Abstract Mutation Operators for Ecore-based Models

This document contains a full list of *Abstract Mutation Operators* for Ecore-based models. In Ecore, an EClass object feature can be either EAttribute or EReference and also can be either single-valued or multi-valued. While considering these special characteristics of features, we have defined three main abstract mutation operators. This paper presents those operators in details.

1 EAttribute - Single-valued (AMO:single-attr)

The abstract mutation operators for attributes that are single-valued can take certain mutation operators based on the datatype of feature. This section presents each datatype defined in Ecore along with its possible mutation operators.

1.1 String and Characters Data-types

1.1.1 EChar and ECharacterObject

ADD(EChar/ECharacterObject subject, EChar/ECharacterObject toAssign): Assign the value of *toAssign* to the *subject*.
Preconditions

- $subject.isUndefined()$ & $toAssign.isDefined()$
  Ensure that *subject* is not defined, only allowing new assignment to it. However, *toAssign* must be valid and not *null*.

DEL(EChar/ECharacterObject subject): Remove the value of *subject*.
Preconditions

- $subject.isDefined()$
  Check that *subject* is defined to be able to remove its value.

REP(EChar/ECharacterObject subject, EChar/ECharacterObject newValue): Replace the value of *subject* with the value of *newValue*.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verity that both *subject* and $newValue$ are valid.
- $subject \neq newValue$
  Ensure that the values of *subject* and $newValue$ are not equal. Otherwise the mutation is useless (i.e. equivalent).

1.1.2 EString

ADD(EString subject, EString toAdd): Append to the value of *subject* the value specified by *toAdd*.
Preconditions

- $subject.isDefined()$ & $toAdd.isDefined()$
  Check that both *subject* and *toAdd* are valid and defined.

- $toAdd.length \geq 1$
  For the changes to take place (or mutation), it is essential to ensure that this operator modifies existing value (i.e. *subject*) with at least one character.

DEL(EString subject, Integer toRemove): Remove randomly a number of *toRemove* characters from *subject*.
Preconditions

- $subject.isDefined()$ & $toRemove.isDefined()$
  Verify that both values represented by *subject* and *toRemove* are valid and defined.
- $subject.length \geq toRemove \geq 1$
  Ensure that the value represented by *toRemove*, which is the number of characters to be removed from *subject*, is less than or equal to the entire string size of *subject* and greater than 0.

REP(EString subject, EString newValue): Replace the value of *subject* with the value of *newValue*.
Preconditions

- $newValue.isDefined()$ & $subject \neq newValue$
  Check that *newValue* is defined (i.e. not *null*) and its value is equal to *subject* in order to generate a valid mutation.

1.2 Boolean Data-types

1.3 EBoolean and EBooleanObject

ADD(EBoolean/EBooleanObject subject, EBoolean/EBooleanObject state): Assign the value of *state* to the *subject*.
Preconditions

- $subject.isUndefined()$ & $state.isDefined()$
  Ensure that *subject* is not defined, only allowing new assignment to it. However, *state* must be valid and not *null*.

DEL(EBoolean/EBooleanObject subject): Remove the value of *subject*.
Preconditions

- $subject.isDefined()$
  Check that *subject* is defined to be able to remove its value.

REP(EBoolean/EBooleanObject subject): Negate the value of *subject*.
Preconditions

- $subject.isDefined()$
  Check that *subject* is defined so that its value can be negated.

1.4 Numeric Data-types

1.4.1 EInt and EIntegerObject

ADD(EInt/EIntegerObject subject, EInt/EIntegerObject toAdd): Add to the value of *subject* the value specified by *toAdd*.
Preconditions

- $subject.isDefined()$ & $toAdd.isDefined()$
  Make sure that both *subject* and *toAdd* are valid.
- $subject + toAdd \neq subject$
  Ensure that the value of *subject* changes after applying this mutation operator.

DEL(EInt/EIntegerObject subject, EInt/EIntegerObject toSubtract): Subtract from the value of *subject* the value of *toSubtract*.
Preconditions

- $subject.isDefined()$ & $toSubtract.isDefined()$
  Verify that *subject* and *toSubtract* are both valid.
- $subject - toSubtract \neq subject$
  Ensure that the value of *subject* is modified by subtracting *toSubtract* from its value.

REP(EInt/EIntegerObject subject, EInt/EIntegerObject newValue): Replace the value of *subject* with the value of *newValue*.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verify that *subject* and $newValue$ are both valid instances.
- $subject \neq newValue$
  Check that *subject* and $newValue$ have different values. This would prevent the generation of equivalent mutation.

