BETA: How we create partitions based on B machine characteristics

Ernesto C. B. de Matos

Anamaria M. Moreira

August 3, 2015

Contents

Chapter 1

Mapping BETA's Input Space Partitioning

This document explains how the BETA approach creates test partitions based on B machine characteristics. It presents, for each kind of characteristic, how to create test partitions (or blocks) using both *Equivalence Classes* and *Boundary Value Analysis* techniques. The next subsections explain this process individually for each kind of characteristic.

1.1 *Belong* characteristics

The *belongs* characteristic is described using a B notation belong clause, such as:

$$expression_l \in expression_r$$

This characteristic states that the result of $expression_l$ is a member of the set described by $expression_r$. In some cases the B notation also uses belong clauses to define types of variables or parameters. For this particular cases we have to pay extra attention while creating test partitions since it is possible to create invalid B notation statements or scenarios which are troublesome to implement on concrete test cases. Due to this, the explanation for partitioning belong characteristics is divided in *Typing characteristics* and *Non-typing characteristics*.

1.1.1 Typing characteristics

There are eight kinds of typing characteristics which are expressed using a belong clause: *belong_abs_data*, *belong_bool_data*, *NATDATA*, *NAT1DATA*, *INTDATA*, *RANGEDATA*, *SETDATA* and *STRINGDATA*. Those are considered typing characteristics because they are used to infer the types of variables and parameters in the specification. The main difference between a typing belong and a non-typing belong is that on a typing belong the left member of the clause is always an identifier.

belong_abs_data

$$belong\_abs\_data := ID \in EXP \text{ where } EXP \text{ is an abstract set}$$

A *belong_abs_data* characteristic states that a variable or parameter belongs to an abstract set which was defined either on the machine's SET clause or as a machine's parameter. For this kind of characteristic the BETA approach will create only one block for both equivalence classes and boundary value analysis. There is no block for negative test data because it would be difficult to implement a test case which requires that an element is the type of the abstract set but does not belong to the given set. See Table 1.1.

E. C. and B. A. Blocks	Test Scenario	Block Type
belong_abs_data	*ID* belongs to the abstract set *EXP*	Positive

Table 1.1: Equivalence Classes and Boundary Values blocks for *belong_abs_data*

belong_bool_data

$$belong\_bool\_data := ID \in BOOL$$

A *belong_bool_data* characteristic states that a given variable or parameter is a boolean. For this kind of characteristic there is no block for negative test data since a clause stating that a variable is not a boolean would not be a valid statement for a B specification. It would also bring problems while translating a test in a concrete test implementation using a strongly typed programming language. Table 1.2 shows the blocks for both equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	Test Scenario	Block Type
belong_bool_data	*ID* is a boolean	Positive

Table 1.2: Equivalence Classes and Boundary Values blocks for *belong_bool_data*

belong_nat_data

$$belong\_nat\_data := ID \in NAT$$

A *belong_nat_data* characteristic states that a variable or parameter is a integer which belongs to the set of the natural numbers. This set includes the numbers between the range of 0 and MAXINT where MAXINT is a constant for the maximum integer supported by a B specification. For this kind of characteristic the approach creates two blocks for equivalence classes (see Table 1.3). A negative block for NAT set will return integers which do not belong to the set of natural numbers (e.g.: -2, -1). Six blocks for boundary value analysis will be created (see Table 1.4).

E. C. Blocks	**Test Scenario**	**Block Type**
$belong\_nat\_data$	ID belongs to the set of natural numbers	Positive
$not(belong\_nat\_data)$	ID does not belong to the set of natural numbers	Negative

Table 1.3: Equivalence Classes blocks for *belong_nat_data*

B. A. Blocks	**Test Scenario**	**Block Type**
$ID = -1$	ID is equal to the number right before the inferior limit of the range	Negative
$ID = 0$	ID is equal to the inferior limit of the range	Positive
$ID = 1$	ID is equal to the number right after the inferior limit of the range	Positive
$ID = MAXINT - 1$	ID is equal to the number right before the superior limit of the range	Positive
$ID = MAXINT$	ID is equal to the superior limit of the range	Positive
$ID = MAXINT + 1$ *	ID is equal to the number right after the superior limit of the range	Negative

Table 1.4: Boundary Values blocks for *belong_nat_data*

belong_nat1_data

$$belong\_nat1\_data := ID \in NAT1$$

A *belong_nat1_data* characteristic states that a variable or parameter is a integer which belongs to the set of the positive natural numbers. This set includes the numbers between the range of 1 and MAXINT where MAXINT is a constant for the maximum integer supported by a B specification. For this kind of characteristic the approach creates two blocks for equivalence classes (see Table 1.5). A negative block for NAT1 set will return integers which do not belong to the set of natural numbers different than zero (e.g.: -1, 0). Six blocks for boundary value analysis (see Table 1.6).

