this chapter. The next chapter (Background) presents the necessary background information to help the user understand about AWSOME and how the transformation from the specification model to the design model happens. Chapter 3 is Requirements Analysis in which the problem statement is looked at carefully and the necessary work to be done to solve it, that is, the transforms to be designed are identified. Next is the Design chapter in which the transforms are designed and discussed. Chapter 5 is Implementation and Testing in which the transforms are tested. This chapter presents the test cases used to test the transforms and the output obtained for each transform. The document ends with the Conclusions chapter which briefly reviews the work done so far. Known limitations of the transforms and any future work that can be done to improve the design are also discussed in this chapter.

CHAPTER 2: BACKGROUND

This chapter covers some of the background details regarding AWSOME and AWL. These details should be sufficient enough for the reader to understand the rest of the document.

WHAT IS AWSOME?

During the past several years WSU has been working on a formal-based software synthesis system initially developed at AFIT [1] called AWSOME. This transformation system is object-oriented based on Rumbaugh's Object Modeling Technique (OMT) and the Unified Modeling Language (UML). Given a formal specification of a problem this transformation system aims to generate executable code from it. The requirements are taken and using this, a formal specification model is developed. On this formal specification model some correctness preserving transforms are applied to transform it to a formal design model. Application of further transforms should generate the code.

The AWSOME tool is based on a meta-model represented using an abstract syntax tree (AST). An abstract syntax tree by definition is a data structure representing something which has been parsed, often used as a compiler or interpreter's internal representation of a computer program while it is being optimized and from which code generation is performed. It can be understood as the representation of the input (formal specification) on its way to output (target language program). AWSOME is the result of combining all the meta-model ASTs in the tool.

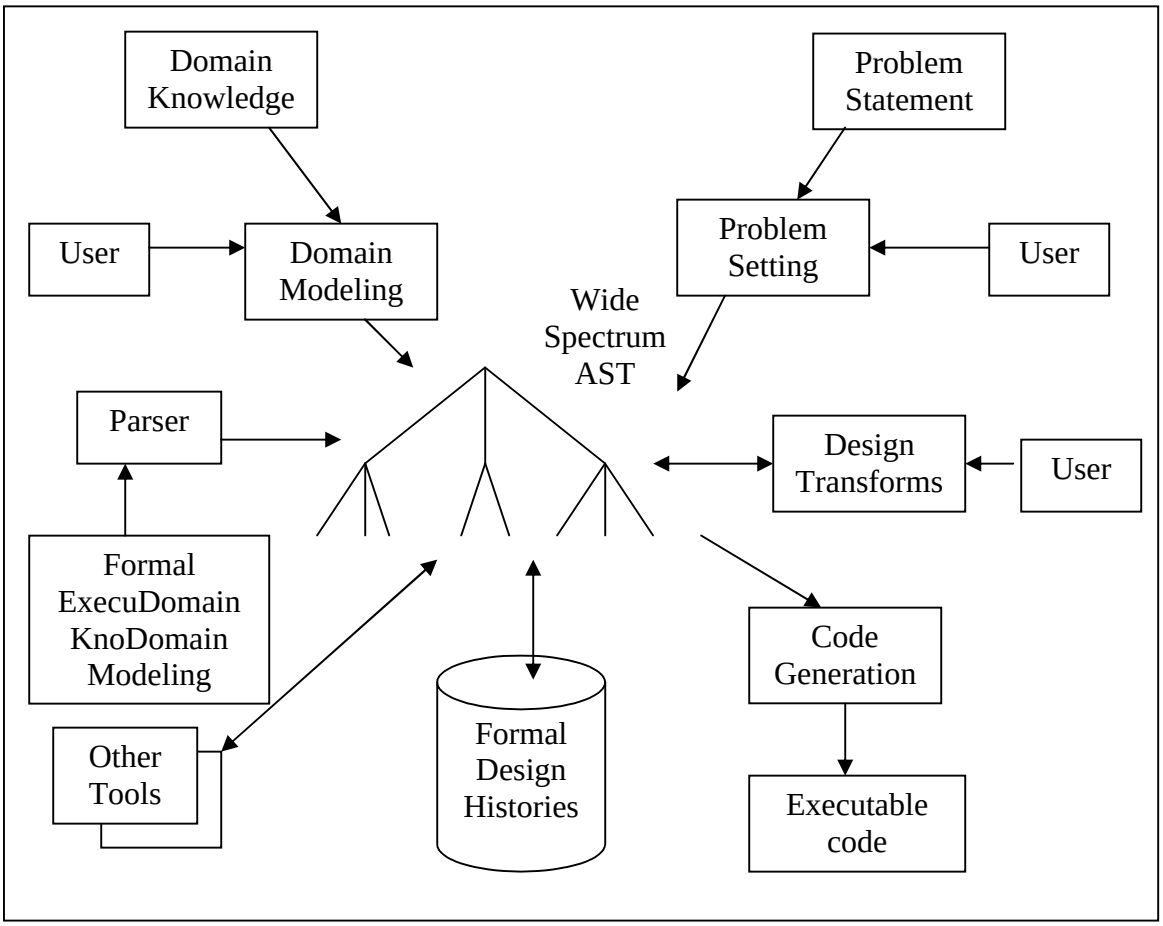

Figure 2.1: AWSOME Transformation System.

The AWSOME transformation system is built using Java. It is based on a single AST. AWSOME supports all the models from the requirements phase until the code phase. Currently in the code phase the target language is Java. Tools can be developed to generate code in other languages.

WHAT IS AWL?

A formal specification language is a mathematical notation used in software development to express the functional specification of a system. The specification defines *what* function is provided without saying *how* it will be provided. AWL (AFIT Wide-Spectrum Language) [2] is one such formal specification language. It was developed by AFIT to provide an easy user interface to AWSOME. AWL supports software synthesis from formal languages as well as reverse engineering of the already developed programs. AWL is defined by both its surface syntax and by its corresponding abstract syntax (AST) meta-model. AWL was designed as a wide spectrum language so that it can be used to write the formal specifications and also can be used as an intermediate language in the translation of one language to another. It has constructs to represent the associations between the classes, including aggregation, and a finite-state dynamic model. It also supports pre- and post-conditions and class invariants expressed using first order predicate logic and set theory. The major advantage that AWL has over the other specification languages is the parser. A parser for AWL syntax was developed using JavaCC, a Java-based compiler-compiler. Instead of just providing a way to write the initial formal specification, AWL lets any model modified in AWSOME to be saved in a parsable format.

THE AWSOME META-MODEL (AST)

When a formal specification (written in AWL) is parsed in, the parser produces a parse tree. The parser checks for syntax errors and then parses it into the AST. Correct application of a series of transformations on this AST results in a form from which executable code can be generated. The AWSOME model consists of a set of object classes. Each object class has a structural model (contains a set of attributes), a functional model (contains a set of methods) and a dynamic model (set of states and transitions).

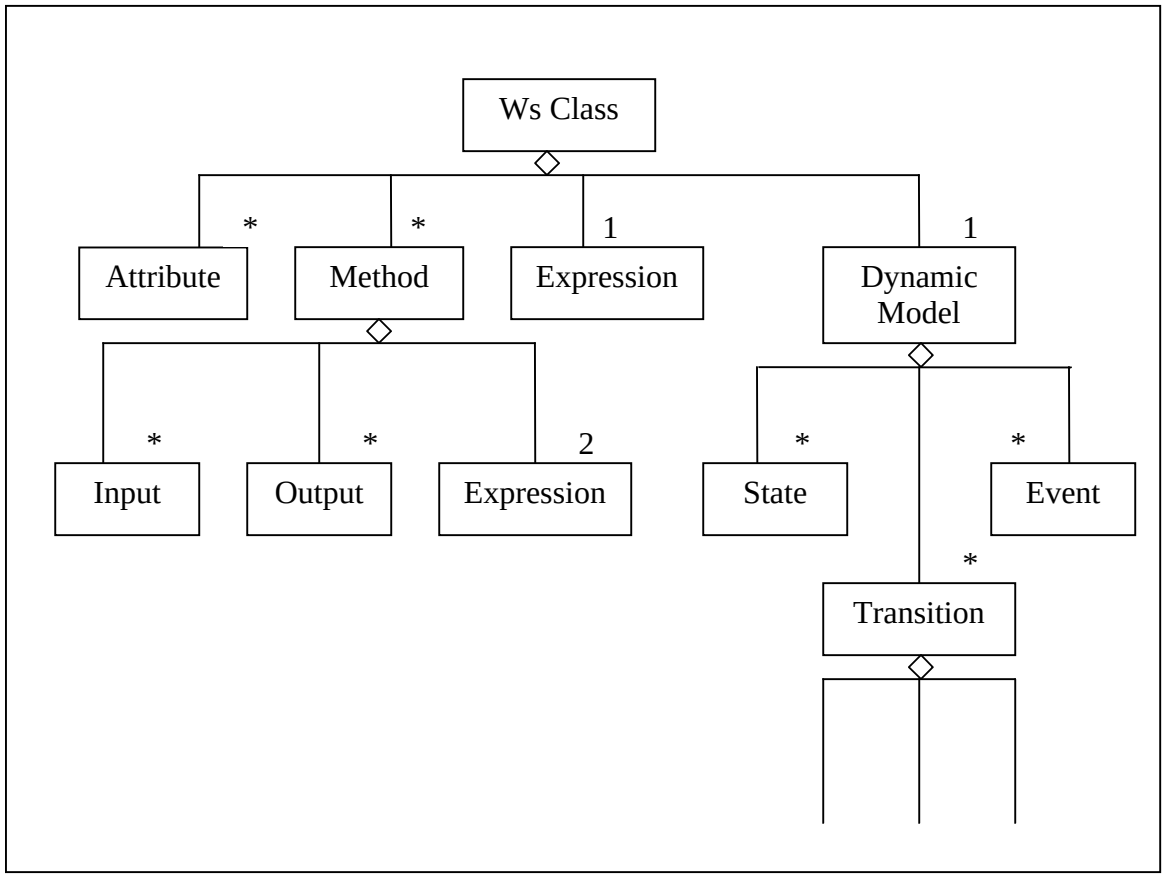

Figure 2.2: AWSOME Meta-Model for a Class

METHODS

The functional model of a class consists of a set of methods. A method is defined by the following: name, a set of input parameters, a set of output parameters, a precondition and a post condition.

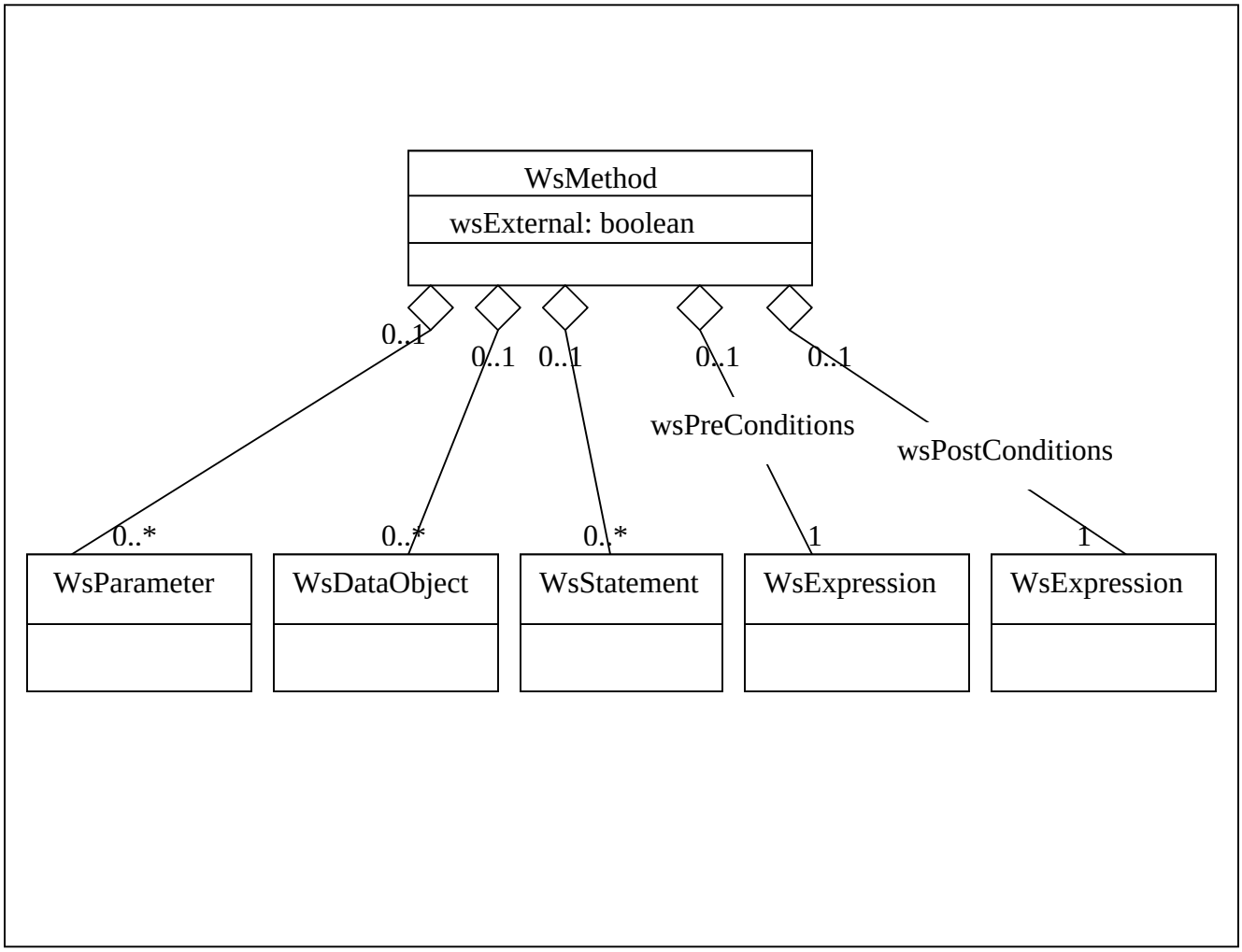

Figure 2.3: AWSOME Model for a Method

In an abstract form methods are described using pre- and post-conditions. A pre-condition is a Boolean expression that must be true before the method is called. A post-condition is also a Boolean expression that must be true after the method is executed. That is, the result of execution of the method is exactly the same as mentioned in the post condition. This thesis aims at designing transforms that can be applied to the post condition so that a sequence of statements can be obtained to replace the post condition.

SUMMARY

The background chapter discusses about AWSOME and AWL in detail. The information presented in this chapter should help the readers get familiar with AWSOME and AWL and should help them to understand the rest of the document.

CHAPTER 3: REQUIREMENTS ANALYSIS

This chapter presents a detailed description of methods and some possible ways to categorize them for a better understanding. Used mainly in object-oriented programming, the term method [15] refers to a piece of code that is exclusively associated either with a class (called class methods, static methods, or factory methods) or with an object (called instance methods). A method usually consists of a sequence of statements to perform an action, a set of input parameters to parameterize those actions along with local variables, and possibly an output value (called return value) of some kind. The purpose of methods is to provide a mechanism for accessing (for both reading and writing) the private data stored in an object or a class. A method should preserve the class invariants of the object it is associated with, and should always assume that they are valid when it commences execution. To this effect, preconditions are used to constrain the method's parameters and postconditions to constrain method's output.

Depending on whether the method returns a value or not we can categorize methods into functions (can return a value) and procedures (do not return any value). A better approach to categorize would be on the basis of modifications done to the attributes. Operations which merely compute a functional value without modifying any values, that is, operations with no side effects on the externally visible state of any object, are called pure functions. These are also called as queries. Those queries which have no arguments except for the target object may be regarded as derived attributes. An action is an operation which has side effects on the target object or other objects in the system reachable from the target object. It has no duration in time; it is logically instantaneous.

We can define an action in terms of the state of the object before and after the action. An activity is an operation to or by an object that has duration in time as a result of which it definitely has side effects. [12]

Simply put, a method in AWSOME is defined by the following: name, a set of input parameters, a set of output parameters, a set of local variables, a precondition and a post-condition. This thesis aims at designing transforms that can be applied to the post-condition so that a sequence of statements can be obtained to replace the post-condition. The pre-condition of a method specifies in a logical manner what restrictions the client invoking a particular method is obliged to comply with. That is, a precondition is a Boolean expression that must be true before the method is called. A post-condition is also a Boolean expression that must be true after the method is executed. That is, the result of execution of the method is exactly the same as mentioned in the post-condition.

SOURCES OF POST-CONDITIONS:

Post-conditions appear in the methods if they are:

- Directly written in the specification.
- Added by other transforms. This could be a result of applying transforms to the method's post-conditions. For example, the post-condition *(((A **and** B) or (**not** (A) **and** C)) and D)* on applying XformArjun8[7] can be changed to *(A => B **and not** A => C) **and** D*. This could also be due to other transforms. For example, Sarvepalli [10] adds the class invariant to the post-conditions of all methods that might change it. These transforms

could result in unexpected post-conditions, such as redundant terms or terms with unrelated variables. Another example is Manubolu's [6] transform that converts derived data types by adding the type constraint to method post-conditions.

- Implied. If there is an attribute that is not shown in a post-condition with a prime (or a tick) then it is implied that the value of the attribute after the operation remains the same as before the operation.

  Consider the following:

  class Test

  int a, b

  setA (**int** x)

  a' = x

  The post-condition in the above case is that after the operation 'setA' the value of 'a' needs to be set to x. Even though there is nothing in the post-condition that indicates the value of attribute 'b' after the operation, b' = b is conjuncted to a' = x (the actual post-condition in the specification), as it is implied that the value of b does not change after the operation.

EXPRESSIONS:

The post-conditions are generally made up of expressions. They can be either binary or unary, a combination of both, or quantified expressions. Binary expressions consist of numeric operators (**+, - , *, /, **, mod**), comparison operators (**=, /=, < , <= , < , >=**), set operators (**subset, subseteq, union, intersect, in**) and Boolean

operators (**and , or**). Unary operators are **not** and **unary minus.** The top level expression (the post-condition) must be a Boolean expression. [2]

Numeric operators have two operands of the same integer or real type except for **mod** which applies only to integers. Comparison operators take two operands of the same integer, real or enumeration type and result in a Boolean type. The set operators **subset** and **subseteq** take two operands of the same set type and results in a Boolean type. The **union** and **intersect** operators take in operands of the same set type and result in an operand of that set type. The **in** operator has the LHS as an element of set type and the RHS as a set type. It results in a Boolean. The Boolean operators take two Boolean operands and result in the Boolean type. [2]

The quantified expression defines a scope containing its logical variables and visible only to its constraint. The constraint must be Boolean. The result type is also Boolean. [2]

Syntax: (**forall | exists)** (*logical-variables*) (*constraint*)

Post-conditions

AWL provides only one expression for the post-condition. But this single expression can be made up of one expression or more expressions in conjunction. The following section gives an idea of the types of post-conditions. The post-condition can contain more than one expression, all of which have to be satisfied by the method on execution. These post-conditions can be either constraints or actions which are defined in

the next section. The result of the post condition is a Boolean. The following is the list of Boolean operators that result in a Boolean.

- Comparison operators (<, <=, >, >=, =, /=)
- AND, OR, NOT operators.
- Set operators (subset, subseteq, in)
- Implies operator (=>).
- Universal quantifiers (forall, exists)

CATEGORIES OF POST-CONDITIONS:

Post-conditions can be categorized as below:

- **Simplified and Non-Simplified**:

The post-conditions which can not be further split into smaller post-conditions are *simplified post-conditions*. (ex.: x' = a + b). If the post-condition can be further divided to form two or more post-conditions then it is a *non-simplified post-condition.* (ex.: (x' = a + b) **or** (y' = a + b) **and (z**' = b + 2)).

- **Dependent and Independent:**

The terms *dependent* and *independent* are defined below:

Dependent: If the post-condition is in the form **exp1(X') and exp2(X')** or **exp1(X) and exp2(X')** , then it can be categorized as a *dependent post-condition*.

Independent: The post-conditions which are in the form **exp1(X) and exp2(X)** are categorized as *independent post-conditions.*

Consider the following post-condition:

• x' = y **and** y' = x. : This is a *dependent post-condition*. The value of ticked variable in both the expressions is dependent on initial values of x and y.

• x' = y' + 1 **and** y' = 2 * x. : This is a *dependent post-condition*. The value of the ticked variable x' is dependent on the value of y', which is two times the value of x. It is obvious that the second expression needs to be evaluated before evaluating the first expression.

• (x' = x + 1 **and** y' = y + 2) **and** z' = z * z.: This is an *independent post-condition*. Any expression in the above post-conditions can be evaluated first. In other words there is no need to follow a particular order of evaluation.

- **Constraints and Actions:**

A *constraint* shows the relationship between two variables at the same time or between different values of the same variable at different times [12]. There are different ways to satisfy a constraint. An *action* is a post-condition that has a noticeable effect on the invoking object. While some of the expressions supported by AWL come under the *constraint* category, every method needs at least one action in the post-condition.

Table 3.1 provides information about expressions that are *constraints* and expressions that are *actions*.

Table 3.1: Classification of Boolean operators.

OPERATOR	CONSTRAINT	ACTION
LESS THAN (<)	YES	NO
LESS THAN OR EQUALS TO (<=)	YES	NO
GREATER THAN (>)	YES	NO
GREATER THAN OR EQUALS TO (>=)	YES	NO
EQUALS TO (=)	YES	YES
NOT EQUAL TO (/=)	YES	YES
NOT	YES	YES
IN	YES	YES
IMPLIES	YES	YES
FORALL	NO	YES
EXISTS	NO	YES
SUBSET	YES	NO
SUBSETEQ	YES	NO
AND	-	-
OR	-	-

The next section describes the Boolean operators in more detail.

BOOLEAN OPERATORS

COMPARISON OPERATORS: Comparison operators allow you to compare two values. A post-condition may contain one or more comparison operator (in combination with other Boolean operators.). The comparison operators (**<, <=, >=, >)** will always be treated as constraints. The operators **equals** (=) and **not equals** (/=) are special cases and will be discussed separately.

AND: It is used to put together two or more Boolean expressions to form the post-condition. If the post-condition is made of just two expressions with **and** as the operator between them, then we can just split it into two and then, treating each of them as a post-condition by itself, we can transform them separately. The main thing to keep in mind is the order of evaluating each of the expressions in the post-condition. This can be better understood with an example. Consider the following post-conditions:

- x' > y' **and** z' = z + 1.
- x' > y' **and** y' = y + 1.

The first post-condition is an *independent post-condition*. So there is no need to follow any particular order when evaluating the expressions. The second post-condition is a *dependent post-condition.* We need to evaluate y' = y + 1 before x' > y' because x' > y' is dependent on the value of y'.

So, if the post condition is *independent* then it is ok to split it into smaller post-conditions. Two transforms were developed to change the post-conditions containing **and** as the top level operator [7]. One of them (XformArjun4) only splits one expression at a time. The other transform (XformArjun5) splits all the post-conditions of the target

method that have **and** as the top level operator into separate post-conditions. If the post-condition is a *dependent post-condition,* it can either be left as it is or we can split it and then re-conjunct the dependencies.

OR: One transform (XformArjun8)[7] was developed to change the post-conditions of the form (expA **AND** expB) **OR** (**NOT** (expA) **AND** expC) to (expA => expB) **AND** (**NOT**(expA) => expC). There may be other special cases that can be transformed. In general, action OR expressions are not expected in action post-conditions. Most OR expressions are constraints (see discussion of EQUALS).

NOT: The **not** operator can only be used with an operand that evaluates to a Boolean value. It is one of those operators that can be both a constraint and an action. It can sometimes be a constraint leading to an action. Consider the following post-condition: **NOT**(x' /= a). This is equivalent to x' = a. The post-conditions with **not** as the top level operator can sometimes be split and simplified using de Morgan's laws. Consider the following post-conditions:

NOT (A **AND** B) is equivalent to **NOT** A **OR NOT** B

NOT (A **OR** B) is equivalent to **NOT** A **AND NOT** B

NOT (A **=>** B) is equivalent to A **AND NOT** B

NOT (a **in** B) is equivalent to B' = B \ {a}

NOT (**NOT** (A')) is equivalent to A'

These are all considered as simplifications that will be done prior to post-condition transforms. Thus, NOT will not be addressed in this thesis.

IMPLIES: The **implies** operator evaluates to true either if the left hand side is false or if both sides are true. Transforms were developed to change a post-condition with implication as the top level operator [7]. One transform (XformArjun6), changes the specified post-condition with **IMPLICATION** as the top level operator into an **IF THEN** statement. Another transform (XformArjun7) removes all the post-conditions that have **IMPLICATION** as top level operator and changes them to **IF THEN** statements.

SUBSET and **SUBSETEQ:** The result of an operation A **SUBSET** B is a Boolean value that is true if every element in A is present in B and false if otherwise. These operators are treated as constraints.

FORALL and **EXISTS**: These operators are treated as actions.

IN: The **in** operator has the LHS as an element of set type and the RHS as a set type. It results in a Boolean. Transforms (XformArjun10 and XformArjun11) change all the post-conditions that have **IN** as top level operator into procedureCall statements [7].

NOT EQUALS: The **not equals (/=)** operator will be treated just as a constraint. A post-condition which uses a **not equals** operator is not a sensible post-condition even though it might be a valid post-condition. Consider the following post-conditions:

- (x' /= 2 * x). The **not equals** operator is a constraint in this post-condition. This indicates that after the method execution, the value of x' can be anything other than 2 * x. This results in allowing one to assign more than one value for x' which is not deterministic.
- **NOT**(x' /= a). The **not equals** operator is a constraint that leads to an action in this post-condition.

EQUALS: The **equals (=)** operator is a constraint as well as an action. Consider the following post-conditions:

Ex1: x' = a. In this post-condition the **equals (=)** operator results in an action. A restriction is placed on the value of x' (must be a). This post-condition can be transformed to an assignment statement (x := a).

Ex2: The post-condition to be true for a triangle to be isosceles would be:

(a' = b') **or** (b' = c') **or** (c' = a'). In this post-condition, the constraint remains a constraint and there is no way it can be changed to an action.

NON-BOOLEAN EXPRESSIONS:

The post-condition can be decomposed using the Boolean operators. This results in bringing the post-condition into the form X' = exp. In order to convert the post-condition into a sequence of statements, we need to evaluate the exp (the expression on the R.H.S). This expression can be any of the following [2]:

- **ARITHMETIC EXPRESSIONS** – Expressions in which the operator involved is a numeric operator **(+, -, *, /, **, mod**).
- **SET EXPRESSIONS** – Expressions that contain the set operators (**union, intersect** and **container formers**).
- **FUNCTION CALLS** – Expressions that contain a function call.

- **LEAF EXPRESSIONS:** These are the lowest level expressions and can be one of the following:

  - **Literal Constants** – These include integers (1, 9, 87 … etc...), real (1.22, 5.3e+3.. etc…), characters (single characters (like a – z, A – Z, 0 -9, special characters and space) and escape sequences enclosed in single quotes), strings (sequence of characters enclosed in double quotes) and null.

  - **Variables** – These are the data objects that represent storage allocated for a value that can be changed, where the value can be of any type (abstract, enumeration, integer, real, derived type like container, access and class). Variables include the attributes of the class, arguments passed to the methods and the local variables in the method.

  - **Selected Components:** A selected component names a component in a package or class. These are expressions that use **dot** (.) notation i.e., the expressions in the form X.Y. Ex. **student1.name** where student1 is an object of type Student and name is an attribute in the Student class.
  - **Associations:** A leaf term can be also be an expression that can be related through an association(including aggregation), which is in the form X.Y.Z,

where X is the ‘local’ variable, Y is the ‘name’ of the association and Z is the role at the ‘other end’ of the association or an attribute name of an associative object. In the leaf term **man.married.wife**, ‘man’ is an attribute, argument of the method or a local variable in the method, ‘married’ is an association and ‘wife’ is the role at the other end of the association. The keyword **this** can be used in the leaf term instead of X. The result of a leaf expression similar to the above example ‘returns’ a set of values whose size depends on the multiplicity (if the multiplicity is one, a set containing a single value is returned).

Transforms for Set Operators

The focus of this thesis is to transform the set operators in the post-conditions from specification model to design model. Transforms are needed to convert the set operators like in, union, intersection, set difference, container formers, subset and subseteq into design model. Consider the example AWL file shown in Fig 3.1.

```
package aTestForSetOperations is
type CHAR is abstract;
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type sIndex is range 0 .. 200;
type StudentArray is array[stuIndex] of Student;
type StudentArray2 is array[sIndex] of Student;

class Student is
public sName : STRING;
end class;
```

Figure 3.1: Example AWL file

```
class School is
public roster : StudentArray;
public rosterMAX : stuIndex;
public roster2 : StudentArray;
public roster2MAX : stuIndex;

// Check if the student is in roster
public function isStudent(S : in Student) : Boolean
guarantees
(S in roster => check' = True)
and ( ( not  (S in roster ) ) => check' = False )
and isStudent' = check'
is
    check : Boolean;
begin
end;

// Add a student to the roster
public procedure addStudent(S : in Student)
guarantees
S in roster'

// Delete a student from roster
public procedure delStudent(S : in Student)
guarantees
not ( S in roster' )

*//Copy all the students from roster to another say rosterC
public procedure copyStudent(roster : in StudentArray, rosterMAX : in stuIndex,
                             copy : out StudentArray, copyMAX : out stuIndex)
guarantees
 copy' = roster

*// difference of 2 sets
public procedure diff(roster : in StudentArray, rosterMAX : in stuIndex,
                roster2 : in StudentArray, rosterMAX : in stuIndex,
                diff : out StudentArray, diffMAX : out stuIndex)
guarantees
diff' = roster - roster2

*//Union of roster and roster2
public procedure joinStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              join : out StudentArray, joinMAX : out sIndex)
guarantees
join' = roster union roster2

*//Intersection of roster and roster2
public procedure commonStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              common : out StudentArray, commonMAX : out stuIndex)
guarantees
common' = roster intersect roster2

end class;
end package;
```

Figure 3.2: Example AWL file

In the example shown in Fig 3.1 converting some of the post-conditions that involve set expressions to code statements is a little complex. Such post-conditions are first transformed into an alternate form that has arrays which are easier to handle. Some of the post-conditions may have more than one alternate form. Table 3.2 lists these post-conditions and their alternate forms. Transforms developed to perform these conversions are discussed in the section Set Transforms in Chapter 4.

