Decomposer

0.9

Generated by Doxygen 1.8.11

Contents

Chapter 1

Decomposer

Decomposer is a relational decomposition tool that uses the functional dependencies of the relation to identify normal form and generate the sub-relations.

It uses the either one of two algorithms - Functional Dependencies Preserving Lossless decomposition or the Non Functional Dependencies Preserving Lossless decomposition. The FD Preserving algorithm guarantees that the sub-relations are at least in third normal form and all the original FDs are preserved. The non FD preserving may not preserve all the original FDs but it guarantees that all the sub-relations are in BC Normal Form.

This is a command line tool built by using GCC the GNU compiler collection on Linux platform. Using command line interface, the user can provide the information about the functional dependencies and the attributes of the relations. The user then can perform decomposition of the relation or find out about analysis of the relation.

The 'relation' in this tool represents the entity which have the identifying name, a set of attributes and a set of functional dependencies among these attributes. The 'attribute' represents characteristics of the relational entity. The dependencies defines the deterministic relation between set of attributes.

For example, R(a,b,c,d,e) {a->b , ab->de} In above example the relation name is R, the set of attributes is {a,b,c,d,e} and functional dependencies are {a} -> {b} and {a,b} -> {d,e}

Along with decomposition the user can choose different operation to be performed on the relation which includes finding minimal cover for dependency set, finding the candidate key for the relation, testing normal form of the relation and getting the dependencies that violates the particular normal form.

A command line based interactive menu driven user interface will be used by the application to accept user input and perform corresponding action. The Makefile can be use to compile the source code create the executable. make run command can be use to compile and run the executable.

The testing for application is performed by using the cppunit library. The unit tests are performed on various class methods and global functions.

The application can be started by using make run command or the executable program Decomposer can be used to start the application. The Relation object will be initialized first by providing details about relation name, attribute set and dependency set. The user can modify or reinitialize the the Relation object as required. The user can selects the different operation to be performed on this object using menu choice. Note: It is assumed that the input relation object is in at least first normal form, meaning the application currently do not supports the multivalued attributes.

Author

Ashish D. Kharde

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Dependency Class Reference

The Dependency class that represents the functional dependency of the relation using the attribute set.

```
#include <dependency.h>
```

Public Member Functions

- ∼Dependency ()
- Dependency (const Dependency &orig)
- bool operator< (const Dependency &) const
- set_str getLhs () const
- set_str getRhs () const
- set_str getAttribs () const

Friends

- class **Relation**
- class **dependency_test**
- class **relation_test**
- ostream & operator<< (ostream &, const Dependency &)

    Insert Dependency object into stream.

5.1.1 Detailed Description

The Dependency class that represents the functional dependency of the relation using the attribute set.

The Dependency class contains the data member that represents the left-hand side and right-hand of the functional dependency. Both lhs and rhs are considered as the set of attributes. Attributes will be represented by the string representing its name. Note that this class have private constructor which restricts the creation of its object other than its friend member functions and class. The object of this class can be constructed in the Relation class methods. This is to ensure the fact that dependency belongs to a particular relation and to identify that unique relation.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Dependency::∼Dependency ( )

Destructor of the Dependency class.

The destructor clears the string objects from the both lhs and rhs data member attributes sets.

5.1.2.2 Dependency::Dependency (const **Dependency** & *orig*)

The copy constructor of the Dependency class.

Parameters

orig	is the Dependency object which to be copied to the newly constructed object

The copy constructor initializes the lhs and rhs data members of the class from the corresponding data members of the parameter object.

5.1.3 Member Function Documentation

5.1.3.1 set_str Dependency::getAttribs ( ) const

The getter method for retrieve the all the attribute from dependency.

Returns

A set of all unique attributes presents in both lhs attribute set and rhs attribute set of the current Dependency object.

5.1.3.2 set_str Dependency::getLhs ( ) const `[inline]`

The getter method for retrieve the left-hand side attribute set.