E. C. Blocks	**Test Scenario**	**Block Type**
$belong\_nat1\_data$	*ID* belongs to the set of positive natural numbers	Positive
$not(belong\_nat1\_data)$	*ID* does not belong to the set of positive natural numbers	Negative

Table 1.5: Equivalence Classes blocks for *belong_nat1_data*

B. A. Blocks	**Test Scenario**	**Block Type**
$ID = 0$	*ID* is equal to the number right before the inferior limit of the range	Negative
$ID = 1$	*ID* is equal to the inferior limit of the range	Positive
$ID = 2$	*ID* is equal to the number right after the inferior limit of the range	Positive
$ID = MAXINT - 1$	*ID* is equal to the number right before the superior limit of the range	Positive
$ID = MAXINT$	*ID* is equal to the superior limit of the range	Positive
$ID = MAXINT + 1$ *	*ID* is equal to the number right after the superior limit of the range	Negative

Table 1.6: Boundary Values blocks for *belong_nat1_data*

belong_int_data

$$belong\_int\_data := ID \in INT$$

A *belong_int_data* characteristic states that a variable or parameter is an integer. An integer belongs to the set of numbers in the range between MININT and MAXINT where MININT and MAXINT represent, respectively, the minimum and the maximum integer supported in a B specification. We do not generate negative blocks for equivalence classes or boundary value analysis (Table 1.7) because it would not be a valid statement for a B specification. Also, it would be difficult to implement such a test using strongly typed programming languages.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$belong\_int\_data$	*ID* is a integer	Positive

Table 1.7: Equivalence Classes and Boundary Values blocks for *belong_int_data*

belong_range_data

$$belong\_range\_data := ID \in X..Y$$

A *belong_range_data* characteristic states that a variable or parameter is an integer which belongs to the range of elements between X and Y (which are also integers). Table 1.8 shows the blocks generated for equivalence classes and Table 1.9 shows the blocks generated for boundary value analysis.

E. C. Blocks	**Test Scenario**	**Block Type**
$ID \in MININT..X-1$	*ID* is a number before the interval	Negative
$ID \in X..Y$	*ID* is a number inside the interval	Positive
$ID \in Y+1..MAXINT$	*ID* is a number after the interval	Negative

Table 1.8: Equivalence Classes blocks for *belong_range_data*

B. A. Blocks	**Test Scenario**	**Block Type**
$ID = X-1$	*ID* is equal to the number right before the inferior limit of the range	Negative
$ID = X$	*ID* is equal to the inferior limit of the range	Positive
$ID = X+1$	*ID* is equal to the number right after the inferior limit of the range	Positive
$ID = Y-1$	*ID* is equal to the number right before the superior limit of the range	Positive
$ID = Y$	*ID* is equal to the superior limit of the range	Positive
$ID = Y+1$	*ID* is equal to the number right after the superior limit of the range	Negative

Table 1.9: Boundary Values blocks for *belong_range_data*

belong_set_data

$$belong\_set\_data := ID \in ID$$

A *belong_set_data* characteristic states that a given variable or parameter belongs to a set which is represented in the clause by an identifier *ID*. Table 1.10 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$belong\_set\_data$	Values that belong to the set *ID*	Positive
$not(belong\_set\_data)$	Values that do not belong to the set *ID*	Negative

Table 1.10: Equivalence Classes and Boundary Values blocks for *belong_set_data*

belong_string_data

$$belong\_string\_data := ID \in STRING$$

A *belong_string_data* characteristic states that a variable or parameter is a String. The approach does not create negative test data blocks for equivalence classes or boundary value analysis since it would not be a valid statement for a B specification. Also, it would be difficult to implement such a test on strongly typed programming languages. Table 1.12 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
belong_string_data	*ID* is a STRING	Positive

Table 1.11: Equivalence Classes and Boundary Values blocks for *belong_string_data*

belong_to_enumerated_set

$$belong\_string\_data := ID \in \{term_1, term_2, ..., term_n\}$$

A *belong_to_enumerated_set* characteristic states that a variable or parameter belongs to an enumerated set. Table 1.12 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$belong\_to\_enumerated\_set$	*ID* belongs to the enumerated set	Positive
$not(belong\_to\_enumerated\_set)$	*ID* doest not belong to the enumerated set	Negative