Table 3.2: Post-conditions with Sets and their alternate forms.

Post-condition with Sets	Post-condition with Arrays(alternate forms)
(S in roster => check' = True) and ((not (S in roster)) => check' = False) and isStudent' = check'	exists(i : Bindex) (B[i] = A)=> check' = true and not (exists(i : Bindex) (B[i] = A)) => check' = false and isStudent' = check'
S in roster'	1. rosterMAX' = rosterMAX + 1 and roster ' [roster MAX'] = A 2. exists(i : BIndex) (B[i] = A)
not (S in roster')	1. exists(i : BIndex) ((B[i] = A) and (B'[i] = B[BMAX]) and (BMAX' = BMAX - 1)) 2. exists(i : BIndex) (B[i] = A)

Now we need to design transforms that convert the alternate forms into appropriate code statements. The post-conditions with set operators set difference, union and intersect can be directly transformed to code statements. These transforms are discussed in the section Array Transforms in Chapter 4.

In some cases, transforms are already developed to handle a part of the post-condition. Consider the isStudent() method shown in Fig 3.2. The post-condition can be spilt into three individual expressions using either XformArjun4 or XformArjun5. The

expression *isStudent' = check'* can be transformed into an assignment statement using XformF1. We need to design transforms that can transform the other two expressions either separately or combined. If the approach chosen is to conjunct the expressions and then transform them then there is need to develop some conjunction transforms. These transforms are discussed in the section Conjunction Transforms in Chapter 4.

Summary

This requirement analysis chapter provides information regarding methods, post-conditions and types, operators and expression types that appear in post-conditions. The existing transforms deal with the post-conditions that include the operators **and, or, implies** and **in.** New transforms should be designed to decompose the post-conditions that include the other Boolean operators, and to convert the decomposed post-condition containing the non-Boolean expression into a sequence of statements. Also, transforms are to be designed to put together the related parts of a post-condition, which result from applying transforms to a *dependent post-condition.* Finally, to convert the post-condition to a sequence of statements we need to evaluate the resultant non-Boolean expressions and leaf expressions.

CHAPTER 4: DESIGN

The previous chapter provides the necessary information regarding the formal methods. The sources of post-conditions, types of post-conditions and the existing transforms were also presented. The design chapter provides the design to meet the requirements mentioned in the last section of the previous chapter.

WORKING DEFINITIONS

In the requirement analysis chapter we categorized the post-conditions into action/constraint and dependent/independent. Creating precise definitions for these cannot be done. The present section provides working definitions to identify the different types of post-conditions of interest.

Action: A post-condition that can be made true by executing a sequence of statements that assign a unique value to the primed attributes is an action.

Constraints: All the post-conditions that are not actions are constraints.

Two guidelines are that a post-condition is identified as an action if either

1) it does not have the comparison operators >, <, >=, <=, subset, subseteq. These are usually constraints.
2) it has an equals (=) operator with one primed variable in the expression and the R.H.S of the expression is a leaf expression, arithmetic expression or function call.

Dependent: Two post-conditions that share a variable are dependent on one another if the shared variable is primed in one expression and unprimed in the other.

An example is: x' = x + 1 **and** y' = x' + 2

Independent: All the post-conditions that are not dependent are independent. Such independent post-conditions can be evaluated in any order without affecting the final output.

For this thesis, only action post-conditions are transformed. Constraints are assumed to be converted to derived pre-conditions. For this research, only independent post-conditions are directly transformed. Dependent post-conditions are conjuncted together to form independent ones, or are not handled.

TRANSFORM CLASS

In the AWSOME system, a transform is itself a first-class object. There is a top-level abstract class *Transform* [3] with a subclass for each specific transform. These are defined in the package *WsTransforms***.** The abstract class *Transform* contains string attribute *rationale* and concrete methods to set and get it. It also has Boolean attribute *applied* initialized to false, and concrete methods to set and get it. The *Transform* class also contains four concrete static methods (*getName(), getDesc(), applicable()* and *explain()*) which are intended to be overridden by every concrete subclass. The five abstract methods (*execute(), undo(), replay(), show()* and *toString()*) should be implemented by every concrete subclass. Of these methods, *applicable()* and *execute()* are the ones that are implemented for this thesis.

- *applicable()* – This method takes two parameters of type WsObject. It is a static method and returns a Boolean value which indicates whether the transform can be applied or not on the current selection. It is to be noted that this method checks for the applicability of the transform and not for the correctness.

- *execute()* – This method takes one parameter of type WsObject. It returns true on correct execution of the transform.

CONJUNCTION TRANSFORMS

In trying to convert post-conditions to executable target language statements we may come across some post-conditions that are too complex to convert. In such cases we need to de-conjunct the post-condition to make it simpler and at the same time taking care that we do not over simplify it. If two terms in the post-condition are independent of each other then they can be handled separately. If some terms are dependent on others, do not decompose them except those terms which are truly constraints and can be enforced via pre-condition. Such terms are not needed and can be separated.

Deciding on when to stop simplifying is tricky. Also, as an AWSOME expression is a binary expansion, related terms may be placed far apart in the tree. So the approach is to completely decompose and then recompose as needed. Decomposing is already accomplished using transforms XformArjun4 and XformArjun5 [7].

The following three transforms are designed to conjunct post-conditions in their respective capacities.

XformSwetha1: This transform is designed to conjunct two post-conditions selected by the user. The transform assumes that the selected post-conditions are related according to the user. If the user wants to conjunct more than two post-conditions he will have to choose transforms XformSwetha2 or XformSwetha3. Alternatively he could use the current transform repeatedly.

The methods implemented in the XformSwetha1 are:

a) **applicable():** This makes sure that the index of the post-conditions selected are valid. This method takes two input arguments.
    - Target method Object
    - Vector containing the indices of the two post-conditions.

b) **execute():** The execute method conjuncts the two post-conditions selected by the user. The resulting post-condition is added to the set of post-conditions and the individual post-conditions are removed from the set. The final set of post-conditions holds the resulting post-condition in addition other post-conditions in the set, if any. This method takes in one argument:
    - Vector containing the indices of the two post-conditions.

XformSwetha2: This transform is designed to conjunct more than two post-conditions selected by the user. The methods implemented in the XformSwetha2 are:

a) **applicable():** This makes sure that the index of the post-conditions selected are valid. This method takes two input arguments.
    - Target method Object
    - Vector containing the indices of the post-conditions which the user intends to conjunct.

b) **execute():** The execute method conjuncts all the post-conditions selected by the user. The resulting post-condition is added to the set of post-conditions and the

individual post-conditions are removed from the set. The final set of post-conditions holds the resulting post-condition in addition other post-conditions in the set, if any. This method takes in one argument:

- Vector containing the indices of the selected post-conditions.

XformSwetha3: This transform is designed to conjunct all the post-conditions in the target method. The methods implemented in the XformSwetha3 are:

a) **applicable():** This is a static method and it makes sure that the post-condition set contains more than one post-condition in order to indicate whether the XformSwetha3 is applicable or not. This method takes two input arguments.

- Target method Object
- Null.

b) **execute():** The execute method conjuncts all the post-conditions selected by the user. The resulting post-condition is added to the set of post-conditions and the individual post-conditions are removed from the set. The final set of post-conditions holds just the resulting post-condition. This method takes in one argument:

- Null.

SET TRANSFORMS

AWL supports container types which represent containers for other types. These include arrays, sequences, sets and bags. The transforms in this section intend to convert post-conditions involving set operations into postconditions that deal with arrays instead of sets [11]. Set operations like adding an element into a set, deleting an element from a set, checking for the presence of an element in a set are addressed here. It is assumed that the declarations are already converted from sets to arrays using transforms previously designed. A second assumption is that when an attribute with the type array A is declared, the class is provided with an attribute that is a pointer to the last element stored in the array named AMAX. The example shown in Fig. 4.1 and Fig 4.2 will be frequently referred to in the rest of the chapter. It is to be noted that in the methods marked with a '*' shown in this example there should be an additional post-condition updating the output MAX variable. Since this is intuitively obvious, the transform correctly updates it based on the post-condition shown.

```
package aTestForSetOperations is
type CHAR is abstract;
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type sIndex is range 0 .. 200;
type StudentArray is array[stuIndex] of Student;
type StudentArray2 is array[sIndex] of Student;

class Student is
public sName : STRING;
end class;
```

Figure 4.1: Example AWL file

```
class School is
public roster : StudentArray;
public rosterMAX : stuIndex;
public roster2 : StudentArray;
public roster2MAX : stuIndex;

// Check if the student is in roster
public function isStudent(S : in Student) : Boolean
guarantees
(S in roster => check' = True)
and ( ( not  (S in roster ) ) => check' = False )
and isStudent' = check'
is
    check : Boolean;
begin
end;

// Add a student to the roster
public procedure addStudent(S : in Student)
guarantees
S in roster'

// Delete a student from roster
public procedure delStudent(S : in Student)
guarantees
not ( S in roster' )

*//Copy all the students from roster to another say rosterC
public procedure copyStudent(roster : in StudentArray, rosterMAX : in stuIndex,
                             copy : out StudentArray, copyMAX : out stuIndex)
guarantees
 copy' = roster

*// difference of 2 sets
public procedure diff(roster : in StudentArray, rosterMAX : in stuIndex,
                roster2 : in StudentArray, rosterMAX : in stuIndex,
                diff : out StudentArray, diffMAX : out stuIndex)
guarantees
diff' = roster - roster2

*//Union of roster and roster2
public procedure joinStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              join : out StudentArray, joinMAX : out sIndex)
guarantees
join' = roster union roster2

*//Intersection of roster and roster2
public procedure commonStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              common : out StudentArray, commonMAX : out stuIndex)
guarantees
common' = roster intersect roster2

end class;
end package;
```

Figure 4.2: Example AWL file

XformSwetha4: This transform is designed to convert post-conditions in the form *A **in** B'* (where A is an element to be added to the set B) into a post-condition involving an array by incrementing the last index and placing the element A at that position. The resulting post-condition is in the form: *BMAX' = BMAX + 1 **and** B' [BMAX'] = A* where B is the array and BMAX is the pointer to the last element stored in the array.

a) **applicable():** This makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition has an **in** operator in it and the right operand is declared as an array in the declarations. This method takes two input arguments.
    - Target method Object
    - Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute creates references to the array and to the index of the last element in the array (arrayNameMAX). These identifier references along with the new element to be inserted into the array are passed to the newPostCon method. The WsExpression returned by the newPostCon is used to replace the selected post-condition. This method takes the following as the input argument.
    - Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

c) **newPostCon():**This is a method private to the class XformSwetha4 and is called from the *execute()* method. newPostCon creates the new expression (which has arrays) which is equivalent to the selected post-condition (which has sets). This newly formed expression which has the required output form is returned to the *execute()* method. This method takes three input arguments.

- Identifier Reference to the array object (WsIdentifierRef object)
- Identifier Reference to the index of the last element (WsIdentifierRef object)
- The new element to be added (WsExpression object)

XformSwetha5: This transform is designed to convert post-conditions in the form ***not****(A **in** B')* (where A is an element to be deleted from the set B) into a post-condition involving an array by replacing the element to be deleted by the last array element and decreasing the last index by 1. The resulting post-condition is in the form:

exists*(i : BIndex) ((B[i] = A)*

and *(B'[i] = B[BMAX])*

and *(BMAX' = BMAX - 1))*

where BIndex is the index type of the array B, 'i' is an logical variable of type BIndex.

a) **applicable():** This makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition has the required form and the left operand of the **in** operator is declared as an array in the declarations. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute creates references to the array and to the index of the last element in the array (arrayNameMAX). These identifier references along with the new element to be inserted into the array and the array index type are passed to the newPostCon method. The WsExpression returned by the newPostCon is used to replace the selected post-condition. This method takes the following as the input argument.

   - Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

c) **newPostCon():**This is a method private to the class XformSwetha5 and is called from the *execute()* method. newPostCon creates the new expression (which has arrays) which is equivalent to the selected post-condition (which has sets). This newly formed expression which has the required output form is returned to the *execute()* method. This method takes three input arguments.

   - Identifier Reference to the array object (WsIdentifierRef object)
   - Identifier Reference to the index of the last element (WsIdentifierRef object)

- Identifier Indicating the array type (WsIdentifier object)
- The new element to be added (WsExpression object)

XformSwetha6: This transform is designed to convert a post-condition that checks for the presence of the element in the set to a post-condition that checks for the presence of the element in the array. This transform requires the post-condition to be in the exact form shown below:

*(A **in** B => check' = true) **and** (**not** A **in** B) => check' = false **and** funcName' = check'*

where 'check' is a local Boolean variable and 'funcName' is the name of the current function. The post-condition after applying the transform is:

***exists**(i : Bindex) (B[i] = A)=> check' = true*

***and not** (**exists**(i : Bindex) (B[i] = A)) => check' = false*

***and** funcName' = check'*

a) **applicable():** This makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition has the required form and the left operand of the **in** operator is declared as an array in the declarations. The application of this transform is limited to those post-conditions which have the exact form as shown above. This method takes two input arguments.

   - Target method Object

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) execute(): Execute creates references to the array, the element to be checked, the array index type, along with the references to the true, false, local variable and the function name are passed to the newPostCon method. The WsExpression obtained is added to the post-condition set. The old post-condition (which has sets) is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

c) newPostCon(): This is a method private to the class XformSwetha6 and is called from the *execute()* method. newPostCon creates the new expression (which has arrays) which is equivalent to the selected post-condition (which has sets). This newly formed expression which has the required output form is returned to the *execute()* method. This method takes three input arguments.

- Identifier Reference to the array object (WsIdentifierRef object)
- Identifier Indicating the array type (WsIdentifier object)
- The new element to be added (WsExpression object)
- Vector containing the Identifier References to the following

- function name
- Boolean attribute
- true
- false

XformSwetha7: This transform is designed to find the post-conditions in the form *A **in** B* or *A **in** B'* in the current class and converts the **in** to an equivalent existential expression.

*A **in** B* is converted into ***exists**(i : BIndex) (B[i] = A)* and

*A **in** B'* is converted into ***exists**(i : BIndex) (B' [i] = A)*

a) **applicable():** This is a static method and it makes sure that the target class is an instance of WsClass. It returns a Boolean value. This method takes two input arguments.
   - Target class Object
   - Vector containing the WsPackage (AST) object.

b) **execute():**Execute creates a ChangeWsInVisitor object, passing in the current class and the vector containing the AST object as the arguments to the constructor. The *acceptVisitor()* method of the WsClass is called and the ChangeWsInVisitor object is passed. This method takes the following as the input argument.
   - Vector containing the WsPackage (AST) object.

ARRAY TRANSFORMS

In the previous section the set transforms convert post-conditions involving set operations into post-conditions that deal with arrays instead of sets. The array transforms in this section are designed to work on the post-conditions that deal with arrays and convert these post-conditions into code statements.

XformSwetha8: This transform is designed to convert specific equality expressions to an assignment statement. It requires that the ticked operand is an array. The following expression *B'[index position] = A* can be converted into *B[index position]:= A* where B is an array, A is the element to be placed in the position indicated by the variable 'index position'.

a) **applicable():** Applicable makes sure that the index of the post-condition selected is valid. It returns true if in the selected post-condition the ticked operand is an array.
    - Target method Object.
    - Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute uses the two operands of the equality expression selected to create a new assignment statement. The tick on the ticked operand is removed and the newly formed assignment statement is added to the target subprogram body. The post-condition is removed from the post-condition set. If this transform is

called from another transform then the equality expression to be converted can be set by calling the setEx() method. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

c) setEx(): This is a static, utility method and it sets the equality expression that has to be converted. If XformSwetha8 is called from another transform then the setEx() method must be called first to test for the applicability and to execute it. It takes a WsExpression object as the input argument.

XformSwetha9: This transform is designed to convert post-conditions in the form *BMAX' = BMAX + 1* ***and*** *B'[BMAX'] = A (*where A is the element to be added to the array B and BMAX indicates the index of the last element stored in the array B). On executing this transform the following code statements are added to the target method body: BMAX**:=** BMAX + 1

B[BMAX] **:=** A

XformSwetha9 calls the following three transforms.

i) XformArjun5 to split the post-condition into two equality expressions.

ii) XformF1 to convert BMAX' = BMAX + 1 to BMAX**:=** BMAX + 1.

iii) XformSwetha8 to convert B[BMAX'] = A to B[BMAX] **:=** A.

a) **applicable():** This makes sure that the index of the post-conditions selected are valid. It returns true if the selected post-condition has the required form and if BMAX matches in the two conjuncted expressions and if the transforms XformArjun5, XformF1, XformSwetha8 are applicable. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute calls the transform XformF1 to convert the first part BMAX' = BMAX + 1 and then calls the Transform XformSwetha8 to convert B'[BMAX'] = A. The post-condition is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha10: This transform is designed to convert post-conditions in the form *X' = **exists**(i : Bindex)(B[i] = A)* (where B is an array) to code statements which search for the element A in the array B and X is the target subprogram name. The value of X is assigned to true or false depending on the result of the search.

a) **applicable():** This makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute transforms the selected post-condition into code statements which include an iterative statement . In this while loop a search for the presence of the element in the array is performed until there are no more elements in the array to compare with (this is upper limit is indicated by the MAX variable of the array BMAX). If the element A is found then the Boolean value true is assigned to X. If the element is not found then X is assigned false. The selected post-condition is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha11: This transform is designed to convert the post-conditions in the form ***exists****(i : Bindex)(B'[i] = A)* (where B is an array) to code statements that increment the BMAX variable by one and insert the element A in the position indicated by BMAX.

a) **applicable():** This makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute creates references to the array and to the index of the last element in the array. These identifier references along with the new element to be inserted into the array are passed to the newPostCon method. The WsExpression (*BMAX' = BMAX + 1* ***and*** *B'[BMAX'] = A)* returned from the newPostCon method is transformed using XformSwetha9. On successful execution of XformSwetha11 the target method body has two assignment statements included into it. One of the assignment statements increment the value of BMAX by one and the second statement places the element A into the array B at the position indicated by the updated BMAX variable. The post-condition is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

c) **newPostCon():**newPostCon creates the new expression which is equivalent to the selected post-condition. This method is private to XformSwetha11. The newly created expression has two equality expressions that are conjuncted(*BMAX' = BMAX + 1* ***and*** *B'[BMAX'] = A).* This expression is returned to the *execute()* method. This method takes three input arguments.

- Identifier Reference to the array object (WsIdentifierRef object)
- Identifier Reference to the index of the last element (WsIdentifierRef object)
- The new element to be added (WsExpression object.)

XformSwetha12: This transform is designed to convert the post-conditions in the form ***exists*** *(i : Bindex) ((B[i] = A)* ***and*** *exp1* ***and*** *exp2* ***and*** *exp3...)* where B is an array, 'i' is a variable of the type Bindex and whose value ranges from 0 to BMAX. On applying this transform the body of the target method includes a loop in which every element in the array B is compared to the element A. If the element is found then a procedure call is made to a new procedure (which is added to the class operations by this transform).

a) **applicable():** This is a static method and it makes sure that the index of the post-conditions selected are valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.

- Target method Object

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) execute(): Execute modifies the body of the target method. The constraint expression in the existential expression is decomposed and those expressions that follow *B[i] = A* are recomposed. A while statement is added to the method body. In this while loop each element in the array B is compared to the element A. If the element is found then a procedure call is made to the procedure doImplementationX (). The doImplementationX () is created by execute() and added to the class operations. The doImplementationX () has the variable 'i' as its formal parameter and the recomposed expressions (*exp1* ***and*** *exp2* ***and*** *exp3..*) are added to this method's post-condition set. The while loop terminates when all the elements in the array are compared. The index of the last elements stored in the array is indicated by the BMAX. The post-condition is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha13 and XformSwetha14 are logical transforms and are discussed in the next section.

XformSwetha15: This transform is designed to convert the post-conditions in the form ***forall****(i : D) Q* (where D is the index type of i and Q is WsExpression). This transform adds a set of code statements (to the target method) in which for each value of 'i' the function doImplementationX () is called. The doImplementationX () with a post-condition Q is added to the class operations by the XformSwetha15. The correctness of this transform depends on applying XformSwetha16.

a) **applicable():** This is a static method and it makes sure that the index of the post-conditions selected are valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.
    - Target method Object
    - Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute transforms the universal quantifier into a while loop that terminates when it reaches the upper limit of the variable 'i'. A new function doImplementationX () is created and added to the class operations. The formal parameters of this function are 'i' and the parameters of the target method. The expression *Q* becomes the post-condition for doImplementationX (). doImplementationX () returns a Boolean value. If it returns true the loop terminates. The post-condition is removed from the post-condition set. The WsExpression. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha16: This transform is designed to convert the post-conditions in the form ***not****(B'[i] = A)*. XformSwetha16 adds code statements to the target method body. These code statements compare the element at i^{th} position in the array B with the element A. If they match then that element is replaced by the last element stored in the array.

a) applicable(): Applicable makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) execute(): Execute transforms the selected post-condition into code statements. The element in the i^{th} position in the array B is compared with the element A and if they match then it is replaced by the last element stored in the array. The pointer to the last element stored in the array BMAX is decremented by 1. The post-condition is removed from the post-condition set.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha17 is a logical transform and is discussed in the next section.

XformSwetha18: This transform is designed to convert the post-conditions in the form $X' = A$ (where X and A are arrays) to code statements. After applying this transform code statements that copy every element in the array A to the corresponding position in array X are added to the target method body.

a) **applicable():** This is a static method and it makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition has the required form and X and A are declared as arrays in the declarations. This method takes two input arguments.
   - Target method Object
   - Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute transforms the selected post-condition into code statements. The post-condition is removed from the post-condition set. The code statements

have a loop in which i^{th} iteration copies the elements in the i^{th} position in array A into the i^{th} position in array X. Element at A[0] is copied to X[0], element at A[1] is copied to X[1] and so on. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha19: This transform is designed to convert the post-condition that finds the difference of two sets. The post-condition needs to be in the form $X' = A - B$. Due to the lack of a proper set difference operator the subtract operator is used to represent the set difference. The post-condition does not contain an expression (*XMAX' = position of the last element stored in X)* updating the MAX variable but the execute() method adds a statement to the method body which updates the XMAX.

a) applicable(): This is a static method and it makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition has the required form and X, A and B are declared as arrays in the declarations. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) execute(): Execute adds code statements to the target method body. These code statements compare each element in the set A with each element in the set B. If the element in A is not found in B then that element is placed in X at the first empty position. When there are no more elements to compare in A the MAX variable XMAX is updated such that it now points to the last element stored in X. The post-condition is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha20: This transform is designed to convert the post-condition that finds the union of two sets into code statements. The post-condition needs to be in the form *X' = A **union** B.* The post-condition does not contain an expression (*XMAX' = position of the last element stored in X)* updating the MAX variable but the execute() method adds a statement to the method body which updates the XMAX.

a) applicable(): This is a static method and it makes sure that the index of the post-condition selected is valid in order to indicate whether the XformSwetha21 is applicable on the current selection or not. Returns true if the selected post-condition is in the required form and X, A and B are declared as arrays in the declarations. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) execute(): Execute copies all the elements in A into X. Next it compares each element in B with every element in X. If the compared B element is not found in X then it is placed in the first empty position in X. If the element is found then the next element in B is compared with every element in X. When there are no more elements to compare in B the MAX variable XMAX is updated such that it now points to the last element stored in X. The post-condition is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha21: This transform is designed to convert the post-condition that finds the intersection of two sets into code statements. The post-condition needs to be in the form $X' = A$ ***intersect*** B.

a) applicable(): This is a static method and it makes sure that the index of the post-condition selected is valid in order to indicate whether the XformSwetha22 is

applicable on the current selection or not. Returns true if the selected post-condition is in the required form and X, A and B are declared as arrays in the declarations. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute adds code statements to the target method body. These code statements compare each element in the set A with each element in the set B. If the element in A is found in B then that element is placed in X at the first empty position. When there are no more elements to compare in A the MAX variable XMAX is updated such that it now points to the last element stored in X. The post-condition is removed from the post-condition set. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

LOGICAL TRANSFORMS

The logical transforms are designed to replace specific post-conditions with an alternate form that is easier to handle. The result of both the replaced post-

condition and the replacing post-condition is same. The post-conditions shown below are transformed using the logical transforms.

- ***not(exists** (i : D) Q)*
- *P => X' = true **and not** P => X' = false*

XformSwetha13 transforms those post-conditions having the form ***not(exists** (i : D) Q)* into ***forall**(i : D) **not** Q.* Transform XformSwetha15 can be used to convert the post-conditions that have the form ***forall**(i : D) **not** Q* into corresponding code statements. XformSwetha14 is the inverse of XformSwetha13.

XformSwetha17 is designed to convert the post-condition *P => X' = true **and not** P => X' = false* into its logical equivalent *X' = P*.

XformSwetha13: This transform is designed to convert the post-condition in the form ***not(exists** (i : D) Q)* to ***forall**(i : D) **not** Q.*

a) **applicable():** This is a static method and it makes sure that the index of the post-conditions selected are valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.
   - Target method Object
   - Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) execute(): Execute transforms the selected post-condition containing an existential into a post-condition containing an universal quantifier. It creates a new WsExpression ***not*** *Q*. A WsUniversal object is constructed with ***not*** *Q* as the constraint and *i : D* is added to the declarations. The old post-condition is removed from the post-condition set and the new post-condition is added. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha14: This transform does the exact opposite of what XformSwetha13. It converts the post-condition in the form ***forall****(i : D)* ***not*** *Q* to ***not(exists*** *(i : D) Q).*

a) applicable(): This is a static method and it makes sure that the index of the post-condition selected is valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) execute(): Execute transforms the selected post-condition containing an universal quantifier into a post-condition containing an existential universal quantifier. This

method creates a new WsExistential object with Q as the constraint and *i : D* is added to the declarations. Next a WsNot object with the existential object as its operand. The old post-condition is removed from the post-condition set and the new post-condition is added. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

XformSwetha17: This transform is designed to convert the post-condition in the form *P =>(X' = true)* ***and not*** *P => (X' = false)* to *X' = P* (where X is the name of the subprogram for which this is a post-condition or a local variable, P is a WsExpression).

a) applicable(): This makes sure that the index of the post-conditions selected are valid. It returns true if the selected post-condition is in the required form. This method takes two input arguments.

- Target method Object
- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

b) **execute():**Execute transforms the selected post-condition into a equality expression with X and P as its operands. This method takes the following as the input argument.

- Vector containing the WsClass object (current class where the target method is defined), the WsPackage (AST) object and index of the selected post-condition.

SUMMARY

The design chapter provides a detailed description of the design decisions involved in the design of the transforms. The results of the methods *applicable()* and *execute()* are described for each transform. Three conjunction transforms are developed to conjunct the selected post-conditions or all the post-conditions as needed. Set transforms are developed to convert post-conditions involving set operations into postconditions that deal with arrays instead of sets. Transforms are designed to convert the post-conditions that have arrays into code statements. Using these transforms, post-conditions that guarantee the set operations like adding an element to a set, deleting an element from a set, checking for the presence of an element in a set, union of two sets, intersection of two sets and the difference of two sets can be converted into code statements that achieve the same when executed. In addition to the set and array transforms four logical transforms are designed to replace some post-conditions into their logical equivalent. Use of these transforms may be clearer after explaining the test case examples in the next chapter.

CHAPTER 5: IMPLEMENTATION AND TESTING

The design chapter provided a description of the design decisions involved in the design of the transforms. The current chapter discusses the implementation of the transforms, the testing approach used and the tools involved. The test cases used for each transform and the results obtained are also included. The code for all the transforms is provided in the appendix.

TESTING ENVIRONMENT

To test the transforms a case tool shown in Fig 5.1 is used. It can hold all the tools required for testing. To use a tool we just need to click on it. The most important tool is the **ToolMethod1** which opens a GUI application that plays a key part in the testing. Some of the supporting tools used during the testing are **ToolLoader**, **ToolToAWL** and **ToolToOutline**. **ToolLoader** is used to parse and load the input file, which is a specification model written in AWL, to AST. This tool takes AWL file as input. **ToolToAWL** writes the AST back to an AWL file. The name of the output file can be specified by the user. **ToolToOutline** prints the AST to a file in outline format. Here again the name of the output file can be chosen. All these tools are a part of a package called **toolbox** [4].

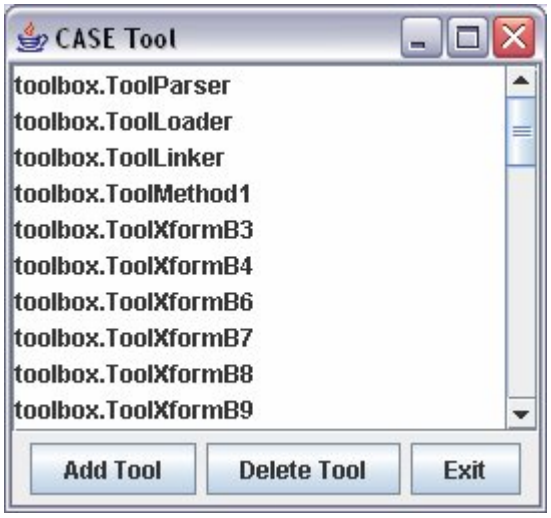

Figure 5.1: Toolbox.