5.1.3.3 set_str Dependency::getRhs ( ) const `[inline]`

The getter method for retrieve the right-hand side attribute set.

5.1.3.4 bool Dependency::operator< (const **Dependency** & *right*) const

The overloaded less operator for testing inequality the dependency objects.

Parameters

right	parameter to represent the rhs dependency object to test less inequality.

Returns

true if the lhs dependency object is logical lesser than right dependency object, false otherwise.

The overloaded operator will test the lhs set size first to determine the result. If the both objects have same size of the lhs set then it will check whether both lhs sets are equla or not. If they are not equal then the static setstr_↩
compare::less function is used to determine the 'less' equality of the lhs set of both objects. In case of the equal lhs set, the same operation will be performed with rhs set of the both parameter objects.

The documentation for this class was generated from the following files:

- dependency.h
- dependency.cc

5.2 Relation Class Reference

The Relation class that represents the relation entity.

```
#include <relation.h>
```

Public Types

- enum Normal { _2NF, _3NF, _BCNF }

Public Member Functions

- Relation (const string &, const set_str &temp=∗(new set_str()), const set_dep &t2=∗(new set_dep()))
- Relation (const Relation &orig)
- virtual ∼Relation ()
- bool addAtributte (const string &)
- unsigned int addAtributtes (const set_str &)
- bool removeAtributte (const string &)
- bool removeDependency (const set_str &, const set_str &)
- bool removeDependency (const Dependency &)
- unsigned int addDependencies (const set_dep &, bool update=true)
- itr_dep addDependency (const set_str &, const set_str &, bool update=true)
- itr_dep findDepLHS (const set_str &) const
- bool isNormal (const Relation::Normal &) const
- bool isDepAttribPresent (const Dependency &) const
- bool isSuperkey (const set_str &) const
- bool isPartialkey (const set_str &) const
- bool isPrime (const set_str &) const
- set_str getClosure (const set_str &) const
- set_key getCandidatekey (void) const
- set_dep getViolation (const Relation::Normal &) const

- set_dep getMinimalCover (bool details=false) const
- set_rel decomposePreserving (bool details=false) const
- set_rel decomposeNotPreserving (bool details=false) const
- bool operator!= (const Relation &right) const
- bool operator== (const Relation &right) const
- bool operator<= (const Relation &right) const
- bool operator>= (const Relation &right) const
- bool operator> (const Relation &right) const
- bool operator< (const Relation &right) const
- void clearDependencies ()
- void clearAttributes ()
- const set_str & getAttributes () const
- const set_dep & getDependencies () const
- string getName () const
- void setName (const string name)
- itr_dep addDependency (const Dependency &, bool update=true)

Friends

- class **relation_test**
- ostream & operator<< (ostream &, const Relation &)

    Insert Relation object into stream.

5.2.1 Detailed Description

The Relation class that represents the relation entity.

The Relation class contains the data member that represents the name of the Relation, the attribute set of the relation and a set of the functional dependency. The relation name will be represented by the string data member name. The attribute set will be represented by data member attributes - an object of set of string (set_str). The functional dependency set will represented by the data member dependencies - an object of set of Dependency objects (set_dep). The class also contains a enumeration data type Normal to identify the different normal form. It provides variety of the public method interface to modify the relation object and perform different operation on it. The class provides private read only access to the overloaded output operator << for the output of the Relation object in the output stream.

5.2.2 Member Enumeration Documentation

5.2.2.1 enum **Relation::Normal**

The enumeration to identify different normal form.

Enumerator

_2NF	Represnts the Second Normal form
_3NF	Represnts the Thirde Normal form
_BCNF	Represnts the Boyce-Codd Normal form

5.2.3 Constructor & Destructor Documentation

5.2.3.1 Relation::Relation (const string & *str*, const **set_str** & *attribs* = *(new **set_str**()), const **set_dep** & *dep* = *(new **set_dep**()))

The parameterized Relation constructor with default values for attribute set and dependencies set.