Table 1.12: Equivalence Classes and Boundary Values blocks for *belong_to_enumerated_set*

belong_func_data

$$belong\_func\_data := SIMPLE\_SET --> SIMPLE\_SET$$
$$belong\_func\_data := SIMPLE\_SET +-> SIMPLE\_SET$$
$$belong\_func\_data := SIMPLE\_SET >+> SIMPLE\_SET$$
$$belong\_func\_data := SIMPLE\_SET -->> SIMPLE\_SET$$
$$belong\_func\_data := SIMPLE\_SET +->> SIMPLE\_SET$$
$$belong\_func\_data := SIMPLE\_SET >->> SIMPLE\_SET$$

where $SIMPLE\_SET$ is equal to NAT or $NAT1$ or INT or $BOOL$ or $X..Y$ or ID or $\{a, b, c\}$

A *belong_func_data* characteristic states that a variable or parameter is a function between a source and a target set which can be any kind of *SIMPLE_SET*. Table 1.13 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$belong\_func\_data$	*ID* is a function between the given source and target sets	Positive

Table 1.13: Equivalence Classes and Boundary Values blocks for *belong_func_data*

1.1.2 Non-typing characteristics

This section presents the remainder of the characteristics which use the belong clause but are not typing characteristics.

belong_characteristic

$$belong\_characteristic := EXP_l \in EXP_r$$

The *belong_characteristic* characteristic encompasses the remainder of the possible clauses using the belong operator. Table 1.14 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$belong\_characteristic$	EXP_l belongs to EXP_r	Positive
$not(belong\_characteristic)$	EXP_l does not belong to EXP_r	Negative

Table 1.14: Equivalence Classes and Boundary Values blocks for *belong_characteristic*

1.2 *Non-belongs* characteristics

The *non-belongs* characteristic is described using a B notation non-belong clause, such as:

$$expression_l \notin expression_r$$

This characteristic states that the result of the $expression_l$ is not a member of the set described by $expression_r$.

non_belong_characteristic

$$non\_belong\_characteristic := EXP_l \notin EXP_r$$

This characteristic has the same blocks for equivalence classes and boundary value analysis, see Table 1.15.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$non\_belong\_characteristic$	EXP_l does not belong to EXP_r	Positive
$not(non\_belong\_characteristic)$	EXP_l belongs to EXP_r	Negative

Table 1.15: Equivalence Classes and Boundary Values blocks for *non_belong_characteristic*

1.3 *Subset* characteristics

The *subset* characteristic is described using a B notation subset clause, such as:

$$expression_l \subseteq expression_r$$

This characteristic states that the result of $expression_l$ is a subset of the set described by $expression_r$. This type of characteristic can also be used to infer the types of variables or parameters in the specification.

1.3.1 Typing characteristics

subset_abs_data

$$subset\_abs\_data := ID \subseteq EXP \text{ where } EXP \text{ is an abstract set}$$

A *subset_abs_data* characteristic states that a variable or parameter is a subset of an abstract set which was defined either on the machine's SET clause or as a machine's parameter. For this kind of characteristic the BETA approach will create only one positive block for both equivalence classes and boundary value analysis, see Table 1.16.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
subset_abs_data	*ID* is a subset of the abstract set *EXP*	Positive

Table 1.16: Equivalence Classes and Boundary Values blocks *for subset_abs_data*

subset_bool_data

$$subset\_bool\_data := ID \subseteq BOOL$$

A *subset_bool_data* characteristic states that a given variable or parameter is a subset of the set of boolean values ({true, false}). For this kind of characteristic there is no block for negative test data since a clause stating that a variable is not a subset of the boolean values would not be a valid statement for a B specification. Table 1.17 shows the blocks for both equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
subset_bool_data	*ID* is a subset of the boolean values	Positive

Table 1.17: Equivalence Classes and Boundary Values blocks for *subset_bool_data*

subset_nat_data

$$subset\_nat\_data := ID \subseteq NAT$$

A *subset_nat_data* characteristic states that a variable or parameter is a subset of the natural numbers. This set includes numbers between the range of 0 and MAXINT where MAXINT is a constant for the maximum integer supported by a B specification. For this kind of characteristic the approach creates two blocks for both equivalence classes and boundary value analysis. In this case, a negative block for a *subset_nat_data* will return a set which contains integers that are not natural numbers, e.g.: {-1, 0, 1}. See Table 1.18.