The **ToolMethod1** GUI contains a lot of buttons, each of which on being clicked applies a corresponding transform. Fields are provided at the top to display the target package name, target class name (if more than one class is present in the package, a drop menu holding all the available classes is provided from which a class can be selected) and the target method name (if more than one method is present in the class, a drop menu holding all the available methods is provided from which a method can be selected). The center section of the GUI is comprised of two text areas. The right text area displays the target class (including all the attributes and the subprograms) and the text area on the left displays the AST of the target class in outline format. Below these text areas there are two more text areas of which one displays the set of pre conditions and the other displays the set of post conditions associated with the target method. Fig 5.2 is a screenshot of the GUI (an example class is loaded before using the **ToolMethod1** to generate it)

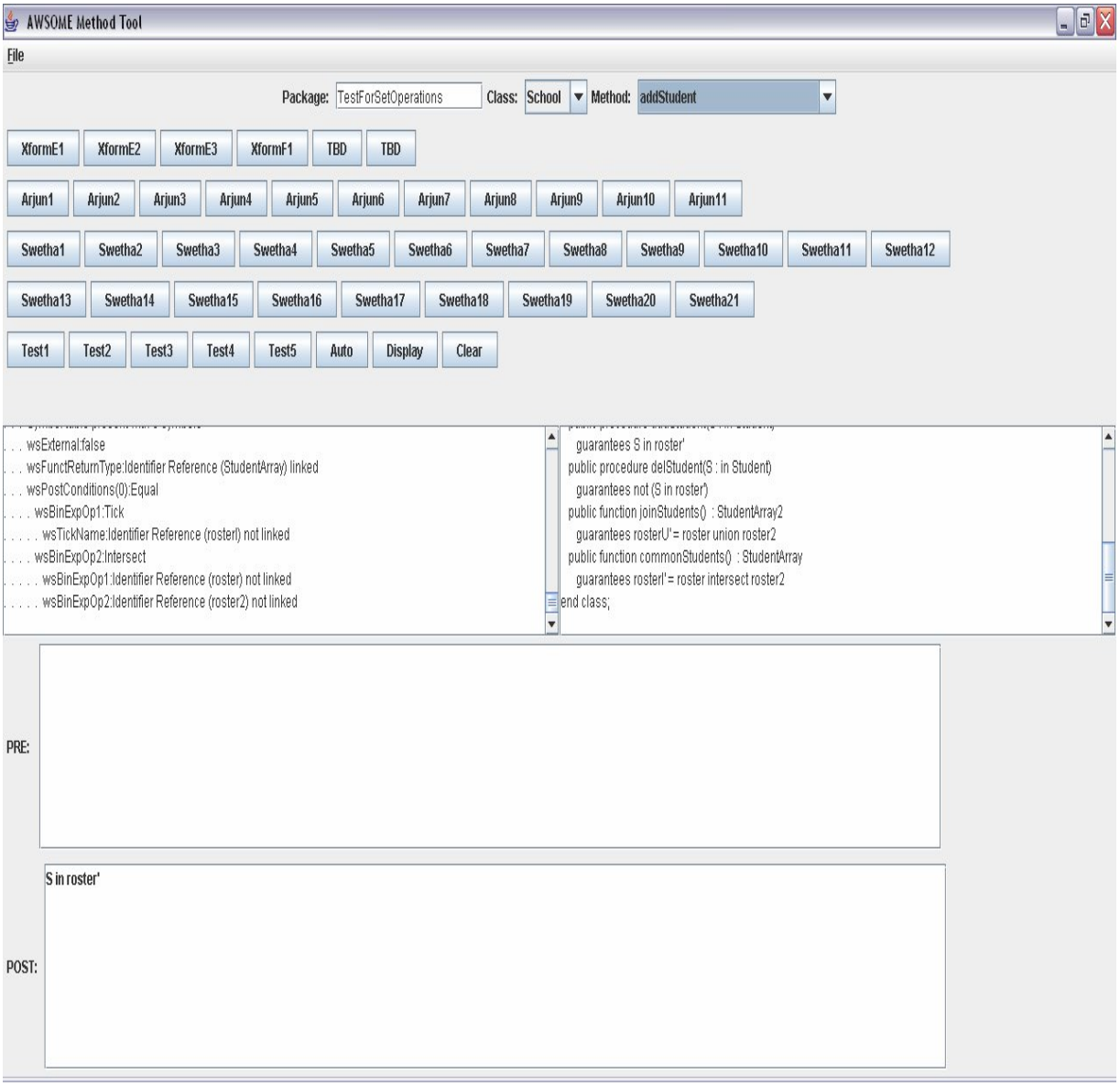

Figure 5.2: User Interface

Each of the buttons corresponds to a transform. The button names are given such that **Swetha1** corresponds to **XformSwetha1 Swetha2** corresponds to **XformSwetha2** and so on. The additional buttons in the GUI correspond to other transforms developed earlier. Some of these transforms are used in the process of testing.

TESTING

Each of the transforms developed in the thesis is tested. Test cases are written and used to test the transforms. The test case is a formal specification file written in AWL. To test a transform first the test case needs to be loaded. This is done by invoking the **ToolLoader** tool. A File Chooser is used to navigate to the file to be loaded. After the file is successfully parsed and loaded the ToolMethod1 tool is invoked to launch the GUI.

The target class and the target method are chosen from the drop menu and a post condition is selected. Clicking on the buttons will call the *applicable()* method of the corresponding transform. If the transform is applicable, then it is executed and the result will be either an alternate post condition or code statements inserted in the target method and target class. To check the modifications either the **ToolToAWL** or the **ToolToOutline** can be used. The **ToolToAWL** will write the AST back to an AWL file which will reflect the changes made to the input AWL file. This is mostly used since it is in a user readable form. The **ToolToOutline** will print the AST to a file which will reflect the changes made to the AST. This is used to verify the internal changes to the AST as needed. The following is a description of the test cases used for each transform and the result obtained.

TESTING THE CONJUNCTION TRANSFORMS

For testing purposes XformArjun5 [7] is applied on the conjuncted post condition to get a set of individual post conditions which are used to test the conjunction transforms. It is assumed that the selected post conditions are related.

XformSwetha1: This transform conjuncts two post-conditions selected by the user. It can be tested by clicking on the button Swetha1 which calls on the *applicable()* method first and depending on the boolean value returned the *execute()* method is called to run the transform. Figure 5.3 is an AWL specification file used as a test case.

```
   package library is
     type PERSON is abstract;
     type BOOK is abstract;
     type TITLE is abstract;
     type PERSONSET is set of PERSON;
     type BOOKSET is set of BOOK;
   //Class:
     class Library is
       public available : BOOKSET;
       public checkedOut : BOOKSET;
       public requested : BOOKSET;
       public onHold : BOOKSET;

      public function CheckIn (returned : in BOOK) :
Boolean

      assumes True

      guarantees
           (not (returned in checkedOut'))
           and (not (returned in requested'))
           and ( (not (returned in requested)) =>
((returned in available') and (CheckIn(returned) = True))
)
           and ( (returned in requested) => ((returned in
onHold') and (CheckIn(returned) = False)) )
     end class;
   end package;
```

Figure 5.3: Input to test the transform XformSwetha1, XformSwetha2 and XformSwetha3 (before splitting the post condition to form individual post conditions)

First XformArjun5 is applied. This iteratively splits all the post-conditions in the method CheckIn, which have **AND** as the top-level operator into individual post-conditions. The resulting post condition set is modified and holds the individual post conditions as shown in Figure 5.4.

```
1. (returned in requested) => ((returned in onHold') and (CheckIn(returned) = False))
2. not (returned in requested) => (returned in available') and (CheckIn(returned) = True)
3. not (returned in checkedOut')
4. not (returned in requested')
```

Figure 5.4: Input to test the transform XformSwetha1, XformSwetha2 and XformSwetha3 (after splitting the post condition to form individual post conditions)

Assuming that the user intends to conjunct the post condition expressions 3 and 4, the result of the transformation is as shown in Figure 5.5.

```
1. (returned in requested) => ((returned in onHold') and (CheckIn(returned) = False))
2. not (returned in requested) => (returned in available') and (CheckIn(returned) = True)
3. not(returned in checkedOut') and not (returned in requested')
```

Figure 5.5: Output obtained from testing the transform XformSwetha1

The AWESOME AST showing the post condition expressions before and after applying XformSwetha1 is shown in Fig. 5.6 and Fig 5.7. For the rest of the transforms only the AWL outputs will be shown. The AWSOME AST will not be shown unless needed.

wsPostConditions(0):Implication
. wsBinExpOp1:In
. ----------*excess code removed for simplicity*----------
. wsBinExpOp2:And
. ----------*excess code removed for simplicity*----------
wsPostConditions(1):Implication
. wsBinExpOp1:Not
. . wsUnaryExpOp:In
. ----------*excess code removed for simplicity*----------
. wsBinExpOp2:And
. ----------*excess code removed for simplicity*----------
wsPostConditions(2):Not
. wsUnaryExpOp:In
. ----------*excess code removed for simplicity*----------
wsPostConditions(3):Not
. wsUnaryExpOp:In
. ----------*excess code removed for simplicity*----------

Figure 5.6: AWSOME AST showing the post-conditions before XformSwetha1 is applied

wsPostConditions(0):Implication
. wsBinExpOp1:In
. ----------*excess code removed for simplicity*----------
. wsBinExpOp2:And
. ----------*excess code removed for simplicity*----------
wsPostConditions(1):Implication
. wsBinExpOp1:Not
. . wsUnaryExpOp:In
. ----------*excess code removed for simplicity*----------
. wsBinExpOp2:And
. ----------*excess code removed for simplicity*----------
wsPostConditions(2):And
. . . . wsBinExpOp1:Not
. . . . . wsUnaryExpOp:In
. ----------*excess code removed for simplicity*----------
. . . . wsBinExpOp2:Not
. . . . . wsUnaryExpOp:In
. ----------*excess code removed for simplicity*----------

Figure 5.7 AWSOME AST showing the post-conditions after XformSwetha1 is applied

XformSwetha2: This transform conjuncts three or more post-condition expressions selected by the user. On applying the transform to the post conditions 1, 3 and 4 in Fig. 5.4 the resulting post condition set is as shown in Fig. 5.8.

```
1. not (returned in requested) => (returned in available') and (CheckIn(returned) = True)
2. (returned in requested) => ((returned in onHold') and (CheckIn(returned) = False)) and
   (not (returned in checkedOut')) and (not (returned in requested'))
```

Figure 5.8: Output obtained from testing the transform XformSwetha2

The output shows that the selected post condition expressions are **'anded'** and the resulting post condition is added to the post condition set.

XformSwetha3: This transform will iteratively conjunct all the individual post condition expressions in the target method. The resulting post condition expression obtained on applying XformSwetha3 is shown in Fig. 5.10.

```
(returned in requested) => ((returned in onHold') and (CheckIn(returned) = False))
and not (returned in requested) => (returned in available') and (CheckIn(returned)
= True) and (not (returned in checkedOut')) and (not (returned in requested'))
```

Figure 5.10: Output obtained from testing the transform XformSwetha3

TESTING THE SET TRANSFORMS

The set transforms are applied to the post-conditions involving sets. On successful execution of the transform the target post-condition is replaced with a new post-condition that deals with arrays. Fig 5.11 and Fig 5.12 show the input AWL file. The input will remain the same for all the set transforms.

```
package aTestForSetOperations is
type CHAR is abstract;
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type sIndex is range 0 .. 200;
type StudentArray is array[stuIndex] of Student;
type StudentArray2 is array[sIndex] of Student;

class Student is
public sName : STRING;
end class;
```

Figure 5.11: Input AWL file to test Set Transforms

```
class School is
public roster : StudentArray;
public rosterMAX : stuIndex;
public roster2 : StudentArray;
public roster2MAX : stuIndex;

// Check if the student is in roster
public function isStudent(S : in Student) : Boolean
guarantees
(S in roster => check' = True)
and ( ( not  (S in roster ) ) => check' = False )
and isStudent' = check'
is
    check : Boolean;
begin
end;

// Add a student to the roster
public procedure addStudent(S : in Student)
guarantees
S in roster'

// Delete a student from roster
public procedure delStudent(S : in Student)
guarantees
not ( S in roster' )

*//Copy all the students from roster to another say rosterC
public procedure copyStudent(roster : in StudentArray, rosterMAX : in stuIndex,
                             copy : out StudentArray, copyMAX : out stuIndex)
guarantees
 copy' = roster

*// difference of 2 sets
public procedure diff(roster : in StudentArray, rosterMAX : in stuIndex,
                roster2 : in StudentArray, rosterMAX : in stuIndex,
                diff : out StudentArray, diffMAX : out stuIndex)
guarantees
diff' = roster - roster2

*//Union of roster and roster2
public procedure joinStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              join : out StudentArray, joinMAX : out sIndex)
guarantees
join' = roster union roster2

*//Intersection of roster and roster2
public procedure commonStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              common : out StudentArray, commonMAX : out stuIndex)
guarantees
common' = roster intersect roster2

end class;
end package;
```

Figure 5.12: Input AWL file to test Set Transforms.

XformSwetha4: This transform is designed to convert post-conditions of the form *A **in** B'* (where A is an element to be added to the set B) into a post-condition the form: *BMAX' = BMAX + 1 **and** B' [BMAX'] = A* where B is the array and BMAX is the pointer to the last element stored in the array. The transform is applied to the addStudent() procedure's post-condition *S **in** roster'*. The output is shown in Fig 5.13. (Only the target method is shown in the output.)

```
// Add a student to the roster
public procedure addStudent(S : in Student)
guarantees
(rosterMAX' = rosterMAX + 1) and (roster'[rosterMAX'] = S)
```

Figure 5.13: Output after transform XformSwetha4 is applied

XformSwetha5: This transform is designed to convert post-conditions of the form ***not**(A **in** B')* into a post-condition in the form:

***exists**(i : BIndex) ((B[i] = A)*

***and** (B'[i] = B[BMAX])*

***and** (BMAX' = BMAX - 1))*

On applying the transform to the post-condition ***not**(S **in** roster')* the output shown in Fig 5.14 is obtained. (Only the target method is shown in the output.)

```
// Delete a student from roster
public procedure delStudent(S : in Student)
guarantees
exists (i : stuIndex) ((roster[i] = S) and (roster'[i] =
roster[rosterMAX]) and (rosterMAX' = rosterMAX - 1))
```

Figure 5.14: Output after transform XformSwetha5 is applied

XformSwetha6: This transform can be applied to post-conditions that check for the presence of a certain element in a set. The target post-condition must be in the exact form shown:

*(A **in** B => check' = true) **and** (**not** A **in** B) => check' = false **and** funcName' = check'*

where ‘check’ is a local Boolean variable and ‘funcName’ is the name of the current function.

The post-condition after applying the transform is:

***exists**(i : Bindex) (B[i] = A)=> check' = true*

***and not** (**exists**(i : Bindex) (B[i] = A)) => check' = false*

***and** funcName' = check'*

Every expression with a ***in*** operator in the post-condition is converted into an equivalent existential expression. On applying this transform on the input AWL file the output shown in Fig 5.15 is obtained.

```
// Check if the student is in roster
public function isStudent(S : in Student) : Boolean
is
    check : Boolean;
guarantees

exists (i : stuIndex) (roster[i] = S) => check' = True

and not (exists (i : stuIndex) (roster[i] = S)) => check' = False

and (isStudent' = check')
```

Figure 5.15: Output after transform XformSwetha6 is applied

XformSwetha7: This transform is designed to find the post-conditions in the form *A **in** B* or form *A **in** B'* in the current class and converts the **in** to an equivalent existential expression.

*A **in** B* is converted into ***exists**(i : BIndex) (B[i] = A)* and

*A **in** B'* is converted into ***exists**(i : BIndex) (B' [i] = A)*

On applying XformSwetha7 the input AWL file is modified as shown in Fig 5.16.

```
class School is
public roster : StudentArray;
public rosterMAX : stuIndex;
public roster2 : StudentArray;
public roster2MAX : stuIndex;

// Check if the student is in roster
public function isStudent(S : in Student) : Boolean
is
     check : Boolean;
guarantees
exists (i : stuIndex) (roster[i] = S) => check' = True
and (not (exists (i : stuIndex) (roster[i] = S) => check' = False))
and (isStudent' = check')

// Add a student to the roster
public procedure addStudent(S : in Student)
guarantees
exists (i : stuIndex) (roster'[i] = S)

// Delete a student from roster
public procedure delStudent(S : in Student)
guarantees
not exists (i : stuIndex) (roster'[i] = S)

------------------Excess code removed for simplicity----------------------------------

end class;
end package;
```

Figure 5.16: Output after transform XformSwetha7 is applied

Note that this provides the same result as XformSwetha6 for isStudent(), and provides an alternate (less complete) result for addStudent() and delStudent() than did XformSwetha4 and XformSwetha5 respectively. Thus XformSwetha7 is a more generally usable transform.

TESTING THE ARRAY TRANSFORMS

XformSwetha8: This transform is designed to convert specific equality expressions to assignment statement. The following expression *B'[index position] = A* can be converted into *B[index position]:= A* where B is an array, A is the element to be placed in the position indicated by the variable 'index position'. Fig 5.17 shows the input AWL file which include the specific post-condition that can be handled by this transform.

```
package TestForSetOperations is

type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is

    private roster : StudentArray;
    private rosterMAX : stuIndex;


    // Add a student to the roster
    public procedure addElement(S : in Student)
    guarantees
            roster'[rosterMAX] = S

end class;

end package;
```

Figure 5.17: Input AWL file to test XformSwetha8

On applying XformSwetha8 to the post-condition *roster'[rosterMAX] = S* it is changed into an assignment statement and the array roster is unticked. The output is AWL file is shown in Fig 5.18.

```
package TestForSetOperations is

type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
     public sName : STRING;
end class;

class School is
   private roster : StudentArray;
   private rosterMAX : stuIndex;
   public procedure addElement(S : in Student)
      is
      begin
        roster[rosterMAX] := S;
      end;
end class;
```

Figure 5.18: Output after transform XformSwetha8 is applied

XformSwetha9: This transform converts post-conditions of the form

BMAX' = BMAX + 1 ***and*** *B'[BMAX'] = A (*where A is the element to be added to the array B and BMAX indicates the index of the last element stored in the array B).

On executing this transform the following code statements are added to the target method body in the order shown: BMAX**:=** BMAX + 1

B[BMAX] **:=** A

The input AWL file used to test XformSwetha9 and the corresponding output are shown in Fig. 5.19 and Fig 5.20 respectively.

```
package aTestForSetOperations is

type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    public roster : StudentArray;
    public rosterMAX : stuIndex;

    // Add a student to the roster
    public procedure addStudent(S : in Student)
    guarantees
            rosterMAX' = rosterMAX + 1 and roster'[rosterMAX'] = S

end class;
end package;
```

Figure 5.19: Input AWL file to test XformSwetha9

```
package aTestForSetOperations is

type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
  public roster : StudentArray;
  public rosterMAX : stuIndex;
  public procedure addStudent(S : in Student)
    is
    begin
      rosterMAX := rosterMAX + 1;
      roster[rosterMAX] := S;
    end;
end class;

end package;
```

Figure 5.20: Output after transform XformSwetha9 is applied

XformSwetha10: This transform is designed to convert post-conditions in the form *X' = **exists**(i : Bindex)(B[i] = A)* (where B is an array and X is the target subprogram name) to code statements which search for the element A in the array B. The return value of X is assigned to true or false depending on the result of the search. The input AWL file used to test XformSwetha10 and the corresponding output are shown in Fig. 5.21 and Fig 5.22 respectively.

```
package TestXformSwetha10 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

    // Check if the student is in roster
    public function test(S : in Student) : Boolean
    guarantees
    test' = exists(i : stuIndex)(roster[i] = S)
end class;

end package;
```

Figure 5.21: Input AWL file to test XformSwetha10

```
package TestXformSwetha10 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
  private roster : StudentArray;
  private rosterMAX : stuIndex;
  public function test(S : in Student) : Boolean
    is
      i : stuIndex;
    begin
      i := 0;
      while i <= rosterMAX do
        if roster[i] = S then
          test := true;
        else
          i := i + 1;
        end if;
      end do;
      test := false;
    end;
end class;
```

Figure 5.22: Output after transform XformSwetha10 is applied

XformSwetha11: This transform is designed to convert the post-conditions in the form ***exists****(i : Bindex)(B'[i] = A)* (where B is an array) to code statements that increment the BMAX variable by one and insert the element A in the position indicated by BMAX. The input AWL file used to test XformSwetha11 is shown in Fig. 5.23. First the post-condition is transformed into *BMAX' = BMAX + 1* ***and*** *B'[BMAX'] = A.* Next XformSwetha9 is used to change the above post-condition into assignment statements that are added to the method body. The output AWL file is shown in Fig. 5.24.

```
package TestXformSwetha11 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

    public procedure addStudent(S : in Student)
    guarantees
    exists( i : StuIndex ) (roster'[i] = S )
end class;
end package;
```

Figure 5.23: Input AWL file to test XformSwetha11

```
package TestXformSwetha11 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
  private roster : StudentArray;
  private rosterMAX : stuIndex;
  public procedure addStudent(S : in Student)
    is
    begin
      rosterMAX := rosterMAX + 1;
      roster[rosterMAX] := S;
    end;
end class;
```

Figure 5.24: Output after transform XformSwetha11 is applied

XformSwetha12: This transform is designed to convert the post-conditions in the form ***exists*** *(i : Bindex) ((B[i] = A)* ***and*** *exp1* ***and*** *exp2* ***and*** *exp3...)* where B is an array, 'i' is a variable of the type Bindex and whose value ranges from 0 to BMAX. On applying this transform the body of the target method includes a loop in which every element in the array B is compared to the element A. If the element is found then a procedure call is made to a new procedure (which is added to the class operations by this transform). The input AWL file used to test XformSwetha12 and the corresponding output are shown in Fig. 5.25 and Fig 5.26 respectively.

```
    package TestXformSwetha12 is
    type CHAR is abstract; //use ASCII character set.
    type STRING is sequence of CHAR;
    type stuIndex is range 0 .. 100;
    type StudentArray is array[stuIndex] of Student;

    class Student is
        public sName : STRING;
    end class;

    class School is
        private roster : StudentArray;
        private rosterMAX : stuIndex;

    public procedure delStudent(S : in Student)
    guarantees
                exists (i : stuIndex) ((roster[i] = S) and (roster'[i] = roster[rosterMAX]) and
(rosterMAX' = rosterMAX - 1))

    end class;

    end package;
```

Figure 5.25: Input AWL file to test XformSwetha12

On applying XformSwetha12 a new procedure doImplementation() is added to the class operations. The element S is compared to all the elements in the array roster. If a match is found then a procedure call is made to the doImplementation() and the value of i is passed as a formal parameter. The while loop ends when the value of i equals rosterMAX. The constraint in the existential is now treated as doImplementation() post-condition.

```
package TestXformSwetha12 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

public procedure delStudent(S : in Student)
   is
      i : stuIndex;
    begin
      i := 0;
      while i <= rosterMAX do
        if roster[i] = S then
          doImplementation(i);
        else
          i := i + 1;
        end if;
      end do;
    end;

  private procedure doImplementation(i : in stuIndex)
    guarantees roster'[i] = roster[rosterMAX] and rosterMAX' = rosterMAX - 1
end class;
```

Figure 5.26: Output after transform XformSwetha12 is applied

XformSwetha13 and XformSwetha14 are logical transforms and are tested in the next section.

XformSwetha15: This transform changes the post-conditions in the form

forall*(i : D) Q* (where D is the index type of i and Q is an WsExpression). This transform adds a set of code statements (to the target method) in which for each value of 'i' the function doImplementationX () is called. The doImplementationX () is added to the class operations by the XformSwetha15 and it returns a Boolean value. Depending on the value returned by the doImplementationX () the loop is terminated or is continued until i reaches the maximum value possible. The input AWL file used to test XformSwetha15 is shown in Fig. 5.27. The correctness of XformSwetha15 is dependant on XformSwetha16. The user needs to call XformSwetha16 after executinfg XformSwetha15.

```
package TestXformSwetha15 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

public procedure addStudent(S : in Student)
guarantees
            forall( i : StuIndex ) (not (roster'[i] = S ))
end class;

end package;
```

Figure 5.27: Input AWL file to test XformSwetha15

On applying XfromSwetha15 to the post-condition

forall*(i : StuIndex)(not(roster'[i] = S)* a new function doImplementation() is added to the School class. The logical variable i and the target method's formal parameter S become the formal parameters for the function doImplementation(). The expression ***(not****(roster'[i] = S)* is treated as its post-condition. Fig 5.28 shows the output AWL file obtained.

```
package TestXformSwetha15 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

    public procedure addStudent(S : in Student)
    is
      i : stuIndex;
      fRet : boolean;
    begin
      i := 0;
      fRet := false;
      while i <= rosterMAX do
        fRet := doImplementation(i, S);
        if fRet = false then
          i := i + 1;
        else
        end if;
      end do;
    end;

  private function doImplementation(i : in stuIndex, S : in Student)  : boolean
    guarantees not (roster'[i] = S)

end class;

end package;
```

Figure 5.28: Output after transform XformSwetha15 is applied

XformSwetha16: This transform is designed to convert the post-conditions in the form ***not****(B'[i] = A).* XformSwetha16 adds code statements to the target method body. These code statements compare the element at i^{th} position in the array B with the element A. If they match then that element is replaced by the last element stored in the array. The input AWL file used to test XformSwetha12 and the corresponding output are shown in Fig. 5.29 and Fig 5.30 respectively.

```
package TestXformSwetha16 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

     private function Test(i : in stuIndex, S : in Student)  : boolean
     guarantees
            not (roster'[i] = S)

    end class;
end package;
```

Figure 5.29: Input AWL file to test XformSwetha16

On applying the transform to the post-condition ***not*** *(roster'[i] = S)* if the element in the i^{th} position in the array roster is S then is replaced by the last element stored in the array. The value of rosterMAX is decremented by 1 and the function returns true. If the element is not S then the function returns false.

```
package TestXformSwetha16 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
  private roster : StudentArray;
  private rosterMAX : stuIndex;
  private function Test(i : in stuIndex, S : in Student)  : boolean
    is
    begin
      if roster[i] = S then
        roster[i] := roster[rosterMAX];
        rosterMAX := rosterMAX - 1;
        Test := true;
      else
        Test := false;
      end if;
    end;
end class;
end package;
```

Figure 5.30: Output after transform XformSwetha16 is applied

XformSwetha17 is a logical transform and is tested in the next section.

XformSwetha18: This transform changes the post-conditions in the form

$X' = A$ (where X and A are arrays) to code statements. After applying this transform code statements that copy every element in the array A to the corresponding position in array X are added to the target method body. The input AWL file used to test XformSwetha18 is shown in Fig. 5.31

```
package Test is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type sIndex is range 0 .. 200;
type StudentArray is array[stuIndex] of Student;
type StudentArray2 is array[sIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;
    public roster2 : StudentArray;
    public roster2MAX : stuIndex;

//Copy all the students from roster to another say rosterC
public procedure copyStudent(roster : in StudentArray,  rosterMAX : in stuIndex,
                             copy : out StudentArray,    copyMAX : out stuIndex)
    guarantees
        copy' = roster

// difference of 2 sets
public procedure diff(roster : in StudentArray, rosterMAX : in stuIndex,
                      roster2 : in StudentArray, rosterMAX : in stuIndex,
                      diff : out StudentArray, diffMAX : out stuIndex)
    guarantees
        diff' = roster - roster2

//Union of roster and roster2
public procedure joinStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              join : out StudentArray, joinMAX : out sIndex)
    guarantees
        join' = roster union roster2


//Intersection of roster and roster2
public procedure commonStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              common : out StudentArray, commonMAX : out stuIndex)
    guarantees
        common' = roster intersect roster2

    end class;
end package;
```

Figure 5.31: Input AWL file to test XformSwetha18, XformSwetha19, XformSwetha20 and XformSwetha21.