Parameters

str	is the string representing the name of the Relation.
attribs	is the set_str object that represents the attribute set for the newly constructed relation.It have default value as empty set_str object so the constructor can be used with different no of arguments the relation object will have an empty attribute set, if the attrib parameter is not provided.
is	the set_dep object that represents the dependency set for the newly constructed relation.It have default value as empty set_dep object so the constructor can be used with different no of arguments and the relation object will have an empty dependency set, if the dep parameter is not provided.

5.2.3.2 Relation::Relation (const Relation & *orig*)

The copy constructor for the Relation class.

Parameters

orig	is the constant reference to the Relation object from which the data members used to initialize newly constructed object.

The copy constructor uses all the data member from the parameter orig and initialize the data members of new object.

5.2.3.3 Relation::∼Relation ( ) `[virtual]`

The destructor for the Relation class.

The destructor will clear all the name, attribute set and dependency set of the relation object.

5.2.4 Member Function Documentation

5.2.4.1 bool Relation::addAtributte (const string & *str*)

A method to add a single attribute to the attribute set.

Parameters

str	a constant string representing the single attribute to be added into the attribute set of the relation object..

Returns

true if the new attribute is inserted, false otherwise.

The parameter str will be inserted if it is no already present in the attribute set.

5.2.4.2 unsigned int Relation::addAtributtes (const set_str & *as*)

A method to add multiple attributes to the attribute set.

Parameters

as	is set_str object represent the attribute set to be added to the attribute set of the relation object.

Returns

true at least one attribute from parameter set is inserted in the relation object attribute set, false otherwise.

5.2.4.3 unsigned int Relation::addDependencies (const set_dep & *dep*, bool *update* = `true`)

A method to add multiple attributes to the attribute set.

Parameters

dep	set of dependencies to be added into the relation.
update	boolean parameter with default value false to indicate that whether to add the attribute if dependency contains the attribute which is not already present in the attribute set of the relation.

Foe every dependency object from the parameter dep, the method Relation::addDependency will be called.

5.2.4.4 itr_dep Relation::addDependency (const set_str & *lhs*, const set_str & *rhs*, bool *update* = `true`)

A method to add a single dependency with explicitly specified lhs and rhs to the dependency set.

Parameters

lhs	is string set representing the lhs side of the dependency.
rhs	is string set representing the rhs side of the dependency.
update	is boolean parameter with default value true. If the update value is true then all the attributes form lhs and rhs which are not part of the attribute set of the relation object will added to the attribute set first. No new attribute will be inserted into the relation attribute set if the update value is false.

Returns

If the dependency is added into the dependency set for the relation object, the iterator pointing to the newly added dependency object will be returned. If dependency is already present in the dependency set then the iterator pointing to the existing dependency object will be returned. If no dependency is added to the relation then the set::end for the dependency set will be returned.

All the the attributes from the rhs which are subset of lhs will be removed first and if the rhs is empty then the dependency will not be added to the relation. If the rhs is not empty then the new Dependency object will be created with lhs and modified rhs and the private method Relation::addDependency will be used with the update parameter value to insert the dependency into the dependency set. In this case the return value from Relation::addDependency will be returned back.

5.2.4.5 itr_dep Relation::addDependency (const Dependency & *dep*, bool *update* = `true`)

A method to add single dependency object to the dependency set.

Parameters

dep	A dependency object to be inserted into the dependency set of the relation.
update	is boolean parameter with default value true. If the update value is true then all the attributes form lhs and rhs which are not subset of attribute set of the relation, will added to the attribute set first. No new attribute will be inserted into the relation attribute set if the update value is false and the lhs and rhs contains some attributes which are not subset of relation attribute set.

Returns

If the dependency is added into the dependency set for the relation object, the iterator pointing to the newly added dependency object will be returned. If dependency is already present in the dependency set then the iterator pointing to the existing dependency object will be returned. If no dependency is added to the relation then the set::end for the dependency set will be returned.