E. C. Blocks	Test Scenario	Block Type
subset_nat_data	*ID* is a subset of the natural numbers	Positive
not(*subset_nat_data*)	*ID* is not a subset of the natural numbers	Negative

Table 1.18: Equivalence Classes and Boundary Values blocks for *subset_nat_data*

subset_nat1_data

$$subset\_nat1\_data := ID \subseteq NAT1$$

A *subset_nat1_data* characteristic states that a variable or parameter is a subset of the set of the positive natural numbers. This set includes numbers between the range of 1 and MAXINT where MAXINT is a constant for the maximum integer supported by a B specification. For this kind of characteristic the approach creates two blocks for both equivalence classes and boundary value analysis, see Table 1.19.

E. C. Blocks	Test Scenario	Block Type
*subset_nat*1*_data*	*ID* is a subset of the set of the positive natural numbers	Positive
not(*subset_nat*1*_data*)	*ID* is not a subset of the set of the positive natural numbers	Negative

Table 1.19: Equivalence Classes and Boundary Values blocks for *subset_nat1_data*

subset_int_data

$$subset\_int\_data := ID \subseteq INT$$

A *subset_int_data* characteristic states that a variable or parameter is a subset of the set of integers. The set of integers contains numbers in the range between MININT and MAXINT where MININT and MAXINT represent, respectively, the minimum and the maximum integer supported in a B specification. We do not generate negative blocks for equivalence classes or boundary value analysis because it would not be a valid statement for a B specification, see Table 1.20.

E. C. and B. A. Blocks	Test Scenario	Block Type
subset_int_data	*ID* is a subset of the integers	Positive

Table 1.20: Equivalence Classes and Boundary Values blocks for *subset_int_data*

subset_range_data

$$subset\_range\_data := ID \subseteq X..Y$$

A *subset_range_data* characteristic states that a variable or parameter is a subset of integers ranging between the values X and Y. Table 1.21 shows the blocks generated for equivalence classes and boundary value analysis.

subset_set_data

$$subset\_set\_data := ID_l \subseteq ID_r$$

A *subset_set_data* characteristic states that a given variable or parameter is a subset of the set represented in the clause by an identifier *ID*. Table 1.22 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	Test Scenario	Block Type
subset_range_data	*ID* is a subset of integers which members belong to the range between X and Y	Positive
$not(subset\_range\_data)$	*ID* is a subset of integers which not all members belong to the range between X and Y	Negative

Table 1.21: Equivalence Classes and Boundary Values blocks for *subset_range_data*

E. C. and B. A. Blocks	Test Scenario	Block Type
subset_set_data	ID_l is a subset of the set ID_r	Positive
$not(subset\_set\_data)$	ID_l is not a subset of the set ID_r	Negative

Table 1.22: Equivalence Classes and Boundary Values blocks for *subset_set_data*

1.3.2 Non-typing characteristics

This section presents the remainder of the characteristics which use the strict subset clause but are not typing characteristics.

subset_characteristic

$$subset\_characteristic := EXP_l \subseteq EXP_r$$

The *subset_characteristic* characteristic encompasses the remainder of the possible clauses using the subset operator. Table 1.23 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	Test Scenario	Block Type
subset_characteristic	EXP_l is a subset of the set described by EXP_r	Positive
$not(subset\_characteristic)$	EXP_l is not a subset of the set described by EXP_r	Negative

Table 1.23: Equivalence Classes and Boundary Values blocks for *subset_characteristic*

1.4 *Non-subset* characteristics

The *non-subset* characteristic is described using a B notation non-subset clause, such as:

$$expression_l \not\subseteq expression_r$$

This characteristic states that the result of the $expression_l$ is not a subset of the set described by $expression_r$.

non_subset_characteristic

$$non\_subset\_characteristic := EXP_l \not\subseteq EXP_r$$

This characteristic has the same blocks for equivalence classes and boundary value analysis, see Table 1.24.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$non\_subset\_characteristic$	EXP_l is not a subset of EXP_r	Positive
$not(non\_subset\_characteristic)$	EXP_l is a subset of EXP_r	Negative

Table 1.24: Equivalence Classes and Boundary Values blocks for *non_subset_characteristic*

1.5 *Strict Subset* characteristics

The *strict subset* characteristic is described using a B notation strict subset clause, such as:

$$expression_l \subset expression_r$$

This characteristic states that the result of $expression_l$ is a subset of the set described by $expression_r$ but is not equal to it. This type of characteristic can also be used to infer the types of variables or parameters in the specification.

1.5.1 Typing characteristics

strict_subset_abs_data

$$strict\_subset\_abs\_data := ID \subset EXP \text{ where } EXP \text{ is an abstract set}$$

A *strict_subset_abs_data* characteristic states that a variable or parameter is a subset of an abstract set, but is not equal to it, which was defined either on the machine's SET clause or as a machine's parameter. For this kind of characteristic two blocks are created, one for positive data and another for negative data, for both equivalence classes and boundary value analysis, see Table 1.25.