Fig 5.32 shows the output AWL file obtained on applying XfromSwetha18 to the copyStudent() method's post-condition. The post-condition is removed and the set of statements that copies every element in the array roster into the out parameter copy is added to the method body.

```
package Test is
----------------------------excess code removed for simplicity----------------------------

class School is
  private roster : StudentArray;
  private rosterMAX : stuIndex;

public procedure copyStudent(roster : in StudentArray, rosterMAX : in stuIndex,
                             copy : out StudentArray, copyMAX : out stuIndex)
    is
      i : stuIndex;
    begin
      i := 0;
      while i <= rosterMAX do
        copy[i] := roster[i];
        copyMAX := i;
        i := i + 1;
      end do;
    end;
----------------------------excess code removed for simplicity----------------------------


end class;
end package;
```

Figure 5.32: Output after transform XformSwetha18 is applied

XformSwetha19: This transform is designed to convert the post-condition that finds the difference of two sets. Fig 5.31 shows the input AWL file used to test this transform. On applying this transform to the post-condition *diff' = roster - roster2* produces the output shown in Fig. 5.33.

```
package Test is
----------------------------excess code removed for simplicity----------------------------
class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;
    public roster2 : StudentArray;
    public roster2MAX : stuIndex;

----------------------------excess code removed for simplicity----------------------------

public procedure diff(roster : in StudentArray, rosterMAX : in stuIndex,
                      roster2 : in StudentArray, rosterMAX : in stuIndex,
                       diff : out StudentArray, diffMAX : out stuIndex)
    is
      i : stuIndex;
      j : stuIndex;
      k : stuIndex;
      found : boolean;
    begin
      i := 0;
      k := 0;
      while i <= rosterMAX do
        j := 0;
        found := false;
        while j <= roster2MAX do
          if roster[i] = roster2[j] then
            found := true;
          else
          end if;
          j := j + 1;
        end do;
        if found = false then
          diff[k] := roster[i];
          k := k + 1;
        else
        end if;
        diffMAX := i;
        i := i + 1;
      end do;
    end;

----------------------------excess code removed for simplicity----------------------------
end class;
end package;
```

Figure 5.33: Output after transform XformSwetha19 is applied

XformSwetha20: This transform is designed to convert the post-condition that finds the union of two sets into code statements. The post-condition needs to be in the form $X' = A$ ***union*** B. Fig 5.31 shows the input AWL file used to test this transform. On applying

XformSwetha20 to the post-condition *join' = roster* ***union*** *roster2* the output shown in Fig. 5.34 is obtained.

```
    package Test is
    ----------------------------excess code removed for simplicity----------------------------
    class School is
        private roster : StudentArray;
        private rosterMAX : stuIndex;
        public roster2 : StudentArray;
        public roster2MAX : stuIndex;
    ---------------------------excess code removed for simplicity---------------------------
    public procedure joinStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                                  roster2 : in StudentArray, rosterMAX : in stuIndex,
                                  join : out StudentArray, joinMAX : out sIndex)
        is
          i : stuIndex;
          j : stuIndex;
          k : sIndex;
          found : boolean;
        begin
          i := 0;
          j := 0;
          k := 0;
          while i <= rosterMAX do
            join[k] := roster[i];
            joinMAX := k;
            i := i + 1;
            k := k + 1;
          end do;
          while j <= roster2MAX do
            found := false;
            i := 0;
            while i <= rosterMAX do
              if roster2[j] = roster[i] then
                found := true;
              else
              end if;
              i := i + 1;
            end do;
            if found = false then
              join[k] := roster2[j];
              joinMAX := k;
              k := k + 1;
            else
            end if;
            j := j + 1;
          end do;
        end;
----------------------------excess code removed for simplicity----------------------------

    end class;
    end package;
```

Figure 5.34: Output after transform XformSwetha20 is applied

XformSwetha21: This transform is designed to convert the post-condition that finds the intersection of two sets into code statements. The post-condition needs to be in the form *X' = A **intersect** B.* Fig 5.31 shows the input AWL file used to test this transform. On applying XformSwetha21 to the post-condition *common' = roster **intersect** roster2* the output shown in Fig. 5.35 is obtained.

```
package Test is
----------------------------excess code removed for simplicity----------------------------
class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;
    public roster2 : StudentArray;
    public roster2MAX : stuIndex;
----------------------------excess code removed for simplicity----------------------------
public procedure commonStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                                roster2 : in StudentArray, rosterMAX : in stuIndex,
                                common : out StudentArray, commonMAX : out sIndex)
 is
      i : stuIndex;
      j : stuIndex;
      k : stuIndex;
      found : boolean;
    begin
      i := 0;
      k := 0;
      while i <= rosterMAX do
        j := 0;
        found := false;
        while j <= roster2MAX do
          if roster[i] = roster2[j] then
            found := true;
          else
          end if;
          j := j + 1;
        end do;
        if found = true then
          common[k] := roster[i];
          commonMAX := k;
          k := k + 1;
        else
        end if;
        i := i + 1;
      end do;
    end;

end class;
end package;
```

Figure 5.35: Output after transform XformSwetha21 is applied

TESTING THE LOGICAL TRANSFORMS

XformSwetha13: This transform is designed to convert the post-condition in the form ***not(exists*** *(i : D) Q)* to ***forall****(i : D)* ***not*** *Q.* The input AWL file used to test XformSwetha13 is shown in Fig. 5.36 and the corresponding output is shown in Fig 5.37.

```
package TestXformSwetha13and14 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

    public procedure testSwetha13(S : in Student)
    guarantees
            not exists (i : stuIndex) (roster'[i] = S)

    public procedure testSwetha14(S : in Student)
    guarantees
            forall (i : stuIndex) (not (roster'[i] = S))

end class;
end package;
```

Figure 5.36: Input AWL file to test XformSwetha13 and XformSwetha14

```
package TestXformSwetha13and14 is
----------------------------excess code removed for simplicity-------------------------
class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

public procedure testSwetha13(S : in Student)
guarantees
            forall (i : stuIndex) (not (roster'[i] = S))
 -------------------------excess code removed for simplicity----------------------------
end class;
end package;
```

Figure 5.37: Output after transform XformSwetha13 is applied

XformSwetha14: This transform is designed to convert the post-condition in the form ***forall****(i : D)* ***not*** *Q* to ***not****(****exists*** *(i : D) Q).* XformSwetha14 is the inverse of XformSwetha13. The input AWL file used to test XformSwetha14 is shown in Fig. 5.36 and the corresponding output is shown in Fig 5.38.

```
package TestXformSwetha13and14 is
----------------------------excess code removed for simplicity-------------------------
class School is
     private roster : StudentArray;
     private rosterMAX : stuIndex;

-------------------------excess code removed for simplicity-----------------------------

public procedure testSwetha14(S : in Student)
guarantees
              not exists (i : stuIndex) (roster'[i] = S)
  end class;
end package;
```

Figure 5.38: Output after transform XformSwetha14 is applied

XformSwetha17: This transform is designed to convert the post-condition in the form

P =>(X' = true) ***and not*** *P => (X' = false)* to *X' = P*.

Here X is the name of the subprogram for which this is a post-condition or a local variable. The input AWL file used to test XformSwetha17 is shown in Fig. 5.39. The output obtained on applying the transform is shown in Fig. 5.40.

```
package TestXfromSwetha17 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

    // Check if the student is in roster
    public function isStudent(S : in Student) : boolean
    guarantees
    (S in roster => isStudent' = true)
    and not ( S in roster => isStudent' = false )

end class;
end package;
```

Figure 5.39: Input AWL file to test XformSwetha17

```
package TestXfromSwetha17 is
type CHAR is abstract; //use ASCII character set.
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type StudentArray is array[stuIndex] of Student;

class Student is
    public sName : STRING;
end class;

class School is
    private roster : StudentArray;
    private rosterMAX : stuIndex;

    // Check if the student is in roster
    public function isStudent(S : in Student) : boolean
    guarantees
    isStudent' = S in roster

end class;
end package;
```

Figure 5.40: Output after transform XformSwetha17 is applied

CODE GENERATION

The formal specification model is parsed into an AST to which a series of transforms are performed to change it to a design AST. This design model is given to the code generator which does the necessary work to generate the executable Java code. The focus of this thesis is to convert set operators in post-conditions into statements. On completing that task we went ahead and tried to generate the Java code. The output design AST obtained by applying the transforms developed in this thesis is given to the code generator and the result is a file with Java code. Input AWL file is shown in Fig 5.41. After applying the transforms the **ToolToAWL** tool is used to write the AST back to an AWL file (shown in Fig. 5.42a, Fig. 5.42b, Fig. 5.42c, Fig. 5.42d, Fig. 5.42e) reflecting the changes made to the input AWL file. Note that the declarations are yet to be transformed using the transforms developed by Manubolu [6]. Fig. 5.43a, Fig. 5.43b and Fig. 5.435c show the resulting Java code. Note that only the School class is shown for the Java code.

```
package aTestForSetOperations is
type CHAR is abstract;
type STRING is sequence of CHAR;
type stuIndex is range 0 .. 100;
type sIndex is range 0 .. 200;
type StudentArray is array[stuIndex] of Student;
type StudentArray2 is array[sIndex] of Student;

class Student is
public sName : STRING;
end class;
```

Figure 5.41a: Input AWL file.

```
class School is
public roster : StudentArray;
public rosterMAX : stuIndex;
public roster2 : StudentArray;
public roster2MAX : stuIndex;

// Check if the student is in roster
public function isStudent(S : in Student) : Boolean
guarantees
(S in roster => isStudent' = True)
and ( ( not  (S in roster ) ) => isStudent' = False )

// Add a student to the roster
public procedure addStudent(S : in Student)
guarantees
S in roster'

// Delete a student from roster
public procedure delStudent(S : in Student)
guarantees
not ( S in roster' )

//Copy all the students from roster
public procedure copyStudent(roster : in StudentArray, rosterMAX : in stuIndex,
                             copy : out StudentArray, copyMAX : out stuIndex)
guarantees
 copy' = roster

// difference of 2 sets
public procedure diff(roster : in StudentArray, rosterMAX : in stuIndex,
                roster2 : in StudentArray, rosterMAX : in stuIndex,
                diff : out StudentArray, diffMAX : out stuIndex)
guarantees
diff' = roster - roster2

//Union of roster and roster2
public procedure joinStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              join : out StudentArray, joinMAX : out sIndex)
guarantees
join' = roster union roster2

//Intersection of roster and roster2
public procedure commonStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              common : out StudentArray, commonMAX : out stuIndex)
guarantees
common' = roster intersect roster2

end class;
end package;
```

Figure 5.41b: Input AWL file.

```
package aTestForSetOperations is
   type CHAR is abstract;
   type STRING is sequence of CHAR;
   type stuIndex is range 0 .. 100;
   type sIndex is range 0 .. 200;
   type StudentArray is array [stuIndex] of Student;
   type StudentArray2 is array [sIndex] of Student;

class Student is
     public sName : STRING;
   end class;

class School is
      public roster : StudentArray;
      public rosterMAX : stuIndex;
      public roster2 : StudentArray;
      public roster2MAX : stuIndex;

  // Check if the student is in roster
   public function isStudent(S : in Student)  : Boolean
        is
          i : stuIndex;
        begin
          i := 0;
          while i <= rosterMAX do
            if roster[i] = S then
              isStudent := true;
            else
              i := i + 1;
            end if;
          end do;
          isStudent := false;
        end;

  // Add a student to the roster
  public procedure addStudent(S : in Student)
        is
        begin
          rosterMAX := rosterMAX + 1;
          roster[rosterMAX] := S;
        end;
```

Figure 5.42a: Output AWL file after applying the transforms.

(shows the isStudent() and the addStudent() methods)

The output shown in Fig 5.44a is obtained as a result of using the transforms XformSwetha7, XfromSwetha17, XformSwetha10 for changing isStudent() and XformSwetha9 for changing addStudent().

```
// Delete a student from roster
 public procedure delStudent(S : in Student)
        is
          i : stuIndex;
          fRet : boolean;
        begin
          i := 0;
          fRet := false;
          while i <= rosterMAX do
            fRet := doImplementation(i, S);
            if fRet = false then
              i := i + 1;
            else
            end if;
          end do;
        end;
   // New method added to the class by one of the transforms
   private function doImplementation(i : in stuIndex, S : in Student)  : boolean
        is
        begin
          if roster[i] = S then
            roster[i] := roster[rosterMAX];
            rosterMAX := rosterMAX - 1;
            doImplementation := true;
          else
            doImplementation := false;
          end if;
        end;

 //Copy all the students from roster
 public procedure copyStudent(roster : in StudentArray, rosterMAX : in stuIndex,
                              copy : out StudentArray, copyMAX : out stuIndex)
        is
          i : stuIndex;
        begin
          i := 0;
          while i <= rosterMAX do
            copy[i] := roster[i];
            copyMAX := i;
            i := i + 1;
          end do;
        end;
```

Figure 5.42b: Output AWL file after applying the transforms.

(Shows the delStudent(), doImplementation() and the copyStudent() methods)

The output shown in Fig 5.42b is the result of using the transforms XformSwetha13, XfromSwetha15, XformSwetha16 for changing delStudent() and XformSwetha18 for changing copyStudent().

```
// difference of 2 sets
public procedure diff(roster : in StudentArray, rosterMAX : in stuIndex,
                      roster2 : in StudentArray, rosterMAX : in stuIndex,
                      diff : out StudentArray, diffMAX : out stuIndex)
       is
         i : stuIndex;
         j : stuIndex;
         k : stuIndex;
         found : boolean;
       begin
         i := 0;
         k := 0;
         while i <= rosterMAX do
           j := 0;
           found := false;
           while j <= roster2MAX do
             if roster[i] = roster2[j] then
               found := true;
             else
             end if;
             j := j + 1;
           end do;
           if found = false then
             diff[k] := roster[i];
             k := k + 1;
           else
           end if;
           diffMAX := i;
           i := i + 1;
         end do;
       end;
```

Figure 5.42c: Output AWL file after applying the transforms.

(Shows the diff() method)

The output shown in Fig 5.42c is the result of applying the transform XformSwetha19 to the diff() post-condition.

```
//Union of roster and roster2
public procedure joinStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                              roster2 : in StudentArray, rosterMAX : in stuIndex,
                              join : out StudentArray, joinMAX : out sIndex)
      is
        i : stuIndex;
        j : stuIndex;
        k : sIndex;
        found : boolean;
      begin
        i := 0;
        j := 0;
        k := 0;
        while i <= rosterMAX do
          join[k] := roster[i];
          joinMAX := k;
          i := i + 1;
          k := k + 1;
        end do;
        while j <= roster2MAX do
          found := false;
          i := 0;
          while i <= rosterMAX do
            if roster2[j] = roster[i] then
              found := true;
            else
            end if;
            i := i + 1;
          end do;
          if found = false then
            join[k] := roster2[j];
            joinMAX := k;
            k := k + 1;
          else
          end if;
          j := j + 1;
        end do;
      end;
```

Figure 5.42d: Output AWL file after applying the transforms.

(Shows the joinStudent() method)

The output shown in Fig 5.42d is the result of applying the transform XformSwetha20 to the joinStudent() post-condition.

```
 //Intersection of roster and roster2
public procedure commonStudents(roster : in StudentArray, rosterMAX : in stuIndex,
                                roster2 : in StudentArray, rosterMAX : in stuIndex,
                               common : out StudentArray, commonMAX : out stuIndex)
        is
          i : stuIndex;
          j : stuIndex;
          k : stuIndex;
          found : boolean;
        begin
          i := 0;
          k := 0;
          while i <= rosterMAX do
            j := 0;
            found := false;
            while j <= roster2MAX do
              if roster[i] = roster2[j] then
                found := true;
              else
              end if;
              j := j + 1;
            end do;
            if found = true then
              common[k] := roster[i];
              commonMAX := k;
              k := k + 1;
            else
            end if;
            i := i + 1;
          end do;
        end;
```

Figure 5.42e: Output AWL file after applying the transforms.

(Shows the commonStudent() method)

The output shown in Fig 5.42c is obtained as a result of applying the transform XformSwetha21 to the commonStudent() post-condition.

```
package aTestForSetOperations;
  /**
  * File    School.java
  */
  public class School {
    public    StudentArray roster;
    public    stuIndex rosterMAX;
    public    StudentArray roster2;
    public    stuIndex roster2MAX;

    public Boolean isStudent(Student S) {
      stuIndex i;
      i=0;
      while (i <= rosterMAX) {
        if (roster[i] == S) {
          return true;
        } else {
          i=i+1;
        }
      }
      return false;
    }

    public void addStudent(Student S) {
      rosterMAX=rosterMAX+1;
      roster[rosterMAX] = S;
    }

    public void delStudent(Student S) {
      stuIndex i;
      boolean fRet;
      i=0;
      fRet=false;
      while (i <= rosterMAX) {
        fRet=doImplementation(i, S) ;
        if (fRet == false) {
          i=i+1;
        }
       }
     }

  protected boolean doImplementation(stuIndex i, Student S) {
      if (roster[i] == S) {
        roster[i] = roster[rosterMAX];
        rosterMAX=rosterMAX-1;
        return true;
      } else {
        return false;
      }

    }
```

Figure 5.43a: Java file generated by the Code Generator.

(Shows the isStudent(), addStudent() delStudent() and the doImplementation() methods)

Fig 5.43a shows the resulting Java code for the addStudent() and delStudent().

```
public void copyStudent(StudentArray roster, stuIndex rosterMAX,
                        StudentArray copy, stuIndex copyMAX) {
    stuIndex i;
    i=0;

    while (i <= rosterMAX) {
      copy[i] = roster[i];
      copyMAX=i;
      i=i+1;
    }

  }

  public void diff(StudentArray roster, stuIndex rosterMAX, StudentArray roster2,
                              stuIndex rosterMAX, StudentArray diff, stuIndex
                              diffMAX) {
    stuIndex i;
    stuIndex j;
    stuIndex k;
    boolean found;
    i=0;
    k=0;

    while (i <= rosterMAX) {
      j=0;
      found=false;

      while (j <= roster2MAX) {

        if (roster[i] == roster2[j]) {
          found=true;
        }

        j=j+1;
      }

      if (found == false) {
        diff[k] = roster[i];
        k=k+1;
      }

      diffMAX=i;
      i=i+1;
    }

  }
```

Figure 5.43b: Java file generated by the Code Generator.

Fig 5.43b shows the resulting Java code for the copyStudent and diff() methods.

```
public void joinStudents(StudentArray roster, stuIndex rosterMAX, StudentArray roster2,
                         stuIndex rosterMAX, StudentArray join, sIndex joinMAX) {
    stuIndex i;
    stuIndex j;
    sIndex k;
    boolean found;
    i=0;
    j=0;
    k=0;
    while (i <= rosterMAX) {
      join[k] = roster[i];
      joinMAX=k;
      i=i+1;
      k=k+1;
    }
    while (j <= roster2MAX) {
      found=false;
      i=0;
      while (i <= rosterMAX) {
        if (roster2[j] == roster[i]) {
          found=true;
        }
        i=i+1;
      }
      if (found == false) {
        join[k] = roster2[j];
        joinMAX=k;
        k=k+1;
      }
      j=j+1;
    }
  }
public void commonStudents(StudentArray roster, stuIndex rosterMAX, StudentArray
      roster2, stuIndex rosterMAX, StudentArray common, stuIndex commonMAX) {
    stuIndex i;
    stuIndex j;
    stuIndex k;
    boolean found;
    i=0;
    k=0;
    while (i <= rosterMAX) {
      j=0;
      found=false;
      while (j <= roster2MAX) {
        if (roster[i] == roster2[j]) {
          found=true;
        }
        j=j+1;
      }
      if (found == true) {
        common[k] = roster[i];
        commonMAX=k;
        k=k+1;
      }
      i=i+1;
    }
  }
```

Figure 5.43c: Java file generated by the Code Generator.

Fig 5.43c shows the resulting Java code for the joinStudent and commonStudent() methods.

USING THE TRANSFORMS

To change some of the post-conditions identified in the requirement analysis chapter into statements we need to use more than one transform developed in this thesis. This section presents the transform combinations for the post-conditions identified in Fig 3.2. Some of the transforms in this thesis are developed to handle individual post-conditions. Later a more general approach was designed to handle them, the result of which is alternate ways to transform the same post-condition. Table 5.1 shows the possible transform sequences (indicated by bullets). It is assumed that the post-conditions are decomposed using XformArjun5 [7] wherever necessary.

Table 5.1: Possible Transform Sequences

POST-CONDITION	POSSIBLE TRANSFORM SEQUENCES
isStudent()	• XformSwetha1, XformSwetha17, XfromSwetha7, XformSwetha10. • XfromSwetha6, XfromSwetha1, XformSwetha17, XfromSwetha10.
addStudent()	• XformSwetha4, XfromSwetha9. • XfromSwetha7, XfromSwetha11.
delStudent()	• XfromSwetha5, XfromSwetha12. • XfromSwetha7, XfromSwetha13, XfromSwetha15, XfromSwetha16.

The post-condition for copyStudent(), diff(), joinStudent() and commonStudent() can be transformed using XfromSwetha18, XfromSwetha19, XfromSwetha20 and XfromSwetha21 respectively.

SUMMARY

This chapter describes the testing environment and the testing approach used. Appropriate test cases are developed and all the transforms are tested. The results obtained show that the transforms meet the desired design goals.

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

The purpose of this thesis is to design transforms that can change a method's post-condition into statements. An object oriented approach is followed in this thesis. The transforms that are needed are first identified and then appropriately designed. Implementation followed the design and the result is a set of working transforms that can be applied to post-conditions to change them into statements. The transforms that are considered simple to implement were designed first and then the more complex ones were designed.

The first two chapters introduce the reader to AWSOME and AWL and provide the background information necessary to understand the rest of the document. The analysis chapter discusses the post-conditions, the expressions, and the Boolean operators in AWL. The focus of this thesis is to change the post-conditions that have set operators into statements. The transforms needed for this thesis were identified in the analysis chapter. The design chapter outlines the transforms that were identified in the analysis chapter. The functionality of all the transforms is discussed in detail in the design chapter. Following this design all the transforms were implemented and carefully tested. The implementation and testing chapter acquaints the reader with the testing environment and the approach used to test the transforms. Appropriate test cases were written in AWL for testing the transforms. These test cases, along with the output obtained on applying the transforms, are presented in the testing chapter. Table 6.1 lists the transforms developed and their functionality.

Table 6.1: Summary of transforms.

TRANSFORM	DESCRIPTION
XformSwetha1	This transform can be used to conjunct any two post-conditions selected by the user. It is assumed that the selected post-conditions are related according to the user.
XformSwetha2	This transform can be used to conjunct more than two post-conditions selected by the user.
XformSwetha3	This transform can be used to conjunct all the post-conditions in the target method.
XformSwetha4	This transform converts post-conditions in the form *A **in** B'* (where A is an element to be added to the set B) into *BMAX' = BMAX + 1 and B' [BMAX'] = A* where B is the array and BMAX is the pointer to the last element stored in the array.
XformSwetha5	This transform converts post-conditions in the form ***not**(A **in** B')* (where A is an element to be deleted from the set B) into ***exists**(i : BIndex) ((B[i] = A)* ***and** (B'[i] = B[BMAX])* ***and** (BMAX' = BMAX - 1))* where BIndex is the index type of the array B, 'i' is an logical variable of type BIndex.

TRANSFORM	DESCRIPTION
XformSwetha6	This transform converts post-conditions in the form *(A **in** B => check' = true)* ***and** (**not** A **in** B) => check' = false* ***and** funcName' = check'* into ***exists**(i : Bindex) (B[i] = A)=> check' = true* ***and not** (**exists**(i : Bindex) (B[i] = A)) => check' = false* ***and** funcName' = check'* where 'check' is a local Boolean variable, 'funcName' is the name of the current function, BIndex is the index type of the array B and 'i' is an logical variable of type BIndex.
XformSwetha7	This transform changes all the post-conditions in the form *A **in** B* or *A **in** B'* in the current class. *A **in** B* is converted into ***exists**(i : BIndex) (B[i] = A)* and *A **in** B'* is converted into ***exists**(i : BIndex) (B' [i] = A)*
XformSwetha8	This transform changes specific equality expressions to an assignment statement. *B'[index position] = A* is converted into *B[index position]:= A* where B is an array, A is the element to be placed in the position indicated by the variable 'index position'.
XformSwetha9	This transform changes post-conditions in the form *BMAX' = BMAX + 1 **and** B'[BMAX'] = A* into statements. On successful execution of this transform the statements : *BMAX:= BMAX + 1* *B[BMAX] := A* are added to method body.
XformSwetha10	This transform is designed to convert post-conditions in the form *X' = **exists**(i : Bindex)(B[i] = A)* into statements which search for the element A in the array B and X is the target subprogram name. The value of X is assigned to true or false depending on the result of the search.

TRANSFORM	DESCRIPTION
XformSwetha11	This transform changes post-conditions in the form ***exists****(i : Bindex)(B'[i] = A)* into statements. On successful execution of this transform the statements : *BMAX:= BMAX + 1* *B[BMAX] := A* are added to method body.
XformSwetha12	This transform changes the post-conditions in the form ***exists*** *(i : Bindex) ((B[i] = A)* ***and*** *exp1* ***and*** *exp2* ***and*** *exp3...)* where B is an array, 'i' is a variable of the type Bindex and whose value ranges from 0 to BMAX. On applying this transform the body of the target method includes a loop in which every element in the array B is compared to the element A. If the element is found then a procedure call is made to a new procedure doImplementationX () (which is added to the class operations by this transform).
XformSwetha13	This transform is designed to convert the post-condition in the form ***not(exists*** *(i : D) Q)* to ***forall****(i : D)* ***not*** *Q*.
XformSwetha14	This transform does the exact opposite of what XformSwetha13. It converts the post-condition in the form ***forall****(i : D)* ***not*** *Q* to ***not(exists*** *(i : D) Q).*
XformSwetha15	This transform changes the post-conditions in the form ***forall****(i : D) Q* (where D is the index type of i and Q is an WsExpression). This transform adds a set of code statements (to the target method) in which for each value of 'i' the function doImplementationX () is called. The doImplementationX () with a post-condition Q is added to the class operations.
XformSwetha16	This transform is designed to convert the post-conditions in the form ***not****(B'[i] = A).* XformSwetha16 adds code statements to the target method body. These code statements compare the element at ith position in the array B with the element A. If they match then that element is replaced by the last element stored in the array.

TRANSFORM	DESCRIPTION
XformSwetha17	This transform changes the post-condition in the form *P =>(X' = true)* ***and not*** *P => (X' = false)* to *X' = P* (where X is the name of the subprogram for which this is a post-condition or a local variable, P is a WsExpression).
XformSwetha18	This transform is can be used to change the post-conditions in the form *X' = A* (where X and A are arrays). After applying this transform code statements that copy every element in the array A to the corresponding position in array X are added to the target method body.
XformSwetha19	This transform changes the post-conditions that find the difference of two sets into statements.
XformSwetha20	This transform changes the post-conditions that find the union of two sets into statements.
XformSwetha21	This transform changes the post-conditions that find the intersection of two sets into statements.

FUTURE WORK

For this thesis, only action post-conditions are transformed. Constraints are assumed to be converted to derived pre-conditions. Another shortcoming is that when the user selects individual post-conditions to conjunct using the Conjunction transforms there is no way to identify the dependency (if any) between the selections. It is assumed that the selected post-conditions are related according to the user.

Transforms are developed to eliminate most of the set operators in the post-conditions. Set operators like subset, subseteq are considered as constraints and hence

transforms to convert post-conditions with the set operators subset, subseteq and container formers were not developed. Set operators in pre-conditions have not been touched. More work has to be done to transform them accordingly. The transforms designed in the section Set Transforms in Chapter 4 convert the post-condition with sets into post-conditions with arrays. This is done to remain consistent with the previous works. The post-conditions that are transformed in this thesis may be handled using a different approach. Instead of using transforms to convert the post-conditions with set into an alternative form, we can leave them untouched. Later transforms can be developed to convert these post-conditions into statements that use the Set class in Java to obtain the executable code.

The work done in this thesis combined with the past and future works can perfect the AWSOME system and provide us with a transformation system that can successfully transform the formal specification model into executable code.

REFERENCES

[1] T. C. Hartrum and R. P. Graham, Jr., "The AFIT Wide Spectrum Object Modeling Environment: an AWSOME Beginning," *Proceedings of the IEEE 2000 National Aerospace & Electronics Conference (NAECON 2000)*, Dayton, OH, October 2000.

[2] R. P. Graham, Jr. and T. C. Hartrum, *AWSOME Wide-Spectrum Language*(AWL), Air Force Institute of Technology, Language Reference Manual, Edition 1.0, July 14th 2003, unpublished.

[3] T. C. Hartrum, *AWSOME Transforms*, Wright State University, Reference Manual, January 17th 2001, unpublished.

[4] T. C. Hartrum, *AWSOME Tool Box*, Wright State University, Reference Manual, Edition 1.0, January 18th 2005, unpublished.

[5] R. Balzer, T. E. Cheathan, Jr., and C. Green, “Software Technology in the 1990’s: Using a New Paradigm”, *IEEE Computer*, November, 1983.

[6] Manubolu, Pratap, *Transformation of Formally Defined User Data Types Into a Target Language*, Masters Thesis, Wright State University, Dayton, OH, 2006.

[7] Singam, Nagarjuna, *Automated Code Generation From a Formally Specified Post-Condition*, Masters Thesis, Wright State University, Dayton, OH, 2005.

[8] Bhooma Raghunathan, *Automated code synthesis from a formal state machine model.* Wright State University, Dayton OH, Masters Thesis, 2004.

[9] Preeti Subhedar, *Automated design transforms for the object-oriented structural model*. Wright State University, Dayton OH, Masters Thesis, 2005.

[10] Sarvepalli, Venkata, *Automated Design Transforms From a Formally Specified Class Definition*, Masters Thesis, Wright State University, Dayton, OH, 2005

[11] J.B Wordsworth, *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*, IBM United Kingdom Ltd., Addison Wesley 1994.

[12] James Rumbaugh, et al., *Object Oriented Modeling and Design*, Prentice Hall, Inc., 1991.

[13] B. Bruegge, and A.H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterens, and Java*, 2nd ed., Upper Saddle River: Prentice hall, 2004.

[14] Erich Gamma et al, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[15] Wikipedia : Online encyclopedia (date: 04/16/2006)
http://en.wikipedia.org/wiki/Method_%28computer_science%29

Appendix

Java Code

The following section includes just the most important methods in the Transforms and leaves out the full source code. For each of the transforms, the *applicable* and the *execute* methods are outlined in this section.