If lhs and rhs are subset of the attribute set of the relation, then the dependency will added to the set. If the update option is ture and lhs or rhs are not subset of the attribute set then all the attributes from both lhs and rhs will be inserted into the relation attribute set first, then the dependency will be added. If update is false and the lhs is not subset of the attribute set of relation then no dependency will be added. If update is false and only rhs is not subset of relation attribute set then the only those attributes from the rhs which are part of the attribute set will be considered and all other attributes will be discarded from rhs and then the dependency will be inserted into the dependency set. Note that if dependency is inserted into the dependency set then private method Relation::reduceDependecy will be used to minimize the dependency set with same lhs values and the resultant position of the newly inserted dependency will be returned.

5.2.4.6 void Relation::clearAttributes ( )

A method to clear the attribute set of the relation.

This method will clear all the attribute set and dependency set of the relation object.

5.2.4.7 void Relation::clearDependencies ( )

A method to clear the dependency set of the relation.

This method will clear all the dependency set of the relation object.

5.2.4.8 set_rel Relation::decomposeNotPreserving (bool *details* = `false`) const

A method to get the sub-relation set for the given relation object by using non FD preserving algorithm.

Parameters

details	a boolean parameter with default value false. If true the steps involved in the decomposition will be printed on standard output stream cout.

Returns

set_rel object representing the set of decomposed sub-relations.

The method will use the FD-preserving decomposition algorithm to decompose the relation into the sub-relations and returns the set of such relations. All the decomposed sub relation will be in BCNF but it will not guarantees that all the original dependencies are preserved from dependencies found in all sub-relations. It uses the private static method Relation::decompose to perform the operation using recursive method.

5.2.4.9 set_rel Relation::decomposePreserving (bool *details* = `false`) const

A method to get the sub-relation set for the given relation object by using FD preserving algorithm.

Parameters

details	a boolean parameter with default value false. If true the steps involved in the decomposition will be printed on standard output stream cout.

Returns

set_rel object representing the set of decomposed sub-relations.

The method will use the FD-preserving decomposition algorithm to decompose the relation into the sub-relations and returns the set of such relations. The decomposed sub relation will have all the dependencies which can preserve the original dependency set. It will also ensures that each sub-relation is in at least 3NF.

5.2.4.10 itr_dep Relation::findDepLHS (const set_str & *str*) const

Find dependency by specifying the lhs.

Parameters

str	a string set parameter represents the lhs value to search for.

Returns

The iterator position of the dependency object from the dependency set if the dependency is found with same lhs as the parameter. It returns the set::end iterator of the attribute set if the parameter is not equal to the any of dependency object's lhs form the dependency set of the relation.

5.2.4.11 const set_str& Relation::getAttributes ( ) const `[inline]`

A getter method to retrieve the attribute set of the relation.

Returns

The constant reference of the set_str object that represents the attribute set of the relation object.

5.2.4.12 set_key Relation::getCandidatekey (void) const

A method to get all the candidate key set for the relation.

Returns

set of keys object containing all the possible candidate keys for the relation.

The method will calculate all the possible candidate key for the relation object using current functional dependencies and the attributes. The single key can be considered as set of attributes which can derive all the attributes of the relation uisng the dependency set of the relation.

5.2.4.13 set_str Relation::getClosure (const set_str & *lhs*) const

A method to get the closure of the given attribute set using functional dependency set of the Relation.

Parameters

lhs	is the set_str object that represents the set of attributes on which closure operation is to be performed.

Returns

the set_str object containing all the attribute which are result of the closure operation. The method will find out all the possible attributes which can be derived by the parameter lhs using dependency set of the relation.

5.2.4.14 const set_dep& Relation::getDependencies ( ) const [inline]

A getter method to retrieve the dependency set of the relation.

Returns

The constant reference of the set_dep object that represents the dependency set of the relation object.

5.2.4.15 set_dep Relation::getMinimalCover (bool *details* = `false`) const

A method to get the minimal cover dependency set for the all dependencies that belongs to the relation.

Parameters

details	a boolean parameter with default value false. If true the steps involved in the finding minimal cover will be printed on standard output stream cout.