E. C. and B. A. Blocks	Test Scenario	Block Type
$strict\_subset\_abs\_data$	ID is a strict subset of the abstract set EXP	Positive
$not(strict\_subset\_abs\_data)$	ID is not a strict subset of the abstract set EXP	Negative

Table 1.25: Equivalence Classes and Boundary Values blocks for *strict_subset_abs_data*

strict_subset_bool_data

$$strict\_subset\_bool\_data := ID \subset BOOL$$

A *strict_subset_bool_data* characteristic states that a given variable or parameter is a strict subset of the set of boolean values ({true, false}). For this kind of characteristic there is only one block for positive test data. Table 1.26 shows the blocks for both equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	Test Scenario	Block Type
$strict\_subset\_bool\_data$	ID is a strict subset of the boolean values	Positive

Table 1.26: Equivalence Classes and Boundary Values blocks for *strict_subset_bool_data*

strict_subset_nat_data

$$strict\_subset\_nat\_data := ID \subset NAT$$

A *strict_subset_nat_data* characteristic states that a variable or parameter is a strict subset of the natural numbers. This set includes numbers between the range of 0 and MAXINT where MAXINT is a constant for the maximum integer supported by a B specification. For this kind of characteristic the approach creates two blocks for both equivalence classes and boundary value analysis, see Table 1.27.

E. C. Blocks	Test Scenario	Block Type
$strict\_subset\_nat\_data$	ID is a strict subset of the natural numbers	Positive
$not(strict\_subset\_nat\_data)$	ID is not a strict subset of the natural numbers	Negative

Table 1.27: Equivalence Classes and Boundary Values blocks for *strict_subset_nat_data*

strict_subset_nat1_data

$$strict\_subset\_nat1\_data := ID \subset NAT1$$

A *strict_subset_nat1_data* characteristic states that a variable or parameter is a strict subset of the set of the positive natural numbers. This set includes numbers between the range of 1 and MAXINT where MAXINT is a constant for the maximum integer supported by a B specification. For this kind of characteristic the approach creates two blocks for both equivalence classes and boundary value analysis, see Table 1.28.

E. C. Blocks	Test Scenario	Block Type
$strict\_subset\_nat1\_data$	ID is a strict subset of the set of the positive natural numbers	Positive
$not(strict\_subset\_nat1\_data)$	ID is not a strict subset of the set of the positive natural numbers	Negative

Table 1.28: Equivalence Classes and Boundary Values blocks for *strict_subset_nat1_data*

strict_subset_int_data

$$strict\_subset\_int\_data := ID \subset INT$$

A *strict_subset_int_data* characteristic states that a variable or parameter is a strict subset of the set of integers. The set of integers contains numbers in the range between MININT and MAXINT where MININT and MAXINT represent, respectively, the minimum and the maximum integer supported in a B specification. BETA generates only one positive block for both equivalence classes and boundary values analysis, see Table 1.29.

E. C. and B. A. Blocks	Test Scenario	Block Type
$strict\_subset\_int\_data$	ID is a strict subset of the integers	Positive

Table 1.29: Equivalence Classes and Boundary Values blocks for *strict_subset_int_data*

strict_subset_range_data

$$strict\_subset\_range\_data := ID \subset X..Y$$

A *strict_subset_range_data* characteristic states that a variable or parameter is a strict subset of integers ranging between the values X and Y. Table 1.30 shows the blocks generated for equivalence classes and boundary value analysis.

strict_subset_set_data

$$strict\_subset\_set\_data := ID_l \subset ID_r$$

E. C. and B. A. Blocks	Test Scenario	Block Type
$strict\_subset\_range\_data$	ID is a strict subset of integers which members belong to the range between X and Y	Positive
$not(strict\_subset\_range\_data)$	ID is a strict subset of integers which not all members belong to the range between X and Y	Negative

Table 1.30: Equivalence Classes and Boundary Values blocks for *strict_subset_range_data*

A *strict_subset_set_data* characteristic states that a given variable or parameter is a strict subset of the set represented in the clause by an identifier *ID*. Table 1.31 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	Test Scenario	Block Type
$strict\_subset\_set\_data$	ID_l is a strict subset of the set ID_r	Positive
$not(strict\_subset\_set\_data)$	ID_l is not a strict subset of the set ID_r	Negative

Table 1.31: Equivalence Classes and Boundary Values blocks for *strict_subset_set_data*