Transform 1: XformSwetha1.java

```
/*****************************************************************
* Source file: XformSwetha1.java
* Purpose: This transform conjuncts any two post-conditions selected
* by the user.
*****************************************************************/


/*****************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions are valid.
* Parameters: target is the target method object
*             params refers to a Vector holding the indices of the
*             selected post-conditions wrapped in a string.
*****************************************************************/
public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIS = (WsClasses.WsMethod) target;
      WsSubprogram subprgIS = methodIS.getWsMethodSubprogram();
      Vector postconIS = subprgIS.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      for(int i = 0; i < params1.size(); i++)
      {
            String pc1 = (String) params1.get(i);
            int index = Integer.parseInt(pc1);
            WsExpression exp;
            // Checking to see if it is a valid index.
            if(index <= postconIS.size() && index >=0)
                  exp = (WsExpression) postconIS.get(index);
            else
                  return false;
      }

      return true;
}



/*****************************************************************
* This transform conjuncts the two post conditions selected in the
* target method into one post condition and adds it to the
* post condition set. The individual post conditions are removed
```

```
*
* Parameters: params refers to a Vector holding the indices of the
*                  selected post-conditions wrapped in a string.
*********************************************************************/
public boolean execute(Object params)
{
      WsSubprogram subprog = targetmethod.getWsMethodSubprogram();
      Vector postconIS = subprog.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      String pc1 = (String) params1.get(0);
      int index1 = Integer.parseInt(pc1);
      WsExpression exp1 = (WsExpression) postconIS.get(index1);
      String pc2 = (String) params1.get(1);
      int index2 = Integer.parseInt(pc2);
      WsExpression exp2 = (WsExpression) postconIS.get(index2);
      WsAnd expFinal = new WsAnd(exp1, exp2);
      postconIS.add(expFinal);
      //remove the individual post-conditions.
      postconIS.remove(index1);
      postconIS.remove(index2 - 1);

      return true;
}
```

Transform 2: XformSwetha2.java

```
/*********************************************************************
* Source file: XformSwetha2.java
* Purpose: This transform any three or more post conditions selected by
* the user.
*********************************************************************/


/*********************************************************************
* Applicable makes sure that the index is valid. Returns true if all *
* the selected post-conditions are valid.
*
* Parameters: target is the target method object
*              params refers to a Vector holding the indices of the
*                   selected post-conditions wrapped in a string.
*********************************************************************/
public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIS = (WsClasses.WsMethod) target;
      WsSubprogram subprgIS = methodIS.getWsMethodSubprogram();
      Vector postconIS = subprgIS.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      for(int i = 0; i < params1.size(); i++)
      {
            String pc1 = (String) params1.get(i);
            int index = Integer.parseInt(pc1);
            WsExpression exp;
            // Checking to see if it is a valid index.
            if(index <= postconIS.size() && index >=0)
                  exp = (WsExpression) postconIS.get(index);
            else
```

```
                return false;
        }
        return true;
}

/*********************************************************************
* This transform conjuncts the all the post conditions selected in the
* target method into one post condition and adds it to the
* post condition set. The individual post conditions are removed
*
* Parameters: params refers to a Vector holding the indices of the
*                   selected post-conditions wrapped in a string.
*********************************************************************/
public boolean execute(Object params)
{
        WsSubprogram subprog = targetmethod.getWsMethodSubprogram();
        Vector postconIS = subprog.getWsPostConditionSet();
        Vector params1 = (Vector) params;
        String pc1 = (String) params1.get(0);
        int index1 = Integer.parseInt(pc1);
        WsExpression exp1 = (WsExpression) postconIS.get(index1);
        String pc2 = (String) params1.get(1);
        int index2 = Integer.parseInt(pc2);
        WsExpression exp2 = (WsExpression) postconIS.get(index2);
        WsAnd expFinal = new WsAnd(exp1, exp2);
        if (params1.size() > 2)
        {
                for(int i = 2; i < params1.size(); i++)
                {
                        String pc = (String) params1.get(i);
                        int index = Integer.parseInt(pc);
                        WsExpression exp = (WsExpression)
                        postconIS.get(index);
                        WsAnd expinLoop = new WsAnd(expFinal, exp);
                        expFinal = expinLoop;
                }
        }

        postconIS.add(expFinal);
        //remove the individual post-conditions.
        for (int j = 0; j < params1.size(); j++)
        {
                String rem = (String) params1.get(j);
                int remindex = Integer.parseInt(rem);
                postconIS.setElementAt(null, remindex);
        }
        for (int k = postconIS.size() - 1; k >= 0; k--)
        {
                if (postconIS.get(k) == null)
                postconIS.remove(k);
        }
        return true;
}
```

Transform 3: XformSwetha3.java

```
/*********************************************************************
* Source file: XformSwetha2.java
```

```
* Purpose: This transform conjuncts all post conditions in the method.
**********************************************************************/

/**********************************************************************
* Applicable makes sure that there is more than one post condition in
* the post condition set. Returns true if more than one post condition
* in the post condition set.
*
* Parameters: target is the target method object.
**********************************************************************/
public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIS = (WsClasses.WsMethod) target;
      WsSubprogram subprgIS = methodIS.getWsMethodSubprogram();
      Vector postconIS = subprgIS.getWsPostConditionSet();
      if(postconIS.size() <= 1)
            return false;
      else
            return true;
}

/**********************************************************************
* This transform conjuncts the all the post conditions in the
* target method into one post condition and adds it to the
* post condition set. The individual post conditions are removed
*
* Parameters: None
**********************************************************************/
public boolean execute(Object params)
{
      WsSubprogram subprog = targetmethod.getWsMethodSubprogram();
      Vector postconIS = subprog.getWsPostConditionSet();
      WsExpression exp1 = (WsExpression) postconIS.get(0);
      WsExpression exp2 = (WsExpression) postconIS.get(1);
      WsAnd expFinal = new WsAnd(exp1, exp2);
      if (postconIS.size() > 2)
      {
            for(int i = 2; i < postconIS.size(); i++)
            {
                  WsExpression exp = (WsExpression) postconIS.get(i);
                  WsAnd expinLoop = new WsAnd(expFinal, exp);
                  expFinal = expinLoop;
            }
      }
      postconIS.add(expFinal);
      //remove the individual post-conditions.
      for (int k = postconIS.size() - 2; k >= 0; k--)
      {
            postconIS.remove(k);
      }
      return true;
}
```

Transform 4: XformSwetha4.java

```
/**********************************************************************
* Source file: XformSwetha4.java
```

```
* Purpose:  Converts post conditions of the form A in B' (where A is an
* element to be added to the set B) into a post condition involving
* array by incrementing the last index and placing the element A at
* that position.
**********************************************************************/

/**********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions has  an in operator in it and it is delcared
* as an array in the declarations.
*
* Parameters: target is the target method object
*             params refers to a Vector
*               element 0 contains the WsClass object of the class
*                that contains this method.
*               element 1 contains the WsPackage (AST).
*               element 2 contains the index of the selected
*                post-condition.
**********************************************************************/
public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);
      else
            return false;

      // check if the selected postcondition is has in operator in it.
      if(exp instanceof WsClasses.WsIn)
      {
            WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
            WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
            WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
            WsName arrayVarName = null;
            if(oprnd1 instanceof WsClasses.WsTick)
                  arrayVarName = ((WsTick) oprnd1).getWsTickName();
            else if(oprnd2 instanceof WsClasses.WsTick)
                  arrayVarName = ((WsTick) oprnd2).getWsTickName();
            else
                  return false;

            WsClasses.WsAttribute attributeIs =
            classIs.getWsClassDataComponent(arrayVarName.getName());
            if(attributeIs != null)
            {
                  WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
                  Vector Decls = packageIs.getWsDecls();
                  WsClasses.WsDeclaration temp;
```

```
                  for(int j=0; j<Decls.size(); j++)
                  {
                        temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs.getTypeName().equals(temp.getName()))
                  {
                        // once the attribute is found check to see
                        // that it has already been changed to array.
                        if(temp instanceof WsArrayType)
                        {
                              return true;
                        }
                  }
                  }
            }
      }
      return false;
}


/*****************************************************************
* Execute creates refenences to the array and to the index of the
* last element in the array (arrayNameMAX). These identifier refereces
* alongwith the new element to be inserted into the array are passed
* to the newPostCon method. The WsExpression obtained is added to the
* post condition set. The old post-condition (which has sets) is
* removed from the post condition set.
*
* Parameters: params refers to a Vector
*                  element 0 contains the WsClass object of the class
*                        that contains this method.
*                  element 1 contains the WsPackage (AST).
*                  element 2 contains the index of the selected
*                                    post-condition
*****************************************************************/
public boolean execute(Object params)
{
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp = (WsExpression) postconIS.get(index);
      WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
      WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
      WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
      WsName arrayVarName = null;
      if(oprnd2 instanceof WsClasses.WsTick)
            arrayVarName = ((WsTick) oprnd2).getWsTickName();

      WsClasses.WsAttribute attributeIs =
      classIs.getWsClassDataComponent(arrayVarName.getName());
      WsClasses.WsAttribute indexattributeIs =
      classIs.getWsClassDataComponent(arrayVarName.getName() + "MAX");
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      Vector Decls = packageIs.getWsDecls();
```

```
      WsClasses.WsDeclaration temp;
      WsIdentifier forArray = null;
      WsIdentifier forIndex = null;
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(attributeIs.getTypeName().equals(temp.getName()))
            {
                  if(temp instanceof WsArrayType)
                  {
                        forArray = temp.getWsDeclName();
                  }
            }

            if(indexattributeIs.getTypeName().equals(temp.getName()))
            {
                  forIndex = temp.getWsDeclName();
            }
      }

      // Create the Identifier Reference for the array
      WsIdentifierRef arrayRef = new
      WsIdentifierRef(arrayVarName.getName());
      arrayRef.setWsIdentRefTo(forArray);

      // Create the Identifier Reference for the array last index
      WsIdentifierRef lastIndexRef = new
      WsIdentifierRef(arrayVarName.getName() + "MAX");
      lastIndexRef.setWsIdentRefTo(forIndex);

      WsExpression newPC = newPostCon(arrayRef, lastIndexRef, oprnd1);
      postconIS.add(newPC);
      postconIS.remove(index);
      return true;
}



/*********************************************************************
* newPostCon creates the new expression (has arrays) which is
* equivalent to the selected post condition (has sets).
*
* Parameters : arrayRef - Identifier Refernce to the array object
*              LastIndexRef - Identifier Reference to the index of the
*                 last element
*              element - The new element to be added
* Returns a WsExpression.
*********************************************************************/

 private WsExpression newPostCon(WsIdentifierRef arrayRef,
                  WsIdentifierRef LastIndex, WsExpression element)
{
      WsIdentifierRef theArrayRef = arrayRef;
      WsIdentifierRef theLastIndex = LastIndex;
      WsExpression theElement = element;
      // part1 -- increment the last index by 1
      WsTick tickedIndex = new WsTick((WsName) theLastIndex);
```

```
      WsExpression temp = new WsLiteralInteger(1);
      WsAddition rhs = new WsAddition((WsExpression)theLastIndex,
(WsExpression)temp);
      WsEqual part1 = new WsEqual(tickedIndex, rhs);
      System.out.println("part1 is : " + part1.toString());
      //part2 -- set the element in the new value of the last index.
      WsTick tickedArray = new WsTick((WsName) theArrayRef);
      WsClasses.WsIndexedComponent indexName = new
            WsClasses.WsIndexedComponent();
      indexName.setWsIndxCompIndex((WsExpression)tickedIndex);
      indexName.setWsIndxCompName((WsName) tickedArray);
      WsEqual part2 = new WsEqual(indexName, theElement);
      System.out.println("part2 is : " + part2.toString());

      // pcFinal -- transformed postcondition
      WsAnd pcFinal = new WsAnd(part1, part2);
      System.out.println("tthe pc returned : " + pcFinal.toString());
      return pcFinal;
}
```

Transform 5: XformSwetha5.java

```
/*********************************************************************
* Source file: XformSwetha5.java
* Purpose:  Converts post conditions of the form not(A in B') (where A
* is an element to be deleted the set B) into a post condition
* involving array by replacing the element to be deletes by the last
* element and decreasing the last index by 1.
*********************************************************************/

/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-condition is in the required form.
*
* Parameters: target is the target method object
*             params refers to a Vector
*                  element 0 contains the WsClass object of the class
*                       that contains this method.
*                  element 1 contains the WsPackage (AST).
*                  element 2 contains the index of the selected
*                       post-condition
*********************************************************************/
public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);
```

```
      else
            return false;
// check if the selected postcondition is of the form NOT( A IN B')
      if(exp instanceof WsClasses.WsNot)
      {
            WsUnaryExpression Unaryexp = (WsUnaryExpression)exp;
            WsExpression exp1 = Unaryexp.getWsUnaryExpOp();
      // check if the selected postcondition is has in operator in it.
            if(exp1 instanceof WsClasses.WsIn)
            {
            WsBinaryExpression BinaryExp = (WsBinaryExpression) exp1;
            WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
            WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
            WsName arrayVarName = null;

                if(oprnd1 instanceof WsClasses.WsTick)
                  arrayVarName = ((WsTick) oprnd1).getWsTickName();
                else if(oprnd2 instanceof WsClasses.WsTick)
                  arrayVarName = ((WsTick) oprnd2).getWsTickName();
                else
                  return false;

            WsClasses.WsAttribute attributeIs =
            classIs.getWsClassDataComponent(arrayVarName.getName());
            if(attributeIs != null)
            {
            //finding the type of this attribute from the declerations.
                  WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
                  params1.get(1);
                  Vector Decls = packageIs.getWsDecls();
                  WsClasses.WsDeclaration temp;
                  for(int j=0; j<Decls.size(); j++)
                  {
                        temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs.getTypeName().equals(temp.getName()))
                  {
                  // once the attribute is found check to see
                  // that it has already been changed to arrayType.
                  if(temp instanceof WsArrayType
                  {
                        return true;
                  }
            }
            }
            }
      }
      }
      return false;
}



/*****************************************************************
* Execute creates references to the array and to the index of the
* last element in the array (arrayNameMAX). These identifier refereces
* alongwith the new element to be inserted into the array and the array
* index type are passed to the newPostCon method. The WsExpression
```

```
* obtained is added to the post condition set. The old post condition
* (which has sets) is removed from the post condition set.
*
* Parameters: params refers to a Vector
*                 element 0 contains the WsClass object of the class
*                       that contains this method.
*                 element 1 contains the WsPackage (AST).
*                 element 2 contains the index of the selected
*                       post-condition
*****************************************************************/
public boolean execute(Object params)
{
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      //System.out.println(subprgIs.toString());
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp = (WsExpression) postconIS.get(index);
      WsUnaryExpression Unaryexp = (WsUnaryExpression)exp;
      WsExpression exp1 = Unaryexp.getWsUnaryExpOp();
      WsBinaryExpression BinaryExp = (WsBinaryExpression) exp1;
      WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
      WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
      WsName arrayVarName = null;

      if(oprnd2 instanceof WsClasses.WsTick)
            arrayVarName = ((WsTick) oprnd2).getWsTickName();
      WsClasses.WsAttribute attributeIs =
      classIs.getWsClassDataComponent(arrayVarName.getName());
      WsClasses.WsAttribute indexattributeIs =
      classIs.getWsClassDataComponent(arrayVarName.getName() + "MAX");
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp;
      WsIdentifier forArray = null;
      WsIdentifier forIndex = null;
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(attributeIs.getTypeName().equals(temp.getName()))
            {
                  if(temp instanceof WsArrayType)
                  {
                        forArray = temp.getWsDeclName();
                  }
            }
            if(indexattributeIs.getTypeName().equals(temp.getName()))
            {
                  forIndex = temp.getWsDeclName();
            }
      }
      WsIdentifierRef arrayRef = new
      WsIdentifierRef(arrayVarName.getName());
      arrayRef.setWsIdentRefTo(forArray);
```

```
        System.out.println("array ref followed by Identifier" + arrayRef
        + forArray);

        WsIdentifierRef lastIndexRef = new
        WsIdentifierRef(arrayVarName.getName() + "MAX");
        lastIndexRef.setWsIdentRefTo(forIndex);
        System.out.println("index ref followed by Identifier" +
        lastIndexRef + forIndex);
        WsExpression newPC = newPostCon(arrayRef, lastIndexRef, forIndex,
        oprnd1);
        postconIS.add(newPC);
        postconIS.remove(index);
        return true;
}


/*********************************************************************
* Method newPostCon creates the new expression (has arrays) which is
* equivalent to the selected post condition (has sets).
*
* Parameters : arrayRef - Identifier Refernce to the array object
*              LastIndexRef - Identifier Reference to the index of the
*                       last element
*              type - Identifier ( the array index type)
*              element - The new element to be deleted
* Returns a WsExpression.
*********************************************************************/

private WsExpression newPostCon(WsIdentifierRef arrayRef,
          WsIdentifierRef LastIndex, WsIdentifier type, WsExpression
          element)
{
        WsIdentifierRef theArrayRef = arrayRef;
        WsIdentifierRef theLastIndex = LastIndex;
        WsIdentifier theType = type;
        WsExpression theElement = element;
        // get the tick for the last index and the array
        WsTick tickedIndex = new WsTick((WsName) theLastIndex);
        WsTick tickedArray = new WsTick((WsName) theArrayRef);
        WsIdentifier i = new WsIdentifier("i");
        WsIdentifierRef itemp = new WsIdentifierRef("i");
        itemp.setWsIdentRefTo(i);
        String temp1 = theType.getWsIdentSymbol();
        WsIdentifierRef tempRef = new WsIdentifierRef(temp1);
        tempRef.setWsIdentRefTo(theType);
        // logical variable
        WsLogicalVariable logi = new WsLogicalVariable();
        logi.setWsLogVarName(i);
        logi.setWsLogVarType(tempRef);
        // index component with tick with logical i
        WsClasses.WsIndexedComponent existcon = new
        WsClasses.WsIndexedComponent();
        existcon.setWsIndxCompIndex((WsExpression) itemp);
        existcon.setWsIndxCompName((WsName) theArrayRef);
        //System.out.println("WsIndexedComponent :  " + existcon);
        WsEqual existEqual = new WsEqual(existcon, theElement);
        //biex1  -- get the indexed component after deleting
```

```
      WsClasses.WsIndexedComponent biex1 = new
      WsClasses.WsIndexedComponent();
      biex1.setWsIndxCompIndex((WsExpression) itemp);  // changed the
      logi.getWsLogVarType() to itemp
      biex1.setWsIndxCompName((WsName) tickedArray);

      //biex2 -- get the indexed component before deleting
      WsClasses.WsIndexedComponent biex2 = new
      WsClasses.WsIndexedComponent();
      biex2.setWsIndxCompIndex((WsExpression)theLastIndex);
      biex2.setWsIndxCompName((WsName) arrayRef);
      //part 1b
      WsEqual part1b = new WsEqual(biex1, biex2);
      WsAnd and1 = new WsAnd(existEqual, part1b);
      // part2 - subtraction  top operator
      WsExpression temp = new WsLiteralInteger(1);
      WsSubtraction sub = new WsSubtraction((WsExpression)theLastIndex,
      (WsExpression)temp);
      WsEqual part2 = new WsEqual(tickedIndex, sub);
      // pcFinal -- transformed postcondition
      WsAnd and2 = new WsAnd(and1, part2);
      WsExistential ex = new WsExistential();
      ex.addWsQuantExpDeclaration(logi);
      ex.setWsQuantExpConstraint(and2);
      return ex;
}
```

Transform 6: XformSwetha6.java

```
/*********************************************************************
* Source file: XformSwetha6.java
* Purpose:  Transforms the post condition that checks for the presence
* of the element in the set to post condition that checks for the
* presence of the element in the array.
*********************************************************************/

/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions has  an in operator in it and it is delcared
* as an array in the declarations.
*
* Parameters: target is the target method object
*             params refers to a Vector
*                   element 0 contains the WsClass object of the class
*                       that contains this method.
*                  element 1 contains the WsPackage (AST).
*                  element 2 contains the index of the selected
*                       post-condition
*********************************************************************/


public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
```

```
Vector params1 = (Vector) params;
WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);

String pc = (String) params1.get(2);
int index = Integer.parseInt(pc);
WsExpression exp;
WsDataType theType = null;
// Checking to see if it is a valid index.
if(index <= postconIS.size() && index >=0)
      exp = (WsExpression) postconIS.get(index);
else
      return false;
if(exp instanceof WsClasses.WsAnd)
{
      WsBinaryExpression topAnd = (WsBinaryExpression) exp;
      WsExpression and2 = topAnd.getWsBinExpOp1();
      WsExpression topEqual = topAnd.getWsBinExpOp2();
      if(and2 instanceof WsClasses.WsAnd && topEqual instanceof
      WsClasses.WsEqual)
      {
            WsBinaryExpression opr = (WsBinaryExpression) and2;
            WsExpression impli1 = opr.getWsBinExpOp1();
            WsExpression not = opr.getWsBinExpOp2();
            if(impli1 instanceof WsClasses.WsImplication && not
            instanceof WsClasses.WsNot)
            {
                  WsBinaryExpression oprn = (WsBinaryExpression)
                  impli1;
                  WsExpression in1 = oprn.getWsBinExpOp1();
                  WsExpression equal1 = oprn.getWsBinExpOp2();
                  if(in1 instanceof WsClasses.WsIn && equal1
                  instanceof WsClasses.WsEqual)
                  {
                        WsBinaryExpression oprnd =
                        (WsBinaryExpression) in1;
                       WsExpression arr = oprnd.getWsBinExpOp2();
                        WsClasses.WsAttribute attributeIs =
      classIs.getWsClassDataComponent(((WsName)arr).getName());
                        if(attributeIs != null)
                        {
                              WsClasses.WsPackage packageIs =
                        (WsClasses.WsPackage) params1.get(1);
                        Vector Decls = packageIs.getWsDecls();
                        WsClasses.WsDeclaration temp;
                        for(int j=0; j<Decls.size(); j++)
                        {
                              temp = (WsClasses.WsDeclaration)
                              Decls.get(j);
            if(attributeIs.getTypeName().equals(temp.getName()))
            {
                  if(!(temp instanceof WsArrayType
                  {
                        return false;
                  }
            }
      }
}
```

```
      }
      WsUnaryExpression Unaryexp = (WsUnaryExpression) not;
      WsExpression impli2 = Unaryexp.getWsUnaryExpOp();
      if ( impli2 instanceof WsClasses.WsImplication )
      {
      WsBinaryExpression op = (WsBinaryExpression) impli2;
      WsExpression in2 = op.getWsBinExpOp1();
      WsExpression equal2 = op.getWsBinExpOp2();
      if ( in2 instanceof WsClasses.WsIn && equal2 instanceof
      WsClasses.WsEqual )
      {
            WsBinaryExpression oper = (WsBinaryExpression) in2;
            WsExpression arr2 = oper.getWsBinExpOp2();
            WsClasses.WsAttribute attributeIs =
            classIs.getWsClassDataComponent(((WsName)arr2).getName());

            if(attributeIs != null)
            {
                  WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
                  params1.get(1);
                  Vector Decls = packageIs.getWsDecls();
                  WsClasses.WsDeclaration temp;

                  for(int j=0; j<Decls.size(); j++)
                  {
                        temp = (WsClasses.WsDeclaration) Decls.get(j);

      if(attributeIs.getTypeName().equals(temp.getName()))
                                                {
      if(temp instanceof WsArrayType
      {
            return true;
      }
      }
      }
      }
      }
      }
      }
      }
      return false;
}/* END OF APPLICABLE



/*********************************************************************
* Execute creates references to the array, the element to be checked,
* the array index type, alongwith the references to the true, false,
* local variable and the function name are passed to the newPostCon
* method.
* The WsExpression obtained is added to the post condition set.
* The old post condition (which has sets) is removed from the
* post condition set.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
```

```
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* post-condition
*****************************************************************/

public boolean execute(Object params)
{
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;

      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsBinaryExpression topAnd = (WsBinaryExpression) exp;
      WsExpression and2 = topAnd.getWsBinExpOp1();
      WsExpression topEqual = topAnd.getWsBinExpOp2();
      WsBinaryExpression opr = (WsBinaryExpression) and2;
      WsExpression impli1 = opr.getWsBinExpOp1();
      WsExpression not = opr.getWsBinExpOp2();
      WsBinaryExpression oprn = (WsBinaryExpression) impli1;
      WsExpression in1 = oprn.getWsBinExpOp1();
      WsExpression equal1 = oprn.getWsBinExpOp2();
      WsBinaryExpression oprnd1 = (WsBinaryExpression) in1;
      WsExpression theELE = oprnd1.getWsBinExpOp1();
      WsExpression arrayasExp = oprnd1.getWsBinExpOp2();
      WsClasses.WsAttribute attributeIs =
      classIs.getWsClassDataComponent(((WsName)arrayasExp).getName());
      WsClasses.WsAttribute indexattributeIs =
      classIs.getWsClassDataComponent(((WsName)arrayasExp).getName() +
      "MAX");

      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp;
      WsIdentifier forArray = null;
      WsIdentifier forIndex = null;
      String name = " ";
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(attributeIs.getTypeName().equals(temp.getName()))
            {
                  if(temp instanceof WsArrayType)
                  {
                        forArray = temp.getWsDeclName();
                        System.out.println("forArray  : " + forArray);
                        name = attributeIs.getName();
                        System.out.println("name  : " + name);
                        }
            }
            if(indexattributeIs.getTypeName().equals(temp.getName()))
            {
                  forIndex = temp.getWsDeclName();
```

```
                  System.out.println("forIndex  : " + forIndex);
            }
      }
            WsIdentifierRef arrayRef = new WsIdentifierRef(name);
            arrayRef.setWsIdentRefTo(forArray);
            WsBinaryExpression oprnd2 = (WsBinaryExpression) equal1;
            WsExpression boolattri = oprnd2.getWsBinExpOp1();
            WsExpression isTrue = oprnd2.getWsBinExpOp2();

            WsIdentifierRef trueRef = new
            WsIdentifierRef(((WsName)isTrue).getName());
            WsName loName = null;
            if (boolattri instanceof WsClasses.WsTick)
            loName = ((WsTick)boolattri).getWsTickName();

            String varName = loName.getName();
            WsVariable localVariable = new WsVariable();
            if(subprgIs instanceof WsFunction)
            {
                  WsFunction functionIs = (WsFunction)subprgIs;
localVariable.setWsDataObjectType(functionIs.getWsFunctReturnType());
                  localVariable.setName(varName);
            }

            Vector tempV = subprgIs.getWsSubprogLocals();
            for(int i = 0; i < tempV.size(); i++)
            {
                  WsVariable var = (WsVariable) tempV.get(i);
                  if ((var.getTypeName()).equals(varName))
                  {
                        localVariable = var;
                  }
                  else
                  {
                        subprgIs.addWsSubprogLocal(localVariable);
                  }
            }
            WsIdentifierRef localRef = new WsIdentifierRef(varName);
            localRef.setWsIdentRefTo(localVariable.getWsName());
            WsUnaryExpression Unaryexp = (WsUnaryExpression) not;
            WsExpression impli2 = Unaryexp.getWsUnaryExpOp();
            WsBinaryExpression op = (WsBinaryExpression) impli2;
            WsExpression equal2 = op.getWsBinExpOp2();
            WsBinaryExpression opFAL = (WsBinaryExpression) equal2;
            WsExpression isFalse = opFAL.getWsBinExpOp2();
            WsIdentifierRef falseRef = new
            WsIdentifierRef(((WsName)isFalse).getName());
            WsBinaryExpression EQ = (WsBinaryExpression) topEqual;
            WsExpression fnName = EQ.getWsBinExpOp1();
            WsIdentifierRef fnRef = new
            WsIdentifierRef(((WsName)fnName).getName());

            Vector argV = new Vector();
            argV.add(fnRef);
            argV.add(localRef);
            argV.add(trueRef);
            argV.add(falseRef);
```

```
            System.out.println("calling new post cond");
            WsExpression newPC = newPostCon(arrayRef, forIndex, theELE,
            argV);
            System.out.println("back from new post cond");
            postconIS.add(newPC);
            postconIS.remove(index);

      return true;
}


/*********************************************************************
* Method newPostCon creates the new expression (has arrays) which is
* equivalent to the selected post condition (has sets).
*
* Parameters : arrayRef - Identifier Refernce to the array object
*              type - Identifier ( the array index type)
*              element - The element to be checked
*              arg - Vector holding Identifier Refs to
*                          (0) function name
*                          (1) boolean attribute
*                          (2) true
*                        and (3) false
* Returns a WsExpression.
*********************************************************************/

private WsExpression newPostCon(WsIdentifierRef arrayRef, WsIdentifier
indexType, WsExpression element, Vector arg)
{
      System.out.println("start new post con");
      // get the stuff out
      WsIdentifierRef theArrayRef = arrayRef;
      WsIdentifier theType = indexType;
      WsExpression theElement = element;
      Vector argV = arg;
      WsIdentifierRef fnRef = (WsIdentifierRef) argV.get(0);
      WsIdentifierRef lvRef = (WsIdentifierRef) argV.get(1);
      WsIdentifierRef trueRef = (WsIdentifierRef) argV.get(2);
      WsIdentifierRef falseRef = (WsIdentifierRef) argV.get(3);
            //tick the funcRef and the local variable Ref
      WsTick tickfnName = new WsTick((WsName) fnRef);
      WsTick ticklvName = new WsTick((WsName) lvRef);
      WsIdentifier i = new WsIdentifier("i");
      WsIdentifierRef itemp = new WsIdentifierRef("i");
      itemp.setWsIdentRefTo(i);

      String temp1 = theType.getWsIdentSymbol();
      WsIdentifierRef tempRef = new WsIdentifierRef(temp1);
      tempRef.setWsIdentRefTo(theType);
            // logical variable
      WsLogicalVariable logi = new WsLogicalVariable();
      logi.setWsLogVarName(i);
      logi.setWsLogVarType(tempRef);

      // index component with tick with logical i
      WsClasses.WsIndexedComponent existcon = new
      WsClasses.WsIndexedComponent();
```

```
      existcon.setWsIndxCompIndex((WsExpression) itemp);
      existcon.setWsIndxCompName((WsName) theArrayRef);
      WsEqual existEqual = new WsEqual(existcon, theElement);

      WsExistential exst = new WsExistential();
      exst.addWsQuantExpDeclaration(logi);
      exst.setWsQuantExpConstraint(existEqual);
      WsEqual tEqual = new WsEqual(ticklvName, trueRef);
      WsEqual fEqual = new WsEqual(ticklvName, falseRef);
      WsNot theNot = new WsNot(exst);

      WsImplication impli1 = new WsImplication(exst, tEqual);
      WsImplication impli2 = new WsImplication(theNot, fEqual);
      WsAnd part1 = new WsAnd(impli1, impli2);

      //make part 2
      WsEqual part2 = new WsEqual(tickfnName, ticklvName);
      // make the and statement for part 1 and part2
      WsAnd finalPC = new WsAnd(part1, part2);
      return finalPC;
}
```

Transform 7: XformSwetha7.java

```
/*********************************************************************
* Source file: XformSwetha7.java
* Purpose:  Finds post conditions in the form A in B or A in B' in the
* current class and converts the IN to an equivalent existential
* expression.
*********************************************************************/

/*********************************************************************
* Applicable makes sure that that the target class is an instance of
* WsClass in order to indicate whether the transform is applicable on
* the current selection or not. Returns a boolean value
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsPackage (AST).
*
*********************************************************************/

public static boolean applicable(Object target, Object params)
{
      WsClasses.WsClass classIs = (WsClasses.WsClass) target;
      if (classIs instanceof WsClass)
      {
            return true;
      }
      return false;
}
/*********************************************************************
* Execute creates a ChangeWsInVisitor object, passing in the current
* class and the params as the arguments to the constructor. The
* acceptVisitor method of the current class is called and the
* ChangeWsInVisitor object is passed.
```

```
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsPackage (AST).
*
*********************************************************************/

public boolean execute(Object params)
{
      System.out.println("in execute");
      ChangeWsInVisitor changeIn = new ChangeWsInVisitor(classIs,
      params);
      System.out.println("calling accept visitor method");
      classIs.acceptVisitor(changeIn, null);
    return true;
}
```