Returns

Returns the set of dependency representing the minimal form of current functional dependencies of the relation object.

The method will use three different steps reduce lhs, reduce rhs and reduce rules by using private methods of the relation to get the minimal cover.

5.2.4.16 string Relation::getName ( ) const [inline]

A getter method to retrieve the name of the relation.

Returns

The string representing the name of the relation. attribute set of the relation object.

5.2.4.17 set_dep Relation::getViolation (const Relation::Normal & *form*) const

A method to get dependency set that violates given normal form conditions.

Parameters

form	represents the normal form from one of the Relation::Normal value.

Returns

set of dependency from dependencies of the relation object which violates the condition for the given normal form indicated by the parameter form.

It uses the functor object of Violation as predicate to find out the return value.

5.2.4.18 bool Relation::isDepAttribPresent (const Dependency & *dep*) const

A method to check if all the attributes of the parameter dependency present in the attribute set of the relation object.

Parameters

dep	is a constant reference to the Dependency object for which the check is to be performed.

Returns

true if all the attributes form the dependency lhs and rhs set are subset of the attribute set of the relation object, false otherwise.

5.2.4.19 bool Relation::isNormal (const Relation::Normal & *form*) const

A method to test normal form of the current Relation object.

Parameters

form	is a Relation::Normal value represents the normal form for which the relation is to be tested.

Returns

true if the relation is in the normal form provided by the parameter, false otherwise.

The method will use the functor object of Violation class to find the any violation for the normal form for every dependency from dependency set. No violation means the relation is in given normal form.

5.2.4.20 bool Relation::isPartialkey (const set_str & *lhs*) const

A method to test if the given string set is the partial-key to the relation.

Parameters

lhs	represents the attribute set which is to find out is partial key or not.

Returns

true if the parameter lhs is the partial, false otherwise.

The lhs is considered as the partial key if it is subset of at least one candidate key from candidate key set.

5.2.4.21 bool Relation::isPrime (const set_str & *str*) const

A method to test if the given string set is belong to the prime attribute set of the relation.

Parameters

str	represents the attribute set which is to find out is prime or not.

Returns

true if the parameter str is the subset of prime attributes, false otherwise.

The attribute is considered as prime attribute if it belongs to any one of the candidate key. If parameter is sub set of all the prime attribute then true value is returned.

5.2.4.22 bool Relation::isSuperkey (const set_str & *lhs*) const

A method to test if the given string set is the super-key to the relation.

Parameters

lhs	represents the attribute set to find out is supekey or not.

Returns

true if the parameter lhs is the superkey, false otherwise.

The lhs is considered as the superkey if it is superset of at least one candidate key from candidate key set.

5.2.4.23 bool Relation::operator!= (const Relation & *right*) const

A overloaded relational operator to check inequality between two relation objects.

Parameters

right	the rhs relation object for the comparison operation.

Returns

true if current relation object is not equal to the parameter right, false otherwise.

This method reuses the Relation::operator !=.

5.2.4.24 bool Relation::operator< (const Relation & *right*) const

A overloaded relational operator to check less than or inequality between two relation objects.

Parameters

right	the rhs relation object for the comparison operation.

Returns

true if current relation object is logically lesser than the parameter right, false otherwise.

The relation will use the attribute set size and dependency set size to determine the logical lesser relation object. In case of equal attribute set and dependency set the name will be used to determine the result. In case of equal attribute size and but the not equal attribute set, the setstr_compare::less function will be used to compare the attribute sets. In case of the equal dependency set size but not equal dependency set, the overloaded Dependency::operator < will be used to determine the lesser dependency set between two relation objects.

5.2.4.25 bool Relation::operator<= (const Relation & *right*) const

A overloaded relational operator to check less than or equal to inequality between two relation objects.

Parameters

right	the rhs relation object for the comparison operation.

Returns

true if current relation object is logically lesser than or equal to the parameter right, false otherwise.

This method reuses the Relation::operator $<$ and Relation::operator =.