1.5.2 Non-typing characteristics

This section presents the remainder of the characteristics which use the strict subset clause but are not typing characteristics.

strict_subset_characteristic

$$strict\_subset\_characteristic := EXP_l \subset EXP_r$$

The *strict_subset_characteristic* characteristic encompasses the remainder of the possible clauses using the strict subset operator. Table 1.32 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	Test Scenario	Block Type
$strict\_subset\_characteristic$	EXP_l is a strict subset of the set described by EXP_r	Positive
$not(strict\_subset\_characteristic)$	EXP_l is not a strict subset of the set described by EXP_r	Negative

Table 1.32: Equivalence Classes and Boundary Values blocks for *strict_subset_characteristic*

1.6 *Non-strict subset* characteristics

The *non-strict subset* characteristic is described using a B notation non-strict subset clause, such as:

$$expression_l \not\subset expression_r$$

This characteristic states that the result of the $expression_l$ is not a strict subset of the set described by $expression_r$.

non_strict_subset_characteristic

$$non\_strict\_subset\_characteristic := EXP_l \not\subset EXP_r$$

This characteristic has the same blocks for equivalence classes and boundary value analysis, see Table 1.33.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$non\_strict\_subset\_characteristic$	EXP_l is not a strict subset of EXP_r	Positive
$not(non\_strict\_subset\_characteristic)$	EXP_l is a strict subset of EXP_r	Negative

Table 1.33: Equivalence Classes and Boundary Values blocks for *non_strict_subset_characteristic*

1.7 *Equal* characteristics

The *equal* characteristic is described using a B notation equals clause, such as:

$$expression_l = expression_r$$

This characteristic states that the result of $expression_l$ is equal to the result of the of $expression_r$. The equal clause is also used in the B notation to define types of variables and parameters. The explanation for partitioning *equal* characteristics is divided in *Typing characteristics* and *Non-typing characteristics*.

1.7.1 Typing characteristics

There are seven kinds of typing characteristics which are expressed using an equal clause: *equals_abs_data*, *equals_term*, *equals_array*, *equals_range*, *equals_nat*, *equals_nat1* and *equals_int*.

equals_abs_data

$$equals\_abs\_data := ID = EXP \text{ where } EXP \text{ is an abstract set}$$

An *equals_abs_data* characteristic states that a variable or parameter is equal to an abstract set which was defined either on the machine's SET clause or as a machine's parameter. For this kind of characteristic the BETA approach will create two blocks for both equivalence classes and boundary value analysis, see Table 1.34.

E. C. and B. A. Blocks	Test Scenario	Block Type
$equals\_abs\_data$	ID is equal to the abstract set EXP	Positive
$not(equals\_abs\_data)$	ID is not equal to the abstract set EXP	Negative

Table 1.34: Equivalence Classes and Boundary Values blocks for *equals_abs_data*

equals_term

$$equals\_term := ID = TERM$$

where $TERM$ is a literal integer, a literal boolean, a function call or an arithmetical expression

An *equals_term* characteristic states that a variable or parameter is equal to a particular term. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.35.

E. C. and B. A. Blocks	Test Scenario	Block Type
$equals\_term$	ID is equal to the term	Positive
$not(equals\_term)$	ID is not equal to the term	Negative

Table 1.35: Equivalence Classes and Boundary Values blocks for *equals_term*

equals_array

$$equals\_array := ID = \{(0|->TRUE),(1|->FALSE)\}$$
or
$$equals\_array := ID = SET * \{0\}$$

An *equals_array* characteristic states that a variable or parameter is equal to a particular array. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.36.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$equals\_array$	*ID* is equal to the array	Positive
$not(equals\_array)$	*ID* is not equal to the array	Negative

Table 1.36: Equivalence Classes and Boundary Values blocks for *equals_array*

equals_range

$$equals\_range := ID = X..Y$$

An *equals_range* characteristic states that a variable or parameter is equal to a set which the elements belong to a particular range of integers. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.37.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$equals\_range$	*ID* is equal to the set which elements are all the integers between X..Y	Positive
$not(equals\_range)$	*ID* is not equal to the set which elements are all the integers between X..Y	Negative

Table 1.37: Equivalence Classes and Boundary Values blocks for *equals_range*

equals_nat

$$equals\_nat := ID = NAT$$

An *equals_nat* characteristic states that a variable or parameter is equal to a set of the natural numbers. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.38.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$equals\_nat$	*ID* is equal to the set of the natural numbers	Positive
$not(equals\_nat)$	*ID* is not equal to the set of the natural numbers	Negative

Table 1.38: Equivalence Classes and Boundary Values blocks for *equals_nat*

equals_nat1

$$equals\_nat1 := ID = NAT1$$

An *equals_nat1* characteristic states that a variable or parameter is equal to a set of positive natural numbers. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.39.