Transform 8: XformSwetha8.java

```
/*********************************************************************
* Source file: XformSwetha8.java
* Convert equality expressions to assignment statements.
* Requires that the operand on the left side to be an array.
* XformF1 does the same but it does not work for arrays.
*********************************************************************/

/*********************************************************************
* Sets the value of the expression to be transformed if the transform
* is called from another transform.
*
* Parameters: WsExpression object.
*********************************************************************/

public static void setEx(WsExpression ex1)
{
      ex = ex1;
}

/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions has  an array on the left hand side of the
* equality operator.
*
* Parameters: target is the target method object
*             params refers to a Vector
*                   element 0 contains the WsClass object of the class
*                       that contains this method.
*                   element 1 contains the WsPackage (AST).
*                   element 2 contains the index of the selected
*                       post-condition
*********************************************************************/

public static boolean applicable(Object target, Object params)
{System.out.println("in applicable");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
```

```
	Vector params1 = (Vector) params;
	WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
	String pc = (String) params1.get(2);
	int index = Integer.parseInt(pc);
	WsExpression exp;
	if (ex == null)
	{     // Checking to see if it is a valid index.
		if(index <= postconIS.size() && index >=0)
			exp = (WsExpression) postconIS.get(index);
		else
			return false;
	}
	else
		exp = (WsExpression) ex;
	if(exp instanceof WsEqual)
	{
		WsBinaryExpression Eq2 = (WsBinaryExpression) exp;
		WsExpression Eqchild1 =
		Eq2.getWsBinExpOp1();System.out.println("Eqchild1 : " +
		Eqchild1);
		WsExpression Eqchild2 =
		Eq2.getWsBinExpOp2();System.out.println("Eqchild2 : " +
		Eqchild2);
		if(Eqchild1 instanceof WsIndexedComponent == false)
			return false;
		else
		{
			WsName arrName = ((WsIndexedComponent)
			Eqchild1).getWsIndxCompName();
			if (arrName instanceof WsTick == false)
				return false;
			else
			{
				WsClasses.WsAttribute attributeIs =
			classIs.getWsClassDataComponent(arrName.getName());
				WsClasses.WsPackage packageIs =
				(WsClasses.WsPackage) params1.get(1);
				Vector Decls = packageIs.getWsDecls();
				WsClasses.WsDeclaration temp;
				for(int j=0; j<Decls.size(); j++)
				{
					temp = (WsClasses.WsDeclaration)
					Decls.get(j);
					if(attributeIs.getTypeName().equals(temp.
					getName()))
					{// once the attribute is found check to
see that it has already been changed to arrayType.
					if(temp instanceof WsArrayType == false)
						return false;
					}
				}
			}
		WsExpression indEx = ((WsIndexedComponent)
		Eqchild1).getWsIndxCompIndex();
		if (indEx instanceof WsTick)
			twoTick = true;
		else
```

```
                  twoTick = false;
            }
            if(Eqchild2 instanceof WsIndexedComponent)
            {
                  WsName arrName = ((WsIndexedComponent)
            Eqchild2).getWsIndxCompName();
            if (arrName instanceof WsTick)
                  threeTick = true;
            else
                  threeTick = false;
            WsClasses.WsAttribute attributeIs =
            classIs.getWsClassDataComponent(arrName.getName());
            WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
            params1.get(1);
            Vector Decls = packageIs.getWsDecls();
            WsClasses.WsDeclaration temp;
            for(int j=0; j<Decls.size(); j++)
            {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(attributeIs.getTypeName().equals(temp.getName()))
            {// once the attribute is found check to see that it has
already been changed to arrayType.
            if(temp instanceof WsArrayType == false)
                  return false;
            }
            }
            rhsarray = true;
            WsExpression indEx = ((WsIndexedComponent)
            Eqchild2).getWsIndxCompIndex();
            if (indEx instanceof WsTick)
                  {fourTick = true;}
            else
                  {fourTick = false;System.out.println("in applicable
check 7 index2 is not tick");}
            return true;
            }
            else
            {
            System.out.println("in applicable check 8 second not
array");}
            rhsarray = false;
            return true;
      }


      return false;
}




/*****************************************************************
* Execute transforms the post condition into code statements.
* Ticks are removed accordingly. The post condition is removed from the
* post condition set.
*
```

```
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*              that contains this method.
*                element 1 contains the WsPackage (AST).
*              element 2 contains the index of the selected
* post-condition
*****************************************************************/

public boolean execute(Object params)
{System.out.println("in execute... ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      Vector theBody;
      WsExpression exp,firstcomp, secondcomp;
      WsAssignment assgn;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      if (ex == null)
      {
            String pc = (String) params1.get(2);
            int index = Integer.parseInt(pc);
            exp = (WsExpression) postconIS.get(index);
      }
      else
            exp = (WsExpression) ex;
            firstcomp = (WsIndexedComponent) ((WsBinaryExpression)
            exp).getWsBinExpOp1();
            secondcomp = ((WsBinaryExpression) exp).getWsBinExpOp2();

            WsName ind1 = ((WsIndexedComponent)
            firstcomp).getWsIndxCompName();
            WsName indName1  = (WsName) ((WsTick)ind1).getWsTickName();
            ((WsIndexedComponent)
            firstcomp).setWsIndxCompName(indName1);

            if(twoTick == true)
            {System.out.println("unticking first index... ");
            WsExpression ind = ((WsIndexedComponent)
            firstcomp).getWsIndxCompIndex();
            WsName indName  = (WsName) ((WsTick)ind).getWsTickName();
            ((WsIndexedComponent)
            firstcomp).setWsIndxCompIndex(indName);

      }

      if(rhsarray == true)
      {System.out.println("2nd operator is an array");
            secondcomp = (WsIndexedComponent) secondcomp;
            if(threeTick == true)
            {System.out.println("unticking second array...");
                  WsName ind = ((WsIndexedComponent)
                  secondcomp).getWsIndxCompName();
                  WsName indName  = (WsName)
                  ((WsTick)ind).getWsTickName();
                  ((WsIndexedComponent)
                  secondcomp).setWsIndxCompName(indName);
```

```
                }

                if(fourTick == true)
                {System.out.println("unticking second index");
                      WsExpression ind = ((WsIndexedComponent)
secondcomp).getWsIndxCompIndex();
                      WsName indName  = (WsName)
((WsTick)ind).getWsTickName();
                      ((WsIndexedComponent)
secondcomp).setWsIndxCompIndex(indName);

                }
            }

      assgn = new WsAssignment((WsName) firstcomp, (WsExpression)
secondcomp);
      theBody = methodIs.getWsMethodSubprogram().getWsSubprogBody();
      theBody.add(assgn);
    postconIS.remove(exp);
      ex = null;
      rhsarray = false;
      twoTick = false;
      threeTick = false;
      fourTick = false;
      return true;
      }// END OF EXECUTE
```

Transform 9: XformSwetha9.java

```
/*****************************************************************
* Source file: XformSwetha9.java
* Purpose:  Converts post conditions of the form BMAX' = BMAX + 1 and
*           B'[BMAX'] = A.
*           Calls 3 Transforms
*                     i)   XformArjun5 to split the post condition
*                     ii)  XformF1 to convert BMAX' = BMAX + 1
*                     iii) XformSwetha8 to convert B[BMAX'] = A
*****************************************************************/

/*****************************************************************
* Applicable that the index of the post-conditions selected are valid.
* Returns true if the selected post-condition has the required form and
* if BMAX matches in two conjuncted expressions  and
* if the transforms XformArjun5, XformF1, XformSwetha8 are applicable.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*****************************************************************/
```

```
public static boolean applicable(Object target, Object params)
{System.out.println(" in applicable");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      String str1 = " ", str2 = " ";
            // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);
      else
            return false;

      // check for the format.  :: BMAX' = BMAX + 1 and B'[BMAX'] = A
      if(exp instanceof WsClasses.WsAnd)
      {
            WsBinaryExpression topAnd = (WsBinaryExpression) exp;
            WsExpression topchild1 = topAnd.getWsBinExpOp1();
            WsExpression topchild2 = topAnd.getWsBinExpOp2();
            if(topchild1 instanceof WsClasses.WsEqual)
            {
                  WsBinaryExpression Eq1 = (WsBinaryExpression)
                  topchild1;
                  WsExpression Eqchild1 = Eq1.getWsBinExpOp1();
                  WsExpression Eqchild2 = Eq1.getWsBinExpOp2();
                  if(Eqchild1 instanceof WsTick)
                  {
                        WsName idxName = ((WsTick)
                        Eqchild1).getWsTickName();
                        str1 = idxName.getName();
                  }
            }
            if(topchild2 instanceof WsClasses.WsEqual)
            {
                  WsBinaryExpression Eq2 = (WsBinaryExpression)
            topchild2;
                  WsExpression Eqchild1 = Eq2.getWsBinExpOp1();
                  WsExpression Eqchild2 = Eq2.getWsBinExpOp2();
                  if(Eqchild1 instanceof WsIndexedComponent)
                  {
                        WsName arrName = ((WsIndexedComponent)
                  Eqchild1).getWsIndxCompName();
                  WsClasses.WsAttribute attributeIs =
                  classIs.getWsClassDataComponent(arrName.getName());
                  WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
                  params1.get(1);
                  Vector Decls = packageIs.getWsDecls();
                  WsClasses.WsDeclaration temp;
                  for(int j=0; j<Decls.size(); j++)
                  {
                        temp = (WsClasses.WsDeclaration) Decls.get(j);
```

```
      if(attributeIs.getTypeName().equals(temp.getName()))
                                    {// once the attribute is found
check to see that it has already been changed to arrayType.
      if(temp instanceof WsArrayType == false)
                  return false;
      }
      }
            WsExpression arrIndex = ((WsIndexedComponent)
      Eqchild1).getWsIndxCompIndex();
      if(arrIndex instanceof WsTick)
      {str2 = ((WsTick) arrIndex).getName();}
      else
      {str2 = ((WsName) arrIndex).getName();}
      }
            XformSwetha8.setEx(topchild2);
      }
}

if(str1.equals(str2))
{System.out.println(" Check if XformArjun5 is applicable.....");
      if (XformArjun5.applicable(methodIs, null))
      {
            Transform myXform1 = new XformArjun5(methodIs);
            myXform1.execute(null);
            if (XformF1.applicable(methodIs, null) &&
            XformSwetha8.applicable(methodIs, params1))
            {System.out.println(" Checking if XformF1, XformSwetha8 are
applicable.....");return true;}
            }
      }
      return false;
}


/*****************************************************************
* Execute calls the Transform XformF1 to convert the first part BMAX' =
* BMAX + 1 and then calls the Transform XformSwetha8 to convert
* B'[BMAX'] = A into code statements.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* post-condition
*****************************************************************/

public boolean execute(Object params)
{
      Vector params1 = (Vector) params;
      params1.remove(2);
      Transform myXform2 = new XformF1(methodIs);
      myXform2.execute(null);
      Transform myXform3 = new XformSwetha8(methodIs);
      myXform3.execute(params1);
    return true;
```

```
}
```

Transform 10: XformSwetha10.java

```
/*********************************************************************
* Source file: XformSwetha10.java
* Purpose:  Converts the post conditions in the form
* X' = exists(i : Bindex)(B[i] = A) to code statements
*********************************************************************/


/*********************************************************************
*  Applicable makes sure that the index is valid.Returns true if the
* selected post-condition is in the required form.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*********************************************************************/

public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp;
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);
      else
            return false;
      boolean status;
      if (exp instanceof WsEqual)
      {
            WsBinaryExpression eq = (WsBinaryExpression) exp;
            WsExpression tick = eq.getWsBinExpOp1();
            WsExpression ex = eq.getWsBinExpOp2();
            if (ex instanceof WsExistential)
            {
                  CheckArrayVisitor check = new
                  CheckArrayVisitor(classIs, packageIs);
                  ex.acceptVisitor(check, null);
                  status = check.getStatus();
```

```
                  if(status == true)
                        return true;
                  else
                        return false;
            }
      }
      return false;
}/* END OF APPLICABLE


/*********************************************************************
* Execute transforms the selected post-condition into code statements.
* The selected post-condition is removed from the post-condition set.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*********************************************************************/

public boolean execute(Object params)
{System.out.println("in exe ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsBinaryExpression eq = (WsBinaryExpression) exp;
      WsExpression tick = eq.getWsBinExpOp1();
      WsExistential ex = (WsExistential)eq.getWsBinExpOp2();
      WsExpression exCon = ex.getWsQuantExpConstraint();
      WsLogicalVariable theVar = (WsLogicalVariable)
      ex.getWsQuantExpDeclarations().get(0);
      WsIdentifier dtId = theVar.getWsLogVarName();
      WsName dtName = theVar.getWsLogVarType();
      WsBinaryExpression eqCon = (WsBinaryExpression) exCon;
      WsExpression op1 = eqCon.getWsBinExpOp1();
      WsExpression ele = eqCon.getWsBinExpOp2();
      WsIndexedComponent eqId = (WsIndexedComponent) op1;
      WsName arr = eqId.getWsIndxCompName();
      WsExpression idx = eqId.getWsIndxCompIndex();
      WsClasses.WsAttribute indexattributeIs =
      classIs.getWsClassDataComponent(arr.getName() + "MAX");
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp;
      WsIdentifier forIndex = null;
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(indexattributeIs.getTypeName().equals(temp.getName()))
```

```
            {
                  forIndex = temp.getWsDeclName();   // get the
                  WsIdentifier for the array last index
            }
      }
      // Create the Identifier Reference for the array last index
      WsIdentifierRef lastIndexRef = new WsIdentifierRef(arr.getName()
      + "MAX");
      lastIndexRef.setWsIdentRefTo(forIndex);
      WsIdentifierRef identRef = new
      WsIdentifierRef(forIndex.getWsIdentSymbol());
      identRef.setWsIdentRefTo(forIndex);

      WsIdentifierRef iRef = new WsIdentifierRef("i");
      iRef.setWsIdentRefTo(dtId);
      WsDataObject iDataObj = new WsVariable();
      iDataObj.setWsDataObjectType(identRef);
      iDataObj.setWsDeclName(dtId);
      WsAssignment ini = new WsAssignment(iRef, new
      WsLiteralInteger(0));
      WsAddition addI = new WsAddition((WsExpression) iRef, new
      WsLiteralInteger(1));
      WsAssignment inc = new WsAssignment(iRef, addI);

      WsIdentifierRef ret = new WsIdentifierRef(subprgIs.getName());
      ret.setWsIdentRefTo(subprgIs.getWsDeclName());
      WsAssignment iftrue = new WsAssignment(ret, new
      WsIdentifierRef("true"));
      WsAssignment iffalse = new WsAssignment(ret, new
      WsIdentifierRef("false"));

      WsSelection ifelse = new WsSelection();
      ifelse.setWsSelCondition(new WsEqual(eqId, ele));
      ifelse.addWsSelThenPart(iftrue);
      ifelse.addWsSelElsePart(inc);

      WsIteration whileStatement = new WsIteration();
      whileStatement.setWsIterCondition(new WsLessThanOrEqual(iRef,
      lastIndexRef));
      whileStatement.addWsIterBody(ifelse);

      subprgIs.addWsSubprogLocal(iDataObj);
      subprgIs.addWsSubprogBody(ini);
      subprgIs.addWsSubprogBody(whileStatement);
      subprgIs.addWsSubprogBody(iffalse);
      postconIS.remove(index);

      return true;

}
```

Transform 11: XformSwetha11.java

```
/*****************************************************************
* Source file: XformSwetha11.java
* Purpose:  Converts the post conditions in the form
* exists (i : Bindex) (B'[i] = A) to code statements
*****************************************************************/

/*****************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions are valid and the selected post condition is
* in the required form and the indexed component is an array.
*
* Parameters: target is the target method object
*                   params refers to a Vector holding the indices of he
*                   selected post-conditions wrapped in a string.
*****************************************************************/

public static boolean applicable(Object target, Object params)
{System.out.println("in applicable ");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;

      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);  // get the
expression
      else
            return false;

      if(exp instanceof WsExistential)
      {
            WsExistential top = (WsExistential) exp;
            WsExpression expConstraint = top.getWsQuantExpConstraint();
            WsLogicalVariable theVar = (WsLogicalVariable)
            top.getWsQuantExpDeclarations().get(0);

            if (expConstraint instanceof WsEqual)
            {
                  WsBinaryExpression eq = (WsBinaryExpression)
                  expConstraint;
                  WsExpression eqIdx = eq.getWsBinExpOp1();
                  WsExpression eqEle = eq.getWsBinExpOp2();


                  if (eqIdx instanceof WsIndexedComponent)
                  {
                        WsIndexedComponent eqId = (WsIndexedComponent)
                        eqIdx;
                        WsExpression id1 = eqId.getWsIndxCompName();
                        WsExpression id2 = eqId.getWsIndxCompIndex();
                        if (id1 instanceof WsTick)
                        {
```

```
                              WsTick tName = (WsTick) id1;
                              WsName arrName = (WsName)
                              tName.getWsTickName();
                              WsClasses.WsAttribute attributeIs =
classIs.getWsClassDataComponent(arrName.getName());
                              if(attributeIs != null)
                              {
                                    WsClasses.WsPackage packageIs =
(WsClasses.WsPackage) params1.get(1);
                              Vector Decls = packageIs.getWsDecls();
                              WsClasses.WsDeclaration temp;
                              for(int j=0; j<Decls.size(); j++)
                              {
                                    temp = (WsClasses.WsDeclaration)
Decls.get(j);


      if(attributeIs.getTypeName().equals(temp.getName()))
                                                {

if(temp instanceof WsArrayType)// confirms that the attribute is array
type
      {
      return true;
                  }
            }
      }
            }
      }
}
}
}
return false;
}


/*********************************************************************
* Execute creates refenences to the array and to the index of the
* last element in the array (arrayNameMAX). These identifier refereces
* alongwith the new element to be inserted into the array are passed
* to the newPostCon method. The WsExpression obtained is added to the
* post condition set. The old post condition (which has sets) is
* removed from the post condition set.
*
* Parameters: params refers to a Vector
*                  element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* post-condition
*********************************************************************/


public boolean execute(Object params)
{System.out.println("in exe ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      System.out.println(subprgIs.toString());
```

```
Vector postconIS = subprgIs.getWsPostConditionSet();
Vector params1 = (Vector) params;
WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
String pc = (String) params1.get(2);
int index = Integer.parseInt(pc);
WsExpression exp = (WsExpression) postconIS.get(index);
WsExistential top = (WsExistential) exp;
WsExpression expConstraint = top.getWsQuantExpConstraint();
WsLogicalVariable theVar = (WsLogicalVariable)
top.getWsQuantExpDeclarations().get(0);

WsBinaryExpression eq = (WsBinaryExpression) expConstraint;
WsExpression eqIdx = eq.getWsBinExpOp1();
WsExpression Element = eq.getWsBinExpOp2();
WsIndexedComponent eqId = (WsIndexedComponent) eqIdx;
WsExpression id1 = eqId.getWsIndxCompName();
WsExpression id2 = eqId.getWsIndxCompIndex();
WsTick tName = (WsTick) id1;
WsName arrName = (WsName) tName.getWsTickName();

WsClasses.WsAttribute attributeIs =
classIs.getWsClassDataComponent(arrName.getName());
WsClasses.WsAttribute indexattributeIs =
classIs.getWsClassDataComponent(arrName.getName() + "MAX");
WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
Vector Decls = packageIs.getWsDecls();
WsClasses.WsDeclaration temp;

WsIdentifier forArray = null;
WsIdentifier forIndex = null;

for(int j=0; j<Decls.size(); j++)
{
      temp = (WsClasses.WsDeclaration) Decls.get(j);
      if(attributeIs.getTypeName().equals(temp.getName()))
      {
            if(temp instanceof WsArrayType)
            {
                  forArray = temp.getWsDeclName();
            }
      }

if(indexattributeIs.getTypeName().equals(temp.getName()))
      {
            forIndex = temp.getWsDeclName();               }
      }

      // Create the Identifier Reference for the array
      WsIdentifierRef arrayRef = new
      WsIdentifierRef(arrName.getName());
      arrayRef.setWsIdentRefTo(forArray);

      // Create the Identifier Reference for the array last index
      WsIdentifierRef lastIndexRef = new
      WsIdentifierRef(arrName.getName() + "MAX");
      lastIndexRef.setWsIdentRefTo(forIndex);
```

```
            System.out.println("newPostCon called ");
            WsExpression newPC = newPostCon(arrayRef, lastIndexRef,
            Element);
            postconIS.remove(index);
            postconIS.add(newPC);
            if(XformSwetha9.applicable(methodIs, params1))
            {System.out.println("XformSwetha9 is applicable");
                  Transform myXform = new XformSwetha9(methodIs);
                  myXform.execute(params1);
            }
      return true;
}


/*********************************************************************
* newPostCon creates the new expression (has arrays) which is
* equivalent to the selected post condition (has sets).
*
* Parameters : arrayRef - Identifier Refernce to the array object
*              LastIndexRef - Identifier Reference to the index of the
* last element
*              element - The new element to be added
* Returns a WsExpression
*********************************************************************/

private WsExpression newPostCon(WsIdentifierRef arrayRef,
WsIdentifierRef LastIndex, WsExpression element)
{
      WsIdentifierRef theArrayRef = arrayRef;
      WsIdentifierRef theLastIndex = LastIndex;
      WsExpression theElement = element;
      // part1 -- increment the last index by 1
      WsTick tickedIndex = new WsTick((WsName) theLastIndex);
      WsExpression temp = new WsLiteralInteger(1);
      WsAddition rhs = new WsAddition((WsExpression)theLastIndex,
      (WsExpression)temp);
      WsEqual part1 = new WsEqual(tickedIndex, rhs);
      System.out.println("part1 is : " + part1.toString());

      //part2 -- set the element in the new value of the last index.
      WsTick tickedArray = new WsTick((WsName) theArrayRef);
      WsClasses.WsIndexedComponent indexName = new
      WsClasses.WsIndexedComponent();
      indexName.setWsIndxCompIndex((WsExpression)tickedIndex);
      indexName.setWsIndxCompName((WsName) tickedArray);
      WsEqual part2 = new WsEqual(indexName, theElement);
      System.out.println("part2 is : " + part2.toString());
      // pcFinal -- transformed postcondition
      WsAnd pcFinal = new WsAnd(part1, part2);
      System.out.println("tthe pc returned : " + pcFinal.toString());
      return pcFinal;
}
```

Transform 12: XformSwetha12.java

```
/*********************************************************************
* Source file: XformSwetha12.java
```

```
* Purpose:  Transforms the post conditions in the form
*           exists (i : Bindex) (B[i] = A) and exp1 and exp2 and exp3.
**********************************************************************/

/**********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions are valid and the selected post condition is
* in the required form and the indexed component is an array.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
**********************************************************************/

public static boolean applicable(Object target, Object params)
{System.out.println("in applicable ");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      boolean status;
// Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);  // get the
expression
      else
            return false;
      if(exp instanceof WsExistential)
      {
            WsExistential top = (WsExistential) exp;
            WsExpression expConstraint = top.getWsQuantExpConstraint();
            WsLogicalVariable theVar = (WsLogicalVariable)
            top.getWsQuantExpDeclarations().get(0);
            CheckArrayVisitor check = new CheckArrayVisitor(classIs,
packageIs);
            expConstraint.acceptVisitor(check, null);
            status = check.getStatus();
            if(status == true)
                  return true;
            else
                  return false;
            }
            return false;
}
 /**********************************************************************
* Execute transforms the seleted post condition into code statements.
* The post condition is removed from the post condition set.
```

```
* The expressions after (B[i] = A) are treated as the post condition to
* the new method created by this transform.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition(existential)
*****************************************************************/

public boolean execute(Object params)
{System.out.println("in exe ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsExistential top = (WsExistential) exp;
      WsExpression expConstraint = top.getWsQuantExpConstraint();
      WsLogicalVariable theVar = (WsLogicalVariable)
      top.getWsQuantExpDeclarations().get(0);
      WsName dtName = theVar.getWsLogVarType();
      WsIdentifier dtId = theVar.getWsLogVarName();
      WsExpression e = expConstraint;
      Vector v1 = ToolUtils.returnExpressions(expConstraint);
      int size = v1.size();
      WsExpression[] exArr = new WsExpression[size];
      int place = size - 1;
      while(e != null && place > 0)
      {
            if(e instanceof WsClasses.WsAnd)
            {
                  WsBinaryExpression BinaryExp = (WsBinaryExpression) ;
                  WsExpression temp1 = BinaryExp.getWsBinExpOp1();
                  WsExpression temp2 = BinaryExp.getWsBinExpOp2();
                  exArr[place] = temp2;
                  exArr[place - 1] = temp1;
                  e = temp1;
                  place--;
            }
            else
                  e = null;
      }
      WsBinaryExpression eq = (WsBinaryExpression) exArr[0];
      WsExpression eqIdx = eq.getWsBinExpOp1();
      WsExpression Element = eq.getWsBinExpOp2();
      Vector post = new Vector();
      for(int rem = 1; rem < exArr.length; rem++)
      post.add(exArr[rem]);
      WsIndexedComponent eqId = (WsIndexedComponent) eqIdx;
      WsName arrName = eqId.getWsIndxCompName();
```

```
      WsExpression id2 = eqId.getWsIndxCompIndex();
      WsClasses.WsAttribute attributeIs =
      classIs.getWsClassDataComponent(arrName.getName());
      WsClasses.WsAttribute indexattributeIs =
      classIs.getWsClassDataComponent(arrName.getName() + "MAX");
      WsClasses.WsPackage packageIS = (WsClasses.WsPackage)
      params1.get(1);
      Vector Decls = packageIS.getWsDecls();
      WsClasses.WsDeclaration temp;
      WsIdentifier forArray = null;
      WsIdentifier forIndex = null;
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(attributeIs.getTypeName().equals(temp.getName()))
            {
                  if(temp instanceof WsArrayType)
                  {
                        forArray = temp.getWsDeclName();
                  }
            }
            if(indexattributeIs.getTypeName().equals(temp.getName()))
            {
                  forIndex = temp.getWsDeclName();   // get the
WsIdentifier for the array last index
            }
      }
            // Create the Identifier Reference for the array
WsIdentifierRef arrayRef = new WsIdentifierRef(arrName.getName());
      arrayRef.setWsIdentRefTo(forArray);

      // Create the Identifier Reference for the array last index
      WsIdentifierRef lastIndexRef = new
      WsIdentifierRef(arrName.getName() + "MAX");
      lastIndexRef.setWsIdentRefTo(forIndex);

      WsIdentifierRef identRef = new
WsIdentifierRef(forIndex.getWsIdentSymbol());
      identRef.setWsIdentRefTo(forIndex);

      //creating statements for local variable i and to increment it
      WsIdentifierRef iRef = new WsIdentifierRef("i");
      iRef.setWsIdentRefTo(dtId);
      WsDataObject iDataObj = new WsVariable();
      iDataObj.setWsDataObjectType(identRef);
      iDataObj.setWsDeclName(dtId);
      WsExpression temp0 = new WsLiteralInteger(0);
      WsAssignment ini = new WsAssignment(iRef, temp0);
      WsExpression temp1 = new WsLiteralInteger(1);
      WsAddition addI = new WsAddition((WsExpression) iRef,
(WsExpression) temp1);
      WsAssignment inc = new WsAssignment(iRef, addI);

      // creating the procedure
      WsProcedure procA = new WsProcedure();
      String procName = "doImplementation";
      for(int k=1; classIs.getWsClassOperation(procName)!= null; k++)
```

```
        procName = "doImplementation" + k;

        procA.setName(procName);
        procA.setWsPostConditions(post);
        WsParameter para = new WsParameter("i",
        forIndex.getWsIdentSymbol());
        para.setWsParameterIn(true);
        procA.addWsSubprogFormal(para);
        Vector paraV = new Vector();
        paraV.add(para.getWsName());
        WsMethod m = new WsMethod(procA);
        m.setWsPrivate(true);
        m.setWsClassMethod(false);
        //Adding the procedure to the Class
        classIs.addWsClassOperation(m);
        WsProcedureCall procCall = new WsProcedureCall(procA.getName());
        procCall.setWsSubprogCallArgs(paraV);
        WsSelection ifelse = new WsSelection();
        ifelse.setWsSelCondition(eq);
        ifelse.addWsSelThenPart(procCall);
        ifelse.addWsSelElsePart(inc);
        //create the while
        //creating the while condition
        WsLessThanOrEqual itercond = new WsLessThanOrEqual(iRef,
        lastIndexRef);
        WsIteration whileStatement = new WsIteration();
        whileStatement.setWsIterCondition(itercond);
        whileStatement.addWsIterBody(ifelse);
        subprgIs.addWsSubprogLocal(iDataObj);
        subprgIs.addWsSubprogBody(ini);
        subprgIs.addWsSubprogBody(whileStatement);
        //remove the individual post-conditions.
        for (int p = 2; p < params1.size(); p++)
        {
              String rem = (String) params1.get(p);
              int remindex = Integer.parseInt(rem);
              postconIS.setElementAt(null, remindex);
        }
        for (int q = postconIS.size() - 1; q >= 0; q--)
        {
              if (postconIS.get(q) == null)
              postconIS.remove(q);
        }

        return true;
}
```

Transform 13: XformSwetha13.java

```
/*********************************************************************
```

```
* Source file: XformSwetha13.java
* Purpose:  Tranforms the post conditions in the form
* not(exists (i : D) Q) to forall(i : D) not Q
*****************************************************************/

/*****************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions has  an in operator in it and it is delcared
* as an array in the declarations.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the lass
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*****************************************************************/

public static boolean applicable(Object target, Object params)
{System.out.println("in exe ");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      boolean status;
      System.out.println("in applicable 2");
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);      else
                  return false;
      if ( exp instanceof WsNot)
      {     WsUnaryExpression Uexp = (WsUnaryExpression) exp;
            WsExpression child = Uexp.getWsUnaryExpOp();
            if ( child instanceof WsExistential)
            {     WsExistential exist = (WsExistential) child;
            WsExpression existCon = exist.getWsQuantExpConstraint();
            CheckArrayVisitor check = new CheckArrayVisitor(classIs,
packageIs);
            existCon.acceptVisitor(check, null);
            status = check.getStatus();
            if(status == true)
            return true;
            else
            return false;
            }}
      return false;
}/* END OF APPLICABLE
/*****************************************************************
* Execute transforms the post condition not(exists (i : D) Q)
*                 to forall(i : D) not Q
```

```
* The transfomed post condition is removed from the post condition set
* and the new post condition is added to the post condition set
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
*postcondition
*****************************************************************/
public boolean execute(Object params)
{
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
      params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsUnaryExpression Uexp = (WsUnaryExpression) exp;
      WsExpression child = Uexp.getWsUnaryExpOp();
      WsExistential exist = (WsExistential) child;
      WsExpression existCon = exist.getWsQuantExpConstraint();
      Vector decs = exist.getWsQuantExpDeclarations();
      WsNot newCon = new WsNot(existCon);
      WsUniversal uni = new WsUniversal();
      uni.setWsQuantExpConstraint(newCon);
      uni.setWsQuantExpDeclarations(decs);
      postconIS.add(uni);
      postconIS.remove(index);
      return true;
}
```