5.2.4.26 bool Relation::operator== (const Relation & *right*) const

A overloaded relational operator to check equality between two relation objects.

Parameters

right	the rhs relation object for the comparison operation.

Returns

true if both relation are equla, false otherwise.

The relation objects will be considered equal if the name of both relation is equal and the attribute set & dependency set are equal.

5.2.4.27 bool Relation::operator> (const Relation & *right*) const

A overloaded relational operator to check greater than or inequality between two relation objects.

Parameters

right	the rhs relation object for the comparison operation.

Returns

true if current relation object is logically lesser than the parameter right, false otherwise.

This method reuses the Relation::operator $<$.

5.2.4.28 bool Relation::operator>= (const Relation & *right*) const

A overloaded relational operator to check greater than or equal to inequality between two relation objects.

Parameters

right	the rhs relation object for the comparison operation.

Returns

true if current relation object is logically greater than or equal to the parameter right, false otherwise.

This method reuses the Relation::operator $>$ and Relation::operator =.

5.2.4.29 bool Relation::removeAtributte (const string & *str*)

A method to remove a single attribute from the attribute set.

Parameters

str	a constant string representing the single attribute to be removed from the attribute set of the relation object.

Returns

true if attribute is removed from the set, false otherwise.

If the attribute is not found in the attribute set of the relation object then no attribute will be removed. If it is found in the relation then the attribute will be removed. The dependency set will be searched to find out dependencies which contains the removed attribute. If the attribute present in lhs part of the dependency the entire dependency will be removed from dependency set. If the attribute is found only in rhs set of the dependency, then the attribute will be removed from rhs as well.

5.2.4.30 bool Relation::removeDependency (const set_str & *lhs,* const set_str & *rhs*)

A method to remove a single dependency from the dependency set by providing separate set_str to represent the custom lhs and rhs of the dependency.

Parameters

lhs	is string set representing the lhs side of the dependency.
rhs	is string set representing the rhs side of the dependency.

Returns

true if the dependency is found and removed from the dependency set false otherwise.

It creates the temporary Dependency object from lhs and rhs and then uses private method Relation::remove↩ Dependency to remove it from the dependency set.

5.2.4.31 bool Relation::removeDependency (const Dependency & *dep*)

A method to remove a single dependency from the dependency set by providing constant reference to the dependency object.

Parameters

dep	constant reference to the dependency object to be removed from the dependency set.

Returns

true if the dependency is found and removed, false otherwise.

This method will first finds the exact match for the dependency by finding the dependency object from the dependency set which has same lhs and rhs as parameter dep. If no such dependency is found, then the dependency with the same lhs will be searched. If found then the only the rhs part of parameter dep will be removed from the from the matching result. If no dependency is found to remove or modify then the false value will be returned.

5.2.4.32 void Relation::setName (const string *name*) [inline]

A setter method to set the name of the relation.

Parameters

name	a string parameter indicates the new name of the relation.

The documentation for this class was generated from the following files:

- relation.h
- relation.cc

5.3 setstr_compare Class Reference

The function object class for less-than inequality comparison of the set of string.

```
#include <declaration.h>
```

Public Member Functions

- bool operator() (const set_str &lhs, const set_str &rhs) const

Static Public Member Functions

- static bool less (const set_str &lhs, const set_str &rhs)

5.3.1 Detailed Description

The function object class for less-than inequality comparison of the set of string.

5.3.2 Member Function Documentation

5.3.2.1 bool setstr_compare::less (const set_str & *lhs*, const set_str & *rhs*) [static]

Performs the less comparison between set_str objects.

Parameters

lhs	Argument to left hand side of the operation.
rhs	Argument to right hand side of the operation

Returns

true if the lhs object is less than rhs, and false otherwise

compares both objects of the sets of the string based on the size of set and if equal then contents of the string set.

5.3.2.2 bool setstr_compare::operator() (const set_str & *lhs*, const set_str & *rhs*) const

The overloaded operator () to use the function object as a predicate.