E. C. and B. A. Blocks	Test Scenario	Block Type
$equals\_nat1$	ID is equal to the set of positive natural numbers	Positive
$not(equals\_nat1)$	ID is not equal to the set of positive natural numbers	Negative

Table 1.39: Equivalence Classes and Boundary Values blocks for *equals_nat1*

equals_int

$$equals\_int := ID = INT$$

An *equals_int* characteristic states that a variable or parameter is equal to a set of integers. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.40.

E. C. and B. A. Blocks	Test Scenario	Block Type
$equals\_int$	ID is equal to the set of integers	Positive
$not(equals\_int)$	ID is not equal to the set of integers	Negative

Table 1.40: Equivalence Classes and Boundary Values blocks for *equals_int*

1.7.2 Non-typing characteristics

This section presents the remainder of the characteristics which use the *equals* clause but are not typing characteristics.

equals_characteristic

$$equals\_characteristic := EXP_l = EXP_r$$

The *equals_characteristic* characteristic encompasses the remainder of the possible clauses using the *equals* operator. Table 1.41 shows the blocks for equivalence classes and boundary value analysis.

E. C. and B. A. Blocks	Test Scenario	Block Type
$equals\_characteristic$	EXP_l is equal to EXP_r	Positive
$not(equals\_characteristic)$	EXP_l is not equal to EXP_r	Negative

Table 1.41: Equivalence Classes and Boundary Values blocks for *equals_characteristic*

1.8 *Unequal* characteristics

The *unequal* characteristic is described using a B notation *unequal* clause, such as:

$$expression_l \neq expression_r$$

This characteristic states that the result of the $expression_l$ is not equal to $expression_r$.

unequal_characteristic

$$unequal\_characteristic := EXP_l \neq EXP_r$$

This characteristic has the same blocks for equivalence classes and boundary value analysis, see Table 1.42.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$unequal\_characteristic$	EXP_l is not equal to EXP_r	Positive
$not(unequal\_characteristic)$	EXP_l is equal to EXP_r	Negative

Table 1.42: Equivalence Classes and Boundary Values blocks for *unequal_characteristic*

1.9 Conjunction characteristics

A *conjunction* characteristic is written as follows:

$$Predicate_1 \wedge Predicate_2 \wedge ... \wedge Predicate_n$$

The BETA approach breaks the conjunction and deals with each predicate separately, generating blocks according to their types.

1.10 Disjunction characteristics

A *disjunction* characteristic is written as follows:

$$disjunction\_characteristic := Predicate_1 \vee Predicate_2 \vee ... \vee Predicate_n$$

As with conjunction characteristics, the BETA approach breaks the disjunction and deals with each predicate separately, generating blocks according to their types.

1.11 Negation characteristics

A *negation* characteristic is written as follows:

$$negation\_characteristic := \neg(Predicate)$$

A *negation* characteristic negates the predicate inside its brackets. If the given predicate results in TRUE, after the negation it will result in FALSE and vice versa. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.43.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$negation\_characteristic$	inverts the $Predicate$ result	Positive
$not(negation\_characteristic)$	the negation is canceled and we have the result of the actual $Predicate$	Negative

Table 1.43: Equivalence Classes and Boundary Values blocks for *negation*

1.12 Implication characteristics

A *implication* characteristic is written as follows:

$$implication\_characteristic := Predicate_1 \Rightarrow Predicate_2$$

An *implication* characteristic returns FALSE if and only if $Predicate_1$ is equal to TRUE and $Predicate_2$ is equal to FALSE. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.44.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$implication\_characteristic$	the result of the implication should be TRUE	Positive
$not(implication\_characteristic)$	the result of the implication should be FALSE	Negative

Table 1.44: Equivalence Classes and Boundary Values blocks for *implication*

1.13 Equivalence characteristics

A *equivalence* characteristic is written as follows:

$$equivalence\_characteristic := Predicate_1 \Leftrightarrow Predicate_2$$

An *equivalence* characteristic returns TRUE if and only if $Predicate_1$ returns the same boolean value as $Predicate_2$. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.45.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$equivalence\_characteristic$	the result of the equivalence should be TRUE	Positive
$not(equivalence\_characteristic)$	the result of the equivalence should be FALSE	Negative