Transform 14: XformSwetha14.java

```
/*****************************************************************
* Source file: XformSwetha14.java
* Purpose:  Tranforms the post conditions in the form
* forall(i : D) not Q to not(exists (i : D) Q)
*****************************************************************/

/*****************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions has  an in operator in it and it is delcared
* as an array in the declarations.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*           element 2 contains the index of the selected postcondition
*****************************************************************/
```

```
public static boolean applicable(Object target, Object params)
{System.out.println("in exe ");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      boolean status;
      System.out.println("in applicable 2");
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);  // get the
expression
      else
            return false;

      if ( exp instanceof WsUniversal)
      {
            WsUniversal Uexp = (WsUniversal) exp;
            WsExpression con = Uexp.getWsQuantExpConstraint();
            if ( con instanceof WsNot)
            {
            WsUnaryExpression Unot = (WsUnaryExpression) con;
            WsExpression child = Unot.getWsUnaryExpOp();
            CheckArrayVisitor check = new CheckArrayVisitor(classIs,
packageIs);
            child.acceptVisitor(check, null);
            status = check.getStatus();
            if(status == true)
                  return true;
            else
                  return false;
      }
      }

      return false;
}/* END OF APPLICABLE


/*****************************************************************
* Execute transforms the post condition forall(i : D) not Q
*                 to not(exists (i : D) Q)
* The transfomed post condition is removed from the post condition set
* and the new post condition is added to the post condition set
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*****************************************************************/
```

```
public boolean execute(Object params)
{
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsUniversal forall = (WsUniversal) exp;
      WsExpression forallCon = forall.getWsQuantExpConstraint();
      Vector decs = forall.getWsQuantExpDeclarations();
      WsUnaryExpression Uexp = (WsUnaryExpression) forallCon;
      WsExpression child = Uexp.getWsUnaryExpOp();
      WsExistential exist = new WsExistential();
      exist.setWsQuantExpConstraint(child);
      exist.setWsQuantExpDeclarations(decs);
      WsNot finalpc = new WsNot(exist);
      postconIS.add(finalpc);
      postconIS.remove(index);
      return true;
}
```

Transform 15: XformSwetha15.java

```
/*********************************************************************
* Source file: XformSwetha15.java
* Purpose:  Converts Tranforms the post conditions in the form
* forall(i : D) not Q
*                 to code statements
*********************************************************************/

/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions are valid and the selected post condition is
* in the required form and the indexed component is an array.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
*postcondition
*********************************************************************/
public static boolean applicable(Object target, Object params)
{System.out.println("in exe ");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
```

```
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      boolean status;
      System.out.println("in applicable 2");
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);  // get the
expression
      else
            return false;
      if ( exp instanceof WsUniversal)
      {
            WsUniversal Uexp = (WsUniversal) exp;
            WsExpression con = Uexp.getWsQuantExpConstraint();
            if ( con instanceof WsNot)
            {
                  WsUnaryExpression Unot = (WsUnaryExpression) con;
                  WsExpression child = Unot.getWsUnaryExpOp();
                  CheckArrayVisitor check = new
CheckArrayVisitor(classIs, packageIs);
                  child.acceptVisitor(check, null);
                  status = check.getStatus();
                  if(status == true)
                        return true;
                  else
                        return false;
            }
      }
      return false;
}/* END OF APPLICABLE


/*********************************************************************
* Execute transforms the selected post-condition into code statements.
* The selected post-condition is removed from the post-condition set.
* The code statements include a function call to another function
reated and
* added to the class operations by the current transform. The onstraint
expression
* in the forall is passed down as the post condition to this new
* function.
* The parameter of the current method alongwith the logical variable in
* forall are passed down as the parameters to the new function
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*********************************************************************/
public boolean execute(Object params)
{
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
```

```
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);

      WsUniversal forall = (WsUniversal) exp;
      WsExpression forallCon = forall.getWsQuantExpConstraint();
      WsLogicalVariable theVar = (WsLogicalVariable)
forall.getWsQuantExpDeclarations().get(0);
      WsIdentifier dtId = theVar.getWsLogVarName();
      WsName dtName = theVar.getWsLogVarType();
      WsExpression child = ((WsNot)forallCon).getWsUnaryExpOp();
      WsExpression op1 = ((WsBinaryExpression) child).getWsBinExpOp1();
      WsIndexedComponent eqId = (WsIndexedComponent) op1;
      WsName arr = eqId.getWsIndxCompName();
      WsName arrName = ((WsTick) arr) .getWsTickName();
      WsClasses.WsAttribute indexattributeIs =
classIs.getWsClassDataComponent(arrName.getName() + "MAX");
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp;
      WsIdentifier forIndex = null;
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
      if(indexattributeIs.getTypeName().equals(temp.getName()))
      {
            forIndex = temp.getWsDeclName();   // get the WsIdentifier
or the array last index
            }
      }
      // Create the Identifier Reference for the array last index
      WsIdentifierRef lastIndexRef = new
      WsIdentifierRef(arrName.getName() + "MAX");
      lastIndexRef.setWsIdentRefTo(forIndex);
      WsIdentifierRef identRef = new
WsIdentifierRef(forIndex.getWsIdentSymbol());
      identRef.setWsIdentRefTo(forIndex);
      WsIdentifierRef iRef = new WsIdentifierRef("i");
      iRef.setWsIdentRefTo(dtId);
WsDataObject iDataObj = new WsVariable();
      iDataObj.setWsDataObjectType(identRef);
      iDataObj.setWsDeclName(dtId);
      WsAssignment ini = new WsAssignment(iRef, new
WsLiteralInteger(0));
      WsAddition addI = new WsAddition((WsExpression) iRef, new
WsLiteralInteger(1));
      WsAssignment inc = new WsAssignment(iRef, addI);
      WsName fRet = new WsIdentifierRef("boolean");
      WsIdentifier FRI = new WsIdentifier("fRet");
      WsName fRet2 = new WsIdentifierRef("fRet");
      WsDataObject bDataObj = new WsVariable();
      bDataObj.setWsDataObjectType(fRet);
```

```
      bDataObj.setWsDeclName(FRI);
      WsAssignment ini2 = new WsAssignment(fRet2, new
WsIdentifierRef("false"));
      // creating the function
      WsFunction funcA = new WsFunction();
      String funcName = "doImplementation";
      for(int k=1; classIs.getWsClassOperation(funcName)!= null; k++)
            funcName = "doImplementation" + k;
      funcA.setName(funcName);
      funcA.setWsPostConditions(forallCon);
      WsParameter para1 = new WsParameter("i",
      forIndex.getWsIdentSymbol());
      para1.setWsParameterIn(true);
      WsParameter temppara = (WsParameter)
subprgIs.getWsSubprogFormals().get(0);
      WsParameter para2 = new WsParameter();
      para2.setWsParameterName(temppara.getName());
      para2.setWsParameterType(temppara.getType());
      para2.setWsParameterIn(true);
      funcA.addWsSubprogFormal(para1);
      funcA.addWsSubprogFormal(para2);
      funcA.setWsFunctReturnType(new WsIdentifierRef("boolean"));
      System.out.println("name is : " + temppara.getName() + "   type
is : " + temppara.getType());

      Vector paraV = new Vector();
      paraV.add(para1.getWsName());
      paraV.add(para2.getWsName());
      //funcA.setWsSubprogFormals(paraV);

      WsMethod m = new WsMethod(funcA);
      m.setWsPrivate(true);
      m.setWsClassMethod(false);
      //Adding the function to the Class
      classIs.addWsClassOperation(m);
      WsFunctionCall funcCall = new WsFunctionCall(funcA.getName());
      funcCall.setWsSubprogCallArgs(paraV);
      WsAssignment retVal = new WsAssignment(fRet2, funcCall);
      WsSelection ifelse = new WsSelection();
      //need to get the condition -- such tht there is no deletion ie a
-- so can increment
      ifelse.setWsSelCondition(new WsEqual(fRet2, new
WsIdentifierRef("false")));
      ifelse.addWsSelThenPart(inc);
      WsIteration whileStatement = new WsIteration();
      whileStatement.setWsIterCondition(new WsLessThanOrEqual(iRef,
lastIndexRef));
      whileStatement.addWsIterBody(retVal);
      whileStatement.addWsIterBody(ifelse);
      subprgIs.addWsSubprogLocal(iDataObj);
      subprgIs.addWsSubprogLocal(bDataObj);
      subprgIs.addWsSubprogBody(ini);
      subprgIs.addWsSubprogBody(ini2);
      subprgIs.addWsSubprogBody(whileStatement);
      postconIS.remove(index);
      return true;
}
```

Transform 16: XformSwetha16.java

```
/*********************************************************************
* Source file: XformSwetha16.java
* Purpose:  Converts the post conditions in the form not(B'[i] = A) to
code statements. Needs to be called after XformSwetha15.
*********************************************************************/

/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-conditions has  an in operator in it and it is delcared
* as an array in the declarations.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected ostcondition
*********************************************************************/

public static boolean applicable(Object target, Object params)
{System.out.println("in exe ");
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      boolean status;
      System.out.println("in applicable 2");
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);  // get the
expression
      else
      return false;
      if (exp instanceof WsNot)
      {
            WsUnaryExpression Uexp = (WsUnaryExpression) exp;
            WsExpression child = Uexp.getWsUnaryExpOp();
            if (child instanceof WsEqual)
            {
            WsBinaryExpression eq = (WsBinaryExpression) child;
            CheckArrayVisitor check = new CheckArrayVisitor(classIs,
packageIs);
            child.acceptVisitor(check, null);
            status = check.getStatus();

            if(status == true)
                  return true;
```

```
            else
                  return false;
            }
      }
return false;
}/* END OF APPLICABLE


/*****************************************************************
* Execute transforms the current post condition into code statements.
* The selected post condition is removed from the post condition set.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected ostcondition
*****************************************************************/

public boolean execute(Object params)
{
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsExpression child = ((WsNot)exp).getWsUnaryExpOp();
      WsExpression op1 = ((WsBinaryExpression) child).getWsBinExpOp1();
      WsExpression ele = ((WsBinaryExpression) child).getWsBinExpOp2();
      WsIndexedComponent eqId = (WsIndexedComponent) op1;
      WsName arr = eqId.getWsIndxCompName();
      WsExpression idx = eqId.getWsIndxCompIndex();
      WsName arrName = ((WsTick) arr) .getWsTickName();
      eqId.setWsIndxCompName(arrName);
      eqId.setWsIndxCompIndex(idx);
      WsClasses.WsAttribute indexattributeIs =
classIs.getWsClassDataComponent(arrName.getName() + "MAX");
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp;
      WsIdentifier forIndex = null;
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(indexattributeIs.getTypeName().equals(temp.getName()))
            {
                  forIndex = temp.getWsDeclName();   // get the
sIdentifier for the array last index
            }
      }
// Create the Identifier Reference for the array last index
WsIdentifierRef lastIndexRef = new WsIdentifierRef(arrName.getName() +
"MAX");
```

```
lastIndexRef.setWsIdentRefTo(forIndex);
      WsIndexedComponent newIdx = new WsIndexedComponent();
      newIdx.setWsIndxCompName(arrName);
      newIdx.setWsIndxCompIndex(lastIndexRef);
      //create the code statements -- if element found
      WsAssignment st1 = new WsAssignment(eqId, newIdx);
      WsSubtraction sub = new WsSubtraction(lastIndexRef, new
WsLiteralInteger(1));
      WsAssignment st2 = new WsAssignment(lastIndexRef, sub);
      WsIdentifierRef ret = new WsIdentifierRef(subprgIs.getName());
      ret.setWsIdentRefTo(subprgIs.getWsDeclName());
      WsAssignment iftrue = new WsAssignment(ret, new
WsIdentifierRef("true"));
      WsAssignment iffalse = new WsAssignment(ret, new
WsIdentifierRef("false"));
            //create the if
      WsSelection ifelse = new WsSelection();
      ifelse.setWsSelCondition(new WsEqual(eqId, ele));
      ifelse.addWsSelThenPart(st1);
      ifelse.addWsSelThenPart(st2);
      ifelse.addWsSelThenPart(iftrue);
      ifelse.addWsSelElsePart(iffalse);
      WsFunction funcIs = (WsFunction) subprgIs;
      funcIs.addWsSubprogBody(ifelse);
      funcIs.setWsFunctReturnType(new WsIdentifierRef("boolean"));
      postconIS.remove(index);
      return true;
}
```

Transform 17: XformSwetha17.java

```
/*********************************************************************
* Source file: XformSwetha17.java
* Purpose:  Converts the post conditions in the form
* exists (i : Bindex) (B'[i] = A) to code statements
*********************************************************************/

/*********************************************************************
* Execute transforms the selected post-condition into code statements.
* The selected post-condition is removed from the post-condition set.
* The newly formed post condition is added to the post conditon set.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected ostcondition
*********************************************************************/

public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
```

```
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp;
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);  // get the
xpression
      else
      return false;
      WsExpression lhs1 = null;
      WsExpression lhs2 = null;
      WsExpression rhs1 = null;
      WsExpression rhs2 = null;
      WsObject tempparent = exp.getParent();
      if (exp instanceof WsAnd)
      {
            WsBinaryExpression and = (WsBinaryExpression) exp;
            WsExpression imp1 = and.getWsBinExpOp1();
            WsExpression imp2 = and.getWsBinExpOp2();
            if (imp1 instanceof WsImplication)
            {
                  WsBinaryExpression impli1 = (WsBinaryExpression)
imp1;
                  lhs1 = (WsExpression) impli1.getWsBinExpOp1();
                  WsExpression eq1 = impli1.getWsBinExpOp2();
                  if (eq1 instanceof WsEqual)
                  {
                        WsBinaryExpression equal1 =
(WsBinaryExpression) eq1;
                  rhs1 = equal1.getWsBinExpOp1();
                  }
            }

            if (imp2 instanceof WsImplication)
            {
            WsBinaryExpression impli2 = (WsBinaryExpression) imp2;
                  WsExpression nt = impli2.getWsBinExpOp1();
                  WsExpression eq2 = impli2.getWsBinExpOp2();
                  if (nt instanceof WsNot)
                  {
                        WsUnaryExpression not = (WsUnaryExpression) nt;
                              lhs2 = (WsExpression)
not.getWsUnaryExpOp();
                              if (eq2 instanceof WsEqual)
                        {
                              WsBinaryExpression equal2 =
(WsBinaryExpression) eq2;
                              rhs2 = equal2.getWsBinExpOp1();
                        }
                  }
            }
      }
```

```
      if ( lhs1.toString().equalsIgnoreCase( lhs2.toString() ) &&
rhs1.toString().equalsIgnoreCase( rhs2.toString() ) )
{
            P = lhs1;
            X = rhs1;
            parent = tempparent;
            return true;
      }
      return false;
}/* END OF APPLICABLE


/*********************************************************************
* Execute creates the new post-condition
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*********************************************************************/

public boolean execute(Object params)
{System.out.println("in exe ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsEqual newpc = new WsEqual((WsExpression) X, (WsExpression) P);
      P.setParent(newpc);
      X.setParent(newpc);
      newpc.setParent(parent);
      postconIS.add(newpc);
      postconIS.remove(index);
      P = null;
      X = null;
      parent = null;
      return true;
}
```

Transform 18: XformSwetha18.java

```
/*********************************************************************
* Source file: XformSwetha18.java
* Purpose:  Copies all the elements in an array into another array.
*********************************************************************/


/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-condition is in the required form.
```

```
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*****************************************************************/

public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;

      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);   // get the
expression
      else
      return false;

      // check if the selected postcondition is has equal operator in
it.
      if(exp instanceof WsClasses.WsEqual)
      {
            WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
            WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
            WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
            WsName arrayVarName1 = null, arrayVarName2 = null;
            if(oprnd1 instanceof WsTick == false)
                  return false;
            arrayVarName1  = (WsName) ((WsTick)oprnd1).getWsTickName();
            arrayVarName2 = (WsName) oprnd2;

            WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());
            WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
            WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
            Vector Decls = packageIs.getWsDecls();
            WsClasses.WsDeclaration temp;
            if(attributeIs1 != null)
            {
            //finding the type of this attribute from the declerations.
            for(int j=0; j<Decls.size(); j++)
            {
                  temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs1.getTypeName().equals(temp.getName()))
                  {
```

```
                  // once the attribute is found check to see
                  // that it has already been changed to arrayType.
                        if(temp instanceof WsArrayType == false)//
onfirms that the attribute is array type
                        {
                              return false;
                        }
                  }
            }
      }
      if(attributeIs2 != null)
      {
      //finding the type of this attribute from the declerations.
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(attributeIs2.getTypeName().equals(temp.getName()))
            {
            // once the attribute is found check to see
            // that it has already been changed to arrayType.
            if(temp instanceof WsArrayType)// confirms that the
ttribute is array type
            {
                  return true;
            }
      }
      }
      }
      }
      return false;
}/* END OF APPLICABLE


/*****************************************************************
* Execute transforms the selected post-condition into code statements.
* The selected post-condition is removed from the post-condition set.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*****************************************************************/

public boolean execute(Object params)
{System.out.println("in exe ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
      WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
```

```
      WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
      WsName arrayVarName1 = (WsName) ((WsTick)oprnd1).getWsTickName();
      WsName arrayVarName2 = (WsName) oprnd2;
      WsIdentifier forArray1 = null, forIndex1 = null, forArray2 =
null, forIndex2 = null;
      WsName tempPara = null;
      boolean para = false;
      WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());
      if (attributeIs1 == null)
      {
            Vector vec = subprgIs.getOutFormals();
            WsParameter tempP1 = (WsParameter) vec.get(0);
            if (tempP1.getName().equals(arrayVarName1.getName()))
            forArray1 = tempP1.getWsName();
            WsParameter tempP2 = (WsParameter) vec.get(1);
            if (tempP2.getName().equals(arrayVarName1.getName() +
"MAX"))
            forIndex1 = tempP2.getWsName();
            tempPara = tempP2.getWsParameterType();
            para = true;
      }

      WsClasses.WsAttribute indexattributeIs1 = null;
      if (attributeIs1 != null)
            indexattributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName() + "MAX");

      WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
      WsClasses.WsAttribute indexattributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName() + "MAX");
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp1, temp2;
      for(int j=0; j<Decls.size(); j++)
      {
            temp1 = (WsClasses.WsDeclaration) Decls.get(j);
            temp2 = (WsClasses.WsDeclaration) Decls.get(j);
            if((attributeIs1 != null) &&
attributeIs1.getTypeName().equals(temp1.getName()))
            forArray1 = temp1.getWsDeclName();
            if((attributeIs1 != null) &&
indexattributeIs1.getTypeName().equals(temp1.getName()))
            forIndex1 = temp1.getWsDeclName();
            if(attributeIs2.getTypeName().equals(temp2.getName()))
            forArray2 = temp2.getWsDeclName();
            if(indexattributeIs2.getTypeName().equals(temp2.getName()))
            forIndex2 = temp2.getWsDeclName();
      }
            // Create the Identifier Reference for the array
      WsIdentifierRef arrayRef1 = new
WsIdentifierRef(arrayVarName1.getName());
      arrayRef1.setWsIdentRefTo(forArray1);
      WsIdentifierRef arrayRef2 = new
WsIdentifierRef(arrayVarName2.getName());
```

```
      arrayRef2.setWsIdentRefTo(forArray2);
            // Create the Identifier Reference for the array last index
      WsIdentifierRef lastIndexRef1 = new
WsIdentifierRef(arrayVarName1.getName() + "MAX");
      lastIndexRef1.setWsIdentRefTo(forIndex1);
      WsIdentifierRef lastIndexRef2 = new
WsIdentifierRef(arrayVarName2.getName() + "MAX");
      lastIndexRef2.setWsIdentRefTo(forIndex2);
      WsIdentifierRef IndexRef = new
WsIdentifierRef(forIndex2.getWsIdentSymbol());
      IndexRef.setWsIdentRefTo(forIndex2);
      WsIdentifier ident = new WsIdentifier("i");
      WsIdentifierRef iRef = new WsIdentifierRef("i");
      iRef.setWsIdentRefTo(ident);
      WsDataObject iDataObj = new WsVariable();
      iDataObj.setWsDataObjectType(IndexRef);
      iDataObj.setWsDeclName(ident);
      WsAssignment ini = new WsAssignment(iRef, new
WsLiteralInteger(0));
      WsAddition addI = new WsAddition((WsExpression) iRef, new
WsLiteralInteger(1));
      WsAssignment inc = new WsAssignment(iRef, addI);
      WsIndexedComponent arr1 = new WsIndexedComponent();
      arr1.setWsIndxCompName(arrayRef1);
      arr1.setWsIndxCompIndex(iRef);
      WsIndexedComponent arr2 = new WsIndexedComponent();
      arr2.setWsIndxCompName(arrayRef2);
      arr2.setWsIndxCompIndex(iRef);
      WsAssignment copy = new WsAssignment(arr1, arr2);
      WsAssignment updateMAX = new WsAssignment(lastIndexRef1, iRef);
      //while statement
      WsIteration whileStatement = new WsIteration();
      whileStatement.setWsIterCondition(new WsLessThanOrEqual(iRef,
lastIndexRef2));
      whileStatement.addWsIterBody(copy);
      whileStatement.addWsIterBody(updateMAX);
      whileStatement.addWsIterBody(inc);
      subprgIs.addWsSubprogLocal(iDataObj);
      subprgIs.addWsSubprogBody(ini);
      subprgIs.addWsSubprogBody(whileStatement);
      postconIS.remove(index);
      return true;

}
```

Transform 19: XformSwetha19.java

```
/*********************************************************************
* Source file: XformSwetha19.java
* Purpose:  Transforms the post-condition that finds the difference of
* 2 sets into code statements.
*********************************************************************/

/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
```

```
* selected post-condition is in the required form.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
**********************************************************************/

public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;

      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);   // get the
expression
      else
            return false;
      // check if the selected postcondition is has equal operator in
it.
      if(exp instanceof WsClasses.WsEqual)
      {
            WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
            WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
            WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
            WsName arrayVarName1 = null, arrayVarName2 = null,
arrayVarName3 = null;
            if(oprnd1 instanceof WsTick == false)
                  return false;
                  arrayVarName1  = (WsName)
((WsTick)oprnd1).getWsTickName();
            if(oprnd2 instanceof WsSubtraction)
            {
                  WsBinaryExpression BinaryExp2 = (WsBinaryExpression)
oprnd2;
                  WsExpression op1 = BinaryExp2.getWsBinExpOp1();
                  WsExpression op2 = BinaryExp2.getWsBinExpOp2();
                  arrayVarName2 = (WsName) op1;
                  arrayVarName3 = (WsName) op2;
            }
            else
                  return false;
                  WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());
            WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
```

```
            WsClasses.WsAttribute attributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName());
            WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
            Vector Decls = packageIs.getWsDecls();
            WsClasses.WsDeclaration temp;
            if(attributeIs1 != null)
            {
                  //finding the type of this attribute from the
declerations.
                  for(int j=0; j<Decls.size(); j++)
                  {
                        temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs1.getTypeName().equals(temp.getName()))
                        {
                        // once the attribute is found check to see
                        // that it has already been changed to
arrayType.
                        if(temp instanceof WsArrayType == false)//
confirms that the attribute is array type
                        {
                              return false;
                        }
                  }
            }
      }
      if(attributeIs2 != null)
      {
      //finding the type of this attribute from the declerations.
      for(int j=0; j<Decls.size(); j++)
      {
            temp = (WsClasses.WsDeclaration) Decls.get(j);
            if(attributeIs2.getTypeName().equals(temp.getName()))
            {
            // once the attribute is found check to see
                              // that it has already been changed to
arrayType.
            if(temp instanceof WsArrayType == false)// confirms that
the attribute is array type
            {
                  return false;
            }
            }
      }
      }
      if(attributeIs3 != null)
      {
            //finding the type of this attribute from the declerations.
            for(int j=0; j<Decls.size(); j++)
            {
                  temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs3.getTypeName().equals(temp.getName()))
                  {
                  // once the attribute is found check to see
                  // that it has already been changed to arrayType.
                        if(temp instanceof WsArrayType)// confirms that
he attribute is array type
```

```
                        {
                              return true;
                        }
                  }
            }
      }
      }
      return false;
}/* END OF APPLICABLE

/*****************************************************************
* Execute transforms the selected post-condition into code statements.
* The selected post-condition is removed from the post-condition set.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*****************************************************************/

public boolean execute(Object params)
{System.out.println("in exe ........  ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
            WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
      WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
      WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
      WsName arrayVarName1 = (WsName) ((WsTick)oprnd1).getWsTickName();
      WsBinaryExpression BinaryExp2 = (WsBinaryExpression) oprnd2;
      WsExpression op1 = BinaryExp2.getWsBinExpOp1();
      WsExpression op2 = BinaryExp2.getWsBinExpOp2();
      WsName arrayVarName2 = (WsName) op1;
      WsName arrayVarName3 = (WsName) op2;
      WsIdentifier forArray1 = null, forIndex1 = null, forArray2 =
null, forIndex2 = null, forArray3 = null, forIndex3 = null;
      WsName tempPara = null;
      boolean para = false;
      WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());

      if (attributeIs1 == null)
      {
            Vector vec = subprgIs.getOutFormals();
            WsParameter tempP1 = (WsParameter) vec.get(0);
            if (tempP1.getName().equals(arrayVarName1.getName()))
            forArray1 = tempP1.getWsName();
            WsParameter tempP2 = (WsParameter) vec.get(1);
      if (tempP2.getName().equals(arrayVarName1.getName() + "MAX"))
            forIndex1 = tempP2.getWsName();
```

```
            tempPara = tempP2.getWsParameterType();
            para = true;
      }
            WsClasses.WsAttribute indexattributeIs1 = null;
      if (attributeIs1 != null)
                  indexattributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName() + "MAX");

      WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
            WsClasses.WsAttribute indexattributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName() + "MAX");
            WsClasses.WsAttribute attributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName());
      WsClasses.WsAttribute indexattributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName() + "MAX");
            WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp1, temp2, temp3;
      for(int j=0; j<Decls.size(); j++)
      {
            temp1 = (WsClasses.WsDeclaration) Decls.get(j);
            temp2 = (WsClasses.WsDeclaration) Decls.get(j);
            temp3 = (WsClasses.WsDeclaration) Decls.get(j);
            if((attributeIs1 != null) &&
attributeIs1.getTypeName().equals(temp1.getName()))
            forArray1 = temp1.getWsDeclName();
            if((attributeIs1 != null) &&
indexattributeIs1.getTypeName().equals(temp1.getName()))
            forIndex1 = temp1.getWsDeclName();
            if(attributeIs2.getTypeName().equals(temp2.getName()))
            forArray2 = temp2.getWsDeclName();
            if(indexattributeIs2.getTypeName().equals(temp2.getName()))
            forIndex2 = temp2.getWsDeclName();
            if(attributeIs3.getTypeName().equals(temp3.getName()))
            forArray3 = temp3.getWsDeclName();
            if(indexattributeIs3.getTypeName().equals(temp3.getName()))
            forIndex3 = temp3.getWsDeclName();
      }
      // Create the Identifier Reference for the array
      WsIdentifierRef arrayRef1 = new
      WsIdentifierRef(arrayVarName1.getName());
      arrayRef1.setWsIdentRefTo(forArray1);
      WsIdentifierRef arrayRef2 = new
WsIdentifierRef(arrayVarName2.getName());
      arrayRef2.setWsIdentRefTo(forArray2);
      WsIdentifierRef arrayRef3 = new
WsIdentifierRef(arrayVarName3.getName());
      arrayRef3.setWsIdentRefTo(forArray3);

      // Create the Identifier Reference for the array last index
      WsIdentifierRef lastIndexRef1 = new
WsIdentifierRef(arrayVarName1.getName() + "MAX");
      lastIndexRef1.setWsIdentRefTo(forIndex1);
      WsIdentifierRef lastIndexRef2 = new
WsIdentifierRef(arrayVarName2.getName() + "MAX");
```

```
      lastIndexRef2.setWsIdentRefTo(forIndex2);
      WsIdentifierRef lastIndexRef3 = new
WsIdentifierRef(arrayVarName3.getName() + "MAX");
      lastIndexRef3.setWsIdentRefTo(forIndex3);

      WsIdentifierRef IndexRef1 = null;
      if (para == false)
            IndexRef1 = new
WsIdentifierRef(forIndex1.getWsIdentSymbol());   //k
      else
            IndexRef1 = new WsIdentifierRef(tempPara.getName());
//k
            IndexRef1.setWsIdentRefTo(forIndex1);
            WsIdentifierRef IndexRef2 = new
WsIdentifierRef(forIndex2.getWsIdentSymbol());   //i
            IndexRef2.setWsIdentRefTo(forIndex2);
            WsIdentifierRef IndexRef3 = new
WsIdentifierRef(forIndex3.getWsIdentSymbol());   //j
            IndexRef3.setWsIdentRefTo(forIndex3);

            WsIdentifier Kident = new WsIdentifier("k");
            WsIdentifierRef kRef = new WsIdentifierRef("k");
            kRef.setWsIdentRefTo(Kident);
            WsDataObject kDataObj = new WsVariable();
            kDataObj.setWsDataObjectType(IndexRef1);
            kDataObj.setWsDeclName(Kident);
            WsAssignment iniK = new WsAssignment(kRef, new
WsLiteralInteger(0));
            System.out.println("iniK " + iniK);
            WsAddition addK = new WsAddition((WsExpression) kRef, new
WsLiteralInteger(1));
            WsAssignment incK = new WsAssignment(kRef, addK);

            WsIdentifier Iident = new WsIdentifier("i");
            WsIdentifierRef iRef = new WsIdentifierRef("i");
            iRef.setWsIdentRefTo(Iident);
            WsDataObject iDataObj = new WsVariable();
            iDataObj.setWsDataObjectType(IndexRef2);
            iDataObj.setWsDeclName(Iident);
            WsAssignment iniI = new WsAssignment(iRef, new
WsLiteralInteger(0));
            System.out.println("iniI " + iniI);
            WsAddition addI = new WsAddition((WsExpression) iRef, new
WsLiteralInteger(1));
            WsAssignment incI = new WsAssignment(iRef, addI);

            WsIdentifier Jident = new WsIdentifier("j");
            WsIdentifierRef jRef = new WsIdentifierRef("j");
            jRef.setWsIdentRefTo(Jident);
            WsDataObject jDataObj = new WsVariable();
            jDataObj.setWsDataObjectType(IndexRef3);
            jDataObj.setWsDeclName(Jident);
            WsAssignment iniJ = new WsAssignment(jRef, new
WsLiteralInteger(0));
            System.out.println("iniJ " + iniJ);
            WsAddition addJ = new WsAddition((WsExpression) jRef, new
WsLiteralInteger(1));
```

```
            WsAssignment incJ = new WsAssignment(jRef, addJ);

            WsIdentifier found = new WsIdentifier("found");
            WsIdentifierRef foundRef = new WsIdentifierRef("found");
            foundRef.setWsIdentRefTo(found);
            WsDataObject foundDataObj = new WsVariable();
            foundDataObj.setWsDataObjectType(new
WsIdentifierRef("boolean"));
            foundDataObj.setWsDeclName(found);

            WsIndexedComponent arr1 = new WsIndexedComponent();
            arr1.setWsIndxCompName(arrayRef1);
            arr1.setWsIndxCompIndex(kRef);
            WsIndexedComponent arr2 = new WsIndexedComponent();
            arr2.setWsIndxCompName(arrayRef2);
            arr2.setWsIndxCompIndex(iRef);
            WsIndexedComponent arr3 = new WsIndexedComponent();
            arr3.setWsIndxCompName(arrayRef3);
            arr3.setWsIndxCompIndex(jRef);

            WsAssignment copy1 = new WsAssignment(arr1, arr2);
            WsAssignment copy2 = new WsAssignment(arr1, arr3);
            WsAssignment iftrue = new WsAssignment(foundRef, new
WsIdentifierRef("true"));
            WsAssignment iffalse = new WsAssignment(foundRef, new
WsIdentifierRef("false"));
            WsAssignment updateMAX = new WsAssignment(lastIndexRef1,
iRef);

            WsSelection if1 = new WsSelection();
            if1.setWsSelCondition(new WsEqual(arr2, arr3));
            if1.addWsSelThenPart(iftrue);

            WsIteration whileStatement2 = new WsIteration();
            whileStatement2.setWsIterCondition(new
WsLessThanOrEqual(jRef, lastIndexRef3));
            whileStatement2.addWsIterBody(if1);
            whileStatement2.addWsIterBody(incJ);

            WsSelection if2 = new WsSelection();
            if2.setWsSelCondition(new WsEqual(foundRef, new
WsIdentifierRef("false")));
            if2.addWsSelThenPart(copy1);
            if2.addWsSelThenPart(incK);

            WsIteration whileStatement1 = new WsIteration();
            whileStatement1.setWsIterCondition(new
WsLessThanOrEqual(iRef, lastIndexRef2));
            whileStatement1.addWsIterBody(iniJ);
            whileStatement1.addWsIterBody(iffalse);
            whileStatement1.addWsIterBody(whileStatement2);
            whileStatement1.addWsIterBody(if2);
            whileStatement1.addWsIterBody(updateMAX);
            whileStatement1.addWsIterBody(incI);

            subprgIs.addWsSubprogLocal(iDataObj);
            subprgIs.addWsSubprogLocal(jDataObj);
```

```
            subprgIs.addWsSubprogLocal(kDataObj);
            subprgIs.addWsSubprogLocal(foundDataObj);
            subprgIs.addWsSubprogBody(iniI);
            subprgIs.addWsSubprogBody(iniK);
            subprgIs.addWsSubprogBody(whileStatement1);

            postconIS.remove(index);
            return true;

      }
```