1.4.2 EDouble and EDoubleObject

ADD(EDouble/EDoubleObject subject, EDouble/EDoubleObject toAdd): Add to the value of *subject* the value specified by *toAdd*.
Preconditions

- $subject.isDefined()$ & $toAdd.isDefined()$
  Check that both *subject* and *toAdd* are valid.
- $subject + toAdd \neq subject$
  Ensure that the value of *subject* is changeable by adding its value to *toAdd*

DEL(EDouble/EDoubleObject subject, EDouble/EDoubleObject toSubtract): Subtract from the value of *subject* the value of *toSubtract*.
Preconditions

- $subject.isDefined()$ & $toSubtract.isDefined()$
  Verify that *subject* and *toSubtract* are both valid.
- $subject - toSubtract \neq subject$
  Ensure that the value of *subject* is modified by subtracting *toSubtract* from its value.

REP(EDouble/EDoubleObject subject, EDouble/EDoubleObject newValue): Replace the value of *subject* with the value of *newValue*.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verify that *subject* and $newValue$ are both valid instances.
- $subject \neq newValue$
  Check that *subject* and $newValue$ have different values.

1.4.3 EFloat and EFloatObject

ADD(EFloat/EFloatObject subject, EFloat/EFloatObject toAdd): Add to the value of *subject* the value specified by *toAdd*.
Preconditions

- $subject.isDefined()$ & $toAdd.isDefined()$
  Check that both *subject* and *toAdd* are valid.

- $subject + toAdd \neq subject$
  Ensure that the value of *subject* is changeable by adding its value to *toAdd*

DEL(EFloat/EFloatObject subject, EFloat/EFloatObject toSubtract): Subtract from the value of *subject* the value of *toSubtract*.
Preconditions

- $subject.isDefined()$ & $toSubtract.isDefined()$
  Verify that *subject* and *toSubtract* are both valid.
- $subject - toSubtract \neq subject$
  Ensure that the value of *subject* is modified by subtracting *toSubtract* from its value.

REP(EFloat/EFloatObject subject, EFloat/EFloatObject newValue): Replace the value of *subject* with the value of *newValue*.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verify that *subject* and *newValue* are both valid instances.
- $subject \neq newValue$
  Check that *subject* and *newValue* have different values.

1.4.4 ELong and ELongObject

ADD(ELong/ELongObject subject, ELong/ELongObject toAdd): Add to the value of *subject* the value specified by *toAdd*.
Preconditions

- $subject.isDefined()$ & $toAdd.isDefined()$
  Check that both *subject* and *toAdd* are valid.
- $subject + toAdd \neq subject$
  Ensure that the value of *subject* is changeable by adding its value to *toAdd*

DEL(ELong/ELongObject subject, ELong/ELongObject toSubtract): Subtract from the value of *subject* the value of *toSubtract*.
Preconditions

- $subject.isDefined()$ & $toSubtract.isDefined()$
  Verify that *subject* and *toSubtract* are both valid.
- $subject - toSubtract \neq subject$
  Ensure that the value of *subject* is modified by subtracting *toSubtract* from its value.

REP(ELong/ELongObject subject, ELong/ELongObject newValue): Replace the value of *subject* with the value of *newValue*.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verify that *subject* and *newValue* are both valid instances.
- $subject \neq newValue$
  Check that *subject* and *newValue* have different values.

1.4.5 EBigDecimal

ADD(EBigDecimal subject, EBigDecimal toAdd): Add to the value of *subject* the value specified by *toAdd*.
Preconditions

- $subject.isDefined()$ & $toAdd.isDefined()$
  Check that both *subject* and *toAdd* are valid.
- $subject + toAdd \neq subject$
  Ensure that the value of *subject* is changeable by adding its value to *toAdd*

DEL(EBigDecimal subject, EBigDecimal toSubtract): Subtract from the value of *subject* the value specified by *toSubtract*.
Preconditions

- $subject.isDefined()$ & $toSubtract.isDefined()$
  Verify that *subject* and *toSubtract* are both valid.
- $subject - toSubtract \neq subject$
  Ensure that the value of *subject* is modified by subtracting *toSubtract* from its value.

REP(EBigDecimal subject, EBigDecimal newValue): Replace the value of *subject* with the value of *newValue*. The value of *newValue* can be generated/selected randomly.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verify that *subject* and *newValue* are both valid instances.
- $subject \neq newValue$
  Check that *subject* and *newValue* have different values.