Table 1.45: Equivalence Classes and Boundary Values blocks for *equivalence*

1.14 Universal predicate characteristics

An *universal predicate* characteristic is written as follows:

$$universal\_pred\_characteristic := \forall(var\_list).Predicate_1 \Rightarrow Predicate_2$$

An *universal predicate* characteristic states that for given list of variables, $Predicate_2$ holds if $Predicate_1$ is TRUE. For this kind of characteristic the BETA approach will create two blocks for both equivalence classes and boundary value analysis, see Table 1.46.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
universal_pred_characteristic	the result of the universal quantification should be TRUE	Positive
not(*universal_pred_characteristic*)	the result of the universal quantification should be FALSE	Negative

Table 1.46: Equivalence Classes and Boundary Values blocks for *universal predicate*

1.15 Existential predicate characteristics

An *existential predicate* characteristic is written as follows:

$$existential\_pred\_characteristic := \exists(var\_list).Predicate$$

An *existential predicate* characteristic states that for each variable of the given variable list there is a value that satisfies the given $Predicate$. For this kind of characteristic the BETA approach will create two blocks for both equivalence classes and boundary value analysis, see Table 1.47.

E. C. and B. A. Blocks	**Test Scenario**	**Block Type**
$existential\_pred\_characteristic$	the result of the existential quantification should be TRUE	Positive
$not(existential\_pred\_characteristic)$	the result of the existential quantification should be FALSE	Negative

Table 1.47: Equivalence Classes and Boundary Values blocks for *existential predicate*

1.16 Relational characteristics

1.16.1 less_than_or_equal

A *less_than_or_equal* characteristic is written as follows:

$$less\_than\_or\_equal := Expression_1 \leq Expression_2$$

A *less_than_or_equal* characteristic states that $Expression_1$ is less or equal than $Expression_2$. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.48.

E. C. and B. A. Blocks	Test Scenario	Block Type
$Expression_1 \leq Expression_2$	the result of $Expression_1$ is less or equal to $Expression_2$	Positive
$Expression_1 > Expression_2$	the result of $Expression_1$ is greater than $Expression_2$	Negative

Table 1.48: Equivalence Classes and Boundary Values blocks for *less_than_or_equal*

1.16.2 strictly_less

A *strictly_less* characteristic is written as follows:

$$strictly\_less := Expression_1 < Expression_2$$

A *strictly_less* characteristic states that $Expression_1$ is strictly less than $Expression_2$. For this kind of characteristic the BETA approach will create the same three blocks for both equivalence classes and boundary value analysis, see Table 1.49.

E. C. and B. A. Blocks	Test Scenario	Block Type
$Expression_1 < Expression_2$	the result of $Expression_1$ is strictly less than $Expression_2$	Positive
$Expression_1 > Expression_2$	the result of $Expression_1$ is strictly greater than $Expression_2$	Negative
$Expression_1 = Expression_2$	the result of $Expression_1$ is equal to $Expression_2$	Negative

Table 1.49: Equivalence Classes and Boundary Values blocks for *strictly_less*

1.16.3 greater_than_or_equal

A *greater_than_or_equal* characteristic is written as follows:

$$greater\_than\_or\_equal := Expression_1 \geq Expression_2$$

A *greater_than_or_equal* characteristic states that $Expression_1$ is greater or equal than $Expression_2$. For this kind of characteristic the BETA approach will create the same two blocks for both equivalence classes and boundary value analysis, see Table 1.50.

E. C. and B. A. Blocks	Test Scenario	Block Type
$Expression_1 \geq Expression_2$	the result of $Expression_1$ is greater or equal to $Expression_2$	Positive
$Expression_1 < Expression_2$	the result of $Expression_1$ is strictly less than $Expression_2$	Negative

Table 1.50: Equivalence Classes and Boundary Values blocks for *greater_than_or_equal*

1.16.4 strictly_greater

A *strictly_greater* characteristic is written as follows:

$$strictly\_greater := Expression_1 > Expression_2$$

A *strictly_greater* characteristic states that $Expression_1$ is strictly greater than $Expression_2$. For this kind of characteristic the BETA approach will create the same three blocks for both equivalence classes and boundary value analysis, see Table 1.51.

E. C. and B. A. Blocks	Test Scenario	Block Type
$Expression_1 > Expression_2$	the result of $Expression_1$ is strictly greater than $Expression_2$	Positive
$Expression_1 < Expression_2$	the result of $Expression_1$ is strictly less than $Expression_2$	Negative
$Expression_1 = Expression_2$	the result of $Expression_1$ is equal to $Expression_2$	Negative

Table 1.51: Equivalence Classes and Boundary Values blocks for *strictly_greater*