Transform 20: XformSwetha20.java

```
/*********************************************************************
* Source file: XformSwetha20.java
* Purpose:  Transforms the post-condition that finds the union of 2 set
* into code statements.
*********************************************************************/

/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-condition is in the required form.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*********************************************************************/

public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);

      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
            // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);   // get the
expression
      else
            return false;

      // check if the selected postcondition is has equal operator in
it.
      if(exp instanceof WsClasses.WsEqual)
      {
            WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
            WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
            WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
```

```
            WsName arrayVarName1 = null, arrayVarName2 = null,
arrayVarName3 = null;

            if(oprnd1 instanceof WsTick == false)
                  return false;
            arrayVarName1  = (WsName) ((WsTick)oprnd1).getWsTickName();
            if(oprnd2 instanceof WsUnion)
            {
                  WsBinaryExpression BinaryExp2 = (WsBinaryExpression)
oprnd2;
                  WsExpression op1 = BinaryExp2.getWsBinExpOp1();
                  WsExpression op2 = BinaryExp2.getWsBinExpOp2();
                  arrayVarName2 = (WsName) op1;
                  arrayVarName3 = (WsName) op2;
            }
            else
                  return false;

            WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());
            WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
            WsClasses.WsAttribute attributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName());
            WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
            Vector Decls = packageIs.getWsDecls();
            WsClasses.WsDeclaration temp;

            if(attributeIs1 != null)
            {
            //finding the type of this attribute from the declerations.
            for(int j=0; j<Decls.size(); j++)
            {
                  temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs1.getTypeName().equals(temp.getName()))
                  {
                  // once the attribute is found check to see
                  // that it has already been changed to arrayType.
                  if(temp instanceof WsArrayType == false)// confirms
that the attribute is array type
                  {
                        return false;
                  }
            }
            }
      }
            if(attributeIs2 != null)
            {
            //finding the type of this attribute from the declerations.
            for(int j=0; j<Decls.size(); j++)
            {
                  temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs2.getTypeName().equals(temp.getName()))
                  {
                  // once the attribute is found check to see
                  // that it has already been changed to arrayType.
```

```
                  if(temp instanceof WsArrayType == false)// confirms
that the attribute is array type
                  {
                        return false;
                  }
            }
      }
      }
      if(attributeIs3 != null)
      {
      //finding the type of this attribute from the declerations.
      for(int j=0; j<Decls.size(); j++)
      {
      temp = (WsClasses.WsDeclaration) Decls.get(j);

      if(attributeIs3.getTypeName().equals(temp.getName()))
      {
      // once the attribute is found check to see
      // that it has already been changed to arrayType.
      if(temp instanceof WsArrayType)// confirms that the attribute is
array type
      {
            return true;
      }
      }
      }
      }
      }
      return false;
}/* END OF APPLICABLE


/*********************************************************************
* Execute transforms the selected post-condition into code statements.
* The selected post-condition is removed from the post-condition set.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*********************************************************************/

public boolean execute(Object params)
{System.out.println("in exe ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);
      WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
      WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
      WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
```

```
      WsName arrayVarName1 = (WsName) ((WsTick)oprnd1).getWsTickName();
      WsBinaryExpression BinaryExp2 = (WsBinaryExpression) oprnd2;
      WsExpression op1 = BinaryExp2.getWsBinExpOp1();
      WsExpression op2 = BinaryExp2.getWsBinExpOp2();
      WsName arrayVarName2 = (WsName) op1;
      WsName arrayVarName3 = (WsName) op2;
      WsIdentifier forArray1 = null, forIndex1 = null, forArray2 =
null, forIndex2 = null, forArray3 = null, forIndex3 = null;
      WsName tempPara = null;
      boolean para = false;
      WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());

      if (attributeIs1 == null)
      {
            Vector vec = subprgIs.getOutFormals();
            WsParameter tempP1 = (WsParameter) vec.get(0);
            if (tempP1.getName().equals(arrayVarName1.getName()))
            forArray1 = tempP1.getWsName();
            WsParameter tempP2 = (WsParameter) vec.get(1);
            if (tempP2.getName().equals(arrayVarName1.getName() +
MAX"))
            forIndex1 = tempP2.getWsName();
            tempPara = tempP2.getWsParameterType();
            para = true;
      }
      WsClasses.WsAttribute indexattributeIs1 = null;
      if (attributeIs1 != null)
            indexattributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName() + "MAX");
      WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
      WsClasses.WsAttribute indexattributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName() + "MAX");
      WsClasses.WsAttribute attributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName());
      WsClasses.WsAttribute indexattributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName() + "MAX");
      WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
      Vector Decls = packageIs.getWsDecls();
      WsClasses.WsDeclaration temp1, temp2, temp3;
      for(int j=0; j<Decls.size(); j++)
      {
      temp1 = (WsClasses.WsDeclaration) Decls.get(j);
      temp2 = (WsClasses.WsDeclaration) Decls.get(j);
      temp3 = (WsClasses.WsDeclaration) Decls.get(j);
      if((attributeIs1 != null) &&
attributeIs1.getTypeName().equals(temp1.getName()))
      forArray1 = temp1.getWsDeclName();
      if((attributeIs1 != null) &&
indexattributeIs1.getTypeName().equals(temp1.getName()))
      forIndex1 = temp1.getWsDeclName();
      if(attributeIs2.getTypeName().equals(temp2.getName()))
      forArray2 = temp2.getWsDeclName();
      if(indexattributeIs2.getTypeName().equals(temp2.getName()))
      forIndex2 = temp2.getWsDeclName();
```

```
      if(attributeIs3.getTypeName().equals(temp3.getName()))
      forArray3 = temp3.getWsDeclName();
      if(indexattributeIs3.getTypeName().equals(temp3.getName()))
      forIndex3 = temp3.getWsDeclName();
}
// Create the Identifier Reference for the array
WsIdentifierRef arrayRef1 = new
WsIdentifierRef(arrayVarName1.getName());
arrayRef1.setWsIdentRefTo(forArray1);
WsIdentifierRef arrayRef2 = new
WsIdentifierRef(arrayVarName2.getName());
arrayRef2.setWsIdentRefTo(forArray2);
      WsIdentifierRef arrayRef3 = new
WsIdentifierRef(arrayVarName3.getName());
arrayRef3.setWsIdentRefTo(forArray3);

// Create the Identifier Reference for the array last index
WsIdentifierRef lastIndexRef1 = new
WsIdentifierRef(arrayVarName1.getName() + "MAX");
lastIndexRef1.setWsIdentRefTo(forIndex1);
WsIdentifierRef lastIndexRef2 = new
WsIdentifierRef(arrayVarName2.getName() + "MAX");
lastIndexRef2.setWsIdentRefTo(forIndex2);
WsIdentifierRef lastIndexRef3 = new
WsIdentifierRef(arrayVarName3.getName() + "MAX");
lastIndexRef3.setWsIdentRefTo(forIndex3);
WsIdentifierRef IndexRef1 = null;
if (para == false)
      IndexRef1 = new WsIdentifierRef(forIndex1.getWsIdentSymbol());
      else
      IndexRef1 = new WsIdentifierRef(tempPara.getName());         //k
      IndexRef1.setWsIdentRefTo(forIndex1);
      WsIdentifierRef IndexRef2 = new
WsIdentifierRef(forIndex2.getWsIdentSymbol());   //i
IndexRef2.setWsIdentRefTo(forIndex2);
WsIdentifierRef IndexRef3 = new
WsIdentifierRef(forIndex3.getWsIdentSymbol());   //j
IndexRef3.setWsIdentRefTo(forIndex3);

WsIdentifier Kident = new WsIdentifier("k");
WsIdentifierRef kRef = new WsIdentifierRef("k");
kRef.setWsIdentRefTo(Kident);
WsDataObject kDataObj = new WsVariable();
kDataObj.setWsDataObjectType(IndexRef1);
kDataObj.setWsDeclName(Kident);
WsAssignment iniK = new WsAssignment(kRef, new WsLiteralInteger(0));
System.out.println("iniK " + iniK);
WsAddition addK = new WsAddition((WsExpression) kRef, new
WsLiteralInteger(1));
WsAssignment incK = new WsAssignment(kRef, addK);
WsIdentifier Iident = new WsIdentifier("i");
WsIdentifierRef iRef = new WsIdentifierRef("i");
iRef.setWsIdentRefTo(Iident);
WsDataObject iDataObj = new WsVariable();
iDataObj.setWsDataObjectType(IndexRef2);
iDataObj.setWsDeclName(Iident);
WsAssignment iniI = new WsAssignment(iRef, new WsLiteralInteger(0));
```

```
System.out.println("iniI " + iniI);
WsAddition addI = new WsAddition((WsExpression) iRef, new
WsLiteralInteger(1));
WsAssignment incI = new WsAssignment(iRef, addI);
WsIdentifier Jident = new WsIdentifier("j");
WsIdentifierRef jRef = new WsIdentifierRef("j");
jRef.setWsIdentRefTo(Jident);
WsDataObject jDataObj = new WsVariable();
jDataObj.setWsDataObjectType(IndexRef3);
jDataObj.setWsDeclName(Jident);
WsAssignment iniJ = new WsAssignment(jRef, new WsLiteralInteger(0));
System.out.println("iniJ " + iniJ);
WsAddition addJ = new WsAddition((WsExpression) jRef, new
WsLiteralInteger(1));
WsAssignment incJ = new WsAssignment(jRef, addJ);
WsIdentifier found = new WsIdentifier("found");
WsIdentifierRef foundRef = new WsIdentifierRef("found");
foundRef.setWsIdentRefTo(found);
WsDataObject foundDataObj = new WsVariable();
foundDataObj.setWsDataObjectType(new WsIdentifierRef("boolean"));
foundDataObj.setWsDeclName(found);
WsIndexedComponent arr1 = new WsIndexedComponent();
arr1.setWsIndxCompName(arrayRef1);
arr1.setWsIndxCompIndex(kRef);
WsIndexedComponent arr2 = new WsIndexedComponent();
arr2.setWsIndxCompName(arrayRef2);
arr2.setWsIndxCompIndex(iRef);
WsIndexedComponent arr3 = new WsIndexedComponent();
arr3.setWsIndxCompName(arrayRef3);
arr3.setWsIndxCompIndex(jRef);
WsAssignment copy1 = new WsAssignment(arr1, arr2);
WsAssignment copy2 = new WsAssignment(arr1, arr3);
WsAssignment iftrue = new WsAssignment(foundRef, new
WsIdentifierRef("true"));
WsAssignment iffalse = new WsAssignment(foundRef, new
WsIdentifierRef("false"));
WsAssignment updateMAX = new WsAssignment(lastIndexRef1, kRef);
//while statement
WsIteration whileStatement1 = new WsIteration();
whileStatement1.setWsIterCondition(new WsLessThanOrEqual(iRef,
lastIndexRef2));
whileStatement1.addWsIterBody(copy1);
whileStatement1.addWsIterBody(updateMAX);
whileStatement1.addWsIterBody(incI);
whileStatement1.addWsIterBody(incK);
WsSelection if1 = new WsSelection();
if1.setWsSelCondition(new WsEqual(arr3, arr2));
if1.addWsSelThenPart(iftrue);
WsIteration whileStatement3 = new WsIteration();
whileStatement3.setWsIterCondition(new WsLessThanOrEqual(iRef,
lastIndexRef2));
whileStatement3.addWsIterBody(if1);
whileStatement3.addWsIterBody(incI);
WsSelection if2 = new WsSelection();
if2.setWsSelCondition(new WsEqual(foundRef, new
WsIdentifierRef("false")));
if2.addWsSelThenPart(copy2);
```

```
if2.addWsSelThenPart(updateMAX);
if2.addWsSelThenPart(incK);
WsIteration whileStatement2 = new WsIteration();
whileStatement2.setWsIterCondition(new WsLessThanOrEqual(jRef,
lastIndexRef3));
whileStatement2.addWsIterBody(iffalse);
whileStatement2.addWsIterBody(iniI);
whileStatement2.addWsIterBody(whileStatement3);
whileStatement2.addWsIterBody(if2);
whileStatement2.addWsIterBody(incJ);
subprgIs.addWsSubprogLocal(iDataObj);
subprgIs.addWsSubprogLocal(jDataObj);
subprgIs.addWsSubprogLocal(kDataObj);
subprgIs.addWsSubprogLocal(foundDataObj);
subprgIs.addWsSubprogBody(iniI);
subprgIs.addWsSubprogBody(iniJ);
subprgIs.addWsSubprogBody(iniK);
subprgIs.addWsSubprogBody(whileStatement1);
subprgIs.addWsSubprogBody(whileStatement2);
postconIS.remove(index);
return true;
}
```

Transform 21: XformSwetha21.java

```
/*********************************************************************
* Source file: XformSwetha21.java
* Purpose:  Transforms the post-condition that finds the intersection
* of 2 sets into code statements.
*********************************************************************/


/*********************************************************************
* Applicable makes sure that the index is valid.Returns true if the
* selected post-condition is in the required form.
*
* Parameters: target is the target method object
*                   params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*********************************************************************/

public static boolean applicable(Object target, Object params)
{
      WsClasses.WsMethod methodIs = (WsClasses.WsMethod) target;
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);

      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
```

```
      // Checking to see if it is a valid index.
      if(index <= postconIS.size() && index >=0)
            exp = (WsExpression) postconIS.get(index);   // get the
xpression
      else
            return false;
            // check if the selected postcondition is has equal perator
in it.
            if(exp instanceof WsClasses.WsEqual)
            {
                  WsBinaryExpression BinaryExp = (WsBinaryExpression)
exp;
                  WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
                  WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
                  WsName arrayVarName1 = null, arrayVarName2 = null,
arrayVarName3 = null;

                  if(oprnd1 instanceof WsTick == false)
                        return false;

                  arrayVarName1  = (WsName)
((WsTick)oprnd1).getWsTickName();
                  if(oprnd2 instanceof WsIntersection)
                  {
                        WsBinaryExpression BinaryExp2 =
(WsBinaryExpression) oprnd2;
                        WsExpression op1 = BinaryExp2.getWsBinExpOp1();
                        WsExpression op2 = BinaryExp2.getWsBinExpOp2();
                        arrayVarName2 = (WsName) op1;
                        arrayVarName3 = (WsName) op2;
                  }
                  else
                        return false;

                  WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());
                  WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
                  WsClasses.WsAttribute attributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName());
                  WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
                  Vector Decls = packageIs.getWsDecls();
                  WsClasses.WsDeclaration temp;
                  if(attributeIs1 != null)
                  {
                        //finding the type of this attribute from the
declerations.
                  for(int j=0; j<Decls.size(); j++)
                  {
                        temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs1.getTypeName().equals(temp.getName()))
                              {
                              // once the attribute is found check to
see
```

```
                              // that it has already been changed to
arrayType.
                                    if(temp instanceof WsArrayType ==
false)// confirms that the attribute is array type
                                    {
                                          return false;
                                    }
                              }
                        }
                  }
                  if(attributeIs2 != null)
                  {
                        //finding the type of this attribute from the
declerations.
                        for(int j=0; j<Decls.size(); j++)
                        {
                              temp = (WsClasses.WsDeclaration)
Decls.get(j);
                  if(attributeIs2.getTypeName().equals(temp.getName()))
                              {
                        // once the attribute is found check to see
                  // that it has already been changed to arrayType.
                                    if(temp instanceof WsArrayType ==
false)// confirms that the attribute is array type
                                    {
                                    return false;
                              }
                        }
                  }
            }
                  if(attributeIs3 != null)
            {
                  //finding the type of this attribute from the
declerations.
                  for(int j=0; j<Decls.size(); j++)
                  {
                        temp = (WsClasses.WsDeclaration) Decls.get(j);
                  if(attributeIs3.getTypeName().equals(temp.getName()))
                  {
                        // once the attribute is found check to see
                        // that it has already been changed to
arrayType.                                    if(temp instanceof
WsArrayType)// confirms that the attribute is array type
                                    {
                                    return true;
                              }
                        }
                  }
            }
            }
            return false;
      }/* END OF APPLICABLE



/*****************************************************************
* Execute transforms the selected post-condition into code statements.
```

```
* The selected post-condition is removed from the post-condition set.
*
* Parameters: params refers to a Vector
*                   element 0 contains the WsClass object of the class
*             that contains this method.
*               element 1 contains the WsPackage (AST).
*             element 2 contains the index of the selected
* postcondition
*****************************************************************/

public boolean execute(Object params)
{System.out.println("in exe ");
      WsSubprogram subprgIs = methodIs.getWsMethodSubprogram();
      Vector postconIS = subprgIs.getWsPostConditionSet();
      Vector params1 = (Vector) params;
      WsClasses.WsClass classIs = (WsClasses.WsClass) params1.get(0);
      String pc = (String) params1.get(2);
      int index = Integer.parseInt(pc);
      WsExpression exp;
      exp = (WsExpression) postconIS.get(index);

      WsBinaryExpression BinaryExp = (WsBinaryExpression) exp;
      WsExpression oprnd1 = BinaryExp.getWsBinExpOp1();
      WsExpression oprnd2 = BinaryExp.getWsBinExpOp2();
      WsName arrayVarName1 = (WsName) ((WsTick)oprnd1).getWsTickName();
      WsBinaryExpression BinaryExp2 = (WsBinaryExpression) oprnd2;
      WsExpression op1 = BinaryExp2.getWsBinExpOp1();
      WsExpression op2 = BinaryExp2.getWsBinExpOp2();
      WsName arrayVarName2 = (WsName) op1;
      WsName arrayVarName3 = (WsName) op2;
      WsIdentifier forArray1 = null, forIndex1 = null, forArray2 =
null, forIndex2 = null, forArray3 = null, forIndex3 = null;
      WsName tempPara = null;
      boolean para = false;
      WsClasses.WsAttribute attributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName());

      if (attributeIs1 == null)
      {
            Vector vec = subprgIs.getOutFormals();
            WsParameter tempP1 = (WsParameter) vec.get(0);
            if (tempP1.getName().equals(arrayVarName1.getName()))
            forArray1 = tempP1.getWsName();
            WsParameter tempP2 = (WsParameter) vec.get(1);
            if (tempP2.getName().equals(arrayVarName1.getName() +
"MAX");
            forIndex1 = tempP2.getWsName();
            tempPara = tempP2.getWsParameterType();
            para = true;
      }
            WsClasses.WsAttribute indexattributeIs1 = null;
      if (attributeIs1 != null)
            indexattributeIs1 =
classIs.getWsClassDataComponent(arrayVarName1.getName() + "MAX");
            WsClasses.WsAttribute attributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName());
```

```
            WsClasses.WsAttribute indexattributeIs2 =
classIs.getWsClassDataComponent(arrayVarName2.getName() + "MAX");
            WsClasses.WsAttribute attributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName());
            WsClasses.WsAttribute indexattributeIs3 =
classIs.getWsClassDataComponent(arrayVarName3.getName() + "MAX");
            WsClasses.WsPackage packageIs = (WsClasses.WsPackage)
params1.get(1);
            Vector Decls = packageIs.getWsDecls();
            WsClasses.WsDeclaration temp1, temp2, temp3;
      for(int j=0; j<Decls.size(); j++)
            {
                  temp1 = (WsClasses.WsDeclaration) Decls.get(j);
                  temp2 = (WsClasses.WsDeclaration) Decls.get(j);
                  temp3 = (WsClasses.WsDeclaration) Decls.get(j);

                  if((attributeIs1 != null) &&
attributeIs1.getTypeName().equals(temp1.getName()))
                        forArray1 = temp1.getWsDeclName();
                  if((attributeIs1 != null) &&
indexattributeIs1.getTypeName().equals(temp1.getName()))
                        forIndex1 = temp1.getWsDeclName();
            if(attributeIs2.getTypeName().equals(temp2.getName()))
                        forArray2 = temp2.getWsDeclName();
            if(indexattributeIs2.getTypeName().equals(temp2.getName()))
                        forIndex2 = temp2.getWsDeclName();
            if(attributeIs3.getTypeName().equals(temp3.getName()))
                        forArray3 = temp3.getWsDeclName();
            if(indexattributeIs3.getTypeName().equals(temp3.getName()))
                        forIndex3 = temp3.getWsDeclName();
            }

            // Create the Identifier Reference for the array
            WsIdentifierRef arrayRef1 = new
WsIdentifierRef(arrayVarName1.getName());
            arrayRef1.setWsIdentRefTo(forArray1);
            WsIdentifierRef arrayRef2 = new
WsIdentifierRef(arrayVarName2.getName());
            arrayRef2.setWsIdentRefTo(forArray2);
            WsIdentifierRef arrayRef3 = new
WsIdentifierRef(arrayVarName3.getName());
            arrayRef3.setWsIdentRefTo(forArray3);

            // Create the Identifier Reference for the array last index
            WsIdentifierRef lastIndexRef1 = new
WsIdentifierRef(arrayVarName1.getName() + "MAX");
            lastIndexRef1.setWsIdentRefTo(forIndex1);
            WsIdentifierRef lastIndexRef2 = new
WsIdentifierRef(arrayVarName2.getName() + "MAX");
            lastIndexRef2.setWsIdentRefTo(forIndex2);
            WsIdentifierRef lastIndexRef3 = new
WsIdentifierRef(arrayVarName3.getName() + "MAX");
            lastIndexRef3.setWsIdentRefTo(forIndex3);

            WsIdentifierRef IndexRef1 = null;
            if (para == false)
```

```
                  IndexRef1 = new
WsIdentifierRef(forIndex1.getWsIdentSymbol());   //k
            else
                  IndexRef1 = new WsIdentifierRef(tempPara.getName());
//k
            IndexRef1.setWsIdentRefTo(forIndex1);
            WsIdentifierRef IndexRef2 = new
WsIdentifierRef(forIndex2.getWsIdentSymbol());   //i
            IndexRef2.setWsIdentRefTo(forIndex2);
            WsIdentifierRef IndexRef3 = new
WsIdentifierRef(forIndex3.getWsIdentSymbol());   //j
            IndexRef3.setWsIdentRefTo(forIndex3);
            WsIdentifier Kident = new WsIdentifier("k");
            WsIdentifierRef kRef = new WsIdentifierRef("k");
            kRef.setWsIdentRefTo(Kident);
            WsDataObject kDataObj = new WsVariable();
            kDataObj.setWsDataObjectType(IndexRef1);
            kDataObj.setWsDeclName(Kident);
            WsAssignment iniK = new WsAssignment(kRef, new
WsLiteralInteger(0));
            System.out.println("iniK " + iniK);
            WsAddition addK = new WsAddition((WsExpression) kRef, new
WsLiteralInteger(1));
            WsAssignment incK = new WsAssignment(kRef, addK);

            WsIdentifier Iident = new WsIdentifier("i");
            WsIdentifierRef iRef = new WsIdentifierRef("i");
            iRef.setWsIdentRefTo(Iident);
            WsDataObject iDataObj = new WsVariable();
            iDataObj.setWsDataObjectType(IndexRef2);
            iDataObj.setWsDeclName(Iident);
            WsAssignment iniI = new WsAssignment(iRef, new
WsLiteralInteger(0));
            System.out.println("iniI " + iniI);
            WsAddition addI = new WsAddition((WsExpression) iRef, new
WsLiteralInteger(1));
            WsAssignment incI = new WsAssignment(iRef, addI);
            WsIdentifier Jident = new WsIdentifier("j");
            WsIdentifierRef jRef = new WsIdentifierRef("j");
            jRef.setWsIdentRefTo(Jident);
            WsDataObject jDataObj = new WsVariable();
            jDataObj.setWsDataObjectType(IndexRef3);
            jDataObj.setWsDeclName(Jident);
            WsAssignment iniJ = new WsAssignment(jRef, new
WsLiteralInteger(0));
            System.out.println("iniJ " + iniJ);
            WsAddition addJ = new WsAddition((WsExpression) jRef, new
WsLiteralInteger(1));
            WsAssignment incJ = new WsAssignment(jRef, addJ);

            WsIdentifier found = new WsIdentifier("found");
            WsIdentifierRef foundRef = new WsIdentifierRef("found");
            foundRef.setWsIdentRefTo(found);
            WsDataObject foundDataObj = new WsVariable();
            foundDataObj.setWsDataObjectType(new
WsIdentifierRef("boolean"));
            foundDataObj.setWsDeclName(found);
```

```
            WsIndexedComponent arr1 = new WsIndexedComponent();
            arr1.setWsIndxCompName(arrayRef1);
            arr1.setWsIndxCompIndex(kRef);
            WsIndexedComponent arr2 = new WsIndexedComponent();
            arr2.setWsIndxCompName(arrayRef2);
            arr2.setWsIndxCompIndex(iRef);
            WsIndexedComponent arr3 = new WsIndexedComponent();
            arr3.setWsIndxCompName(arrayRef3);
            arr3.setWsIndxCompIndex(jRef);
      WsAssignment copy1 = new WsAssignment(arr1, arr2);
            WsAssignment iftrue = new WsAssignment(foundRef, new
WsIdentifierRef("true"));
            WsAssignment iffalse = new WsAssignment(foundRef, new
WsIdentifierRef("false"));
            WsAssignment updateMAX = new WsAssignment(lastIndexRef1,
kRef);

      WsSelection if1 = new WsSelection();
            if1.setWsSelCondition(new WsEqual(arr2, arr3));
            if1.addWsSelThenPart(iftrue);
            WsIteration whileStatement2 = new WsIteration();
            whileStatement2.setWsIterCondition(new
WsLessThanOrEqual(jRef, lastIndexRef3));
            whileStatement2.addWsIterBody(if1);
      whileStatement2.addWsIterBody(incJ);
            WsSelection if2 = new WsSelection();
            if2.setWsSelCondition(new WsEqual(foundRef, new
WsIdentifierRef("true")));
            if2.addWsSelThenPart(copy1);
            if2.addWsSelThenPart(updateMAX);
            if2.addWsSelThenPart(incK);

            //while statement
            WsIteration whileStatement1 = new WsIteration();
            whileStatement1.setWsIterCondition(new
WsLessThanOrEqual(iRef, lastIndexRef2));
            whileStatement1.addWsIterBody(iniJ);
            whileStatement1.addWsIterBody(iffalse);
            whileStatement1.addWsIterBody(whileStatement2);
            whileStatement1.addWsIterBody(if2);
            whileStatement1.addWsIterBody(incI);

            subprgIs.addWsSubprogLocal(iDataObj);
            subprgIs.addWsSubprogLocal(jDataObj);
            subprgIs.addWsSubprogLocal(kDataObj);
            subprgIs.addWsSubprogLocal(foundDataObj);
            subprgIs.addWsSubprogBody(iniI);
            subprgIs.addWsSubprogBody(iniK);
            subprgIs.addWsSubprogBody(whileStatement1);

            postconIS.remove(index);
            return true;

      }
```