1.4.6 EBigInteger

ADD(EBigInteger subject, EBigInteger toAdd): Add to the value of *subject* the value specified by *toAdd*.
Preconditions

- $subject.isDefined()$ & $toAdd.isDefined()$
  Check that both *subject* and *toAdd* are valid.
- $subject + toAdd \neq subject$ Ensure that the value of *subject* is changeable by adding its value to *toAdd*

DEL(EBigDecimal subject, EBigInteger toSubtract): Subtract from the value of *subject* the value specified by *toSubtract*.
Preconditions

- $subject.isDefined()$ & $toSubtract.isDefined()$
  Verify that *subject* and *toSubtract* are both valid.
- $subject - toSubtract \neq subject$ Ensure that the value of *subject* is modified by subtracting *toSubtract* from its value.

REP(EBigInteger subject, EBigInteger newValue): Replace the value of *subject* with the value of *newValue*. The value of *newValue* can be generated/selected randomly.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verify that *subject* and *newValue* are both valid instances.
- $subject \neq newValue$
  heck that *subject* and *newValue* have different values.

1.5 Other Data-types

1.5.1 EDate

ADD(EDate subject, EDate toAssign): Assign the value of *toAssign* to the *subject*.
Preconditions

- $subject.isUndefined()$ & $toAssign.isDefined()$
  Ensure that *subject* is not defined, only allowing new assignment to it. However, *toAssign* must be valid and not *null*.

DEL(EDate subject): Remove the value of *subject*.
Preconditions

- $subject.isDefined()$

REP(EDate subject, EDate newValue): Replace the value of *subject* with the value of *newValue*.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Verity that both *subject* and *newValue* are valid.
- $subject \neq newValue$
  Ensure that the values of *subject* and *newValue* are not equal.

2 EReference - Single-valued (AMO:single-ref)

The abstract mutation operators for this type of features can be::

ADD(Type subject, Type extra): Assigns the value of *extra* to *subject*.
Preconditions

- $subject.isUndefined()$ & $extra.isDefined()$
  Ensure that *subject* is not defined and only allowing new assignment. However, *extra* must be valid.
- $extra.isKindOf(subject.getType())$
  Check whether *extra* is instance of $subject.getType()$ for valid assignment.

DEL(Type subject): Deletes the value of *subject* (i.e. disjoint this feature from associated value).
Preconditions

- $subject.isDefined()$ In order to disjoint this feature from associated value, its is necessary to be valid and defined.

REP(Type subject, Type newValue): Replaces the value of *subject* with the value of *newValue*.
Preconditions

- $subject.isDefined()$ & $newValue.isDefined()$
  Ensure that both *subject* and *newValue* are valid instances.
- $newValue.isKindOf(subject.getType())$
  Check whether *newValue* is a valid such that it has the same type or one of the subtypes of $subject.getType()$.

3 EFeature - Multi-valued (AMO:multi-feature)

The abstract mutation operators for features that are multi-valued (whether attributes or references) can be:

ADD(Type subjects, Integer index, Type extra) Inserts *extra* at the specific position in *subjects*.
Preconditions

- $subjects.isDefined()$ & $extra.isDefined()$
  Ensure that both *subjects* and *extra* are both valid.
- $NOT subjects.include(extra)$
  Check whether *extra* is already exist in the list of *subjects* because the addition operator is meant to add only a new instance to the list.
- $extra.isKindOf(subjects.getType())$
  Verify that *extra* is of the type or one of the subtypes of $subjects.getType()$
- $lowerBound \leq subjects.size() + 1 \leq upperBound$
  Check whether the feature *subjects* allow additional element by checking its size with lower and upper bounds.
- $lowerBound \leq index < subjects.size()$
  Check the position of insertion *index* is with range of indices.

DEL(Type subjects, Integer index): Deletes the element at the specific position in *subjects*.
Preconditions

- $subjects.isDefined()$ Check that *subjects* is valid and not *null*
- $lowerBound \leq index < subjects.size()$
  Ensure that *index* is within the list range of indices.
- $lowerBound \leq subjects.size - 1 \leq upperBound$
  Verify that the feature *subjects* allows the remove of one of its elements and check its size with lower and upper bound constraint.

REP(Type subjects, Integer index, Type newValue) Replaces the value at the specific position in *subjects* with *newValue*.
Preconditions

- $subjects.isDefined()$ & $newValue.isDefined()$
  Ensure that both *subjects* and *newValue* are defined and not *null*.
- $newValue.isKindOf(subjects.getType())$
  Check whether *newValue* is of the type or one of the subtypes of $subjects.getType()$
- $lowerBound \leq index < subjects.size()$
  Check that the replacement position *index* is within the range of indices.
- $subjects(index) \neq newValue$
  Verify that *newValue* and its features' values do not equal to the one that replacing with (i.e. $subjects(index)$). This would prevent the generation of equivalent mutation.