Parameters

lhs	Argument to left hand side of the operation.
rhs	Argument to right hand side of the operation.

Returns

The result from member function less - true if the lhs object is less than rhs, and false otherwise

The function object operator () for returning whether the first argument compares less than second. Uses the static member function setstr_compare::less to determine the result.

Just calls the less function to determine the inequality of the parameter objects.

The documentation for this class was generated from the following files:

- declaration.h
- setstr_compare.cc

5.4 UserInterface Class Reference

The UserInterface class that represents the interface class which interacts with the user using command line interface and performs the actions accordingly.

```
#include <user_interface.h>
```

Public Member Functions

- void run ()
- ∼UserInterface ()
- template<>
  string **getValidInput** (const string msg, const string error, bool(∗isValid)(const string &))

Static Public Member Functions

- static UserInterface & instance ()

5.4.1 Detailed Description

The UserInterface class that represents the interface class which interacts with the user using command line interface and performs the actions accordingly.

The UserInterface class follows the singleton design pattern and instance of the this class can be retrieve by the UserInterface::instance method. It hides all the other detailed methods that handles the user input, validation of input and performs the actions accordingly. The interface will be 'started' by using the run method of the class.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 UserInterface::∼UserInterface ( ) `[inline]`

The destructor for UserInterface class

5.4.3 Member Function Documentation

5.4.3.1 static UserInterface& UserInterface::instance ( ) `[inline], [static]`

This is static method which provide the reference of the single UserInterface object.

Returns

It returns the reference of the is already existing UserInterface object.

The method will provide the single access point to the shared UserInterface object. This will ensures the singleton design pattern allowing only single instance of the class at any time.

5.4.3.2 void UserInterface::run ( )

This method will starts the user interface programs and handles user interaction using menu driven command line options

The documentation for this class was generated from the following files:

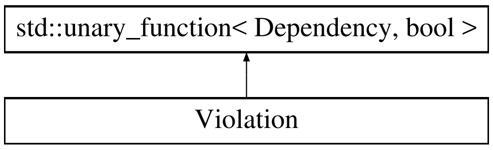
- user_interface.h
- user_interface.cc

5.5 Violation Struct Reference

A unary functor class used to test the dependency for the normal form violation.

```
#include <violation.h>
```

Inheritance diagram for Violation:



Public Member Functions

- bool operator() (const Dependency &dep) const

Friends

- class Relation

5.5.1 Detailed Description

A unary functor class used to test the dependency for the normal form violation.

The class Violation inherits the unary_function to define the unary predicate with Dependency as parameter object. It uses the data members with constant reference to the Relation and the constant reference to enumeration value of Noraml to represent the normal form. Note that the it have private constructor to restrict the use of this class limited to only friend class Relation.

5.5.2 Member Function Documentation

5.5.2.1 bool Violation::operator() (const **Dependency** & *dep*) const `[inline]`

overloaded operator () to be applied as predicate when testing the dependency for the violation.

Parameters

dep	is constant reference to the Dependency object for which the violation is to be tested.

Returns

true value if it violates the normal form, false otherwise.

The function uses the class member Relation and Normal form objects to perform the test against the Dependency object parameter. If the conditions for the particular normal form is violated then the function return true. The different conditions for the normal form violation can be given as follows -

- Second normal form violation if
  1. lhs of Dependency is partial key
  2. and rhs is not prime attribute set
- Third normal form violation if
  1. lhs of Dependency is not super-key
  2. and rhs is not prime attribute set
- Boyce-Codd(BC) normal form violation if
  1. lhs of Dependency is not super-key

5.5.3 Friends And Related Function Documentation

5.5.3.1 friend class **Relation** `[friend]`

only Relation calss have access to construct the object of the Violation struct and to use it.

The documentation for this struct was generated from the following file:

- violation.h

Chapter 6

File Documentation

6.1 declaration.h File Reference

Includes declaration of classes and definition of comparator class.

```
#include "typedef.h"
```

Classes

- class setstr_compare

  The function object class for less-than inequality comparison of the set of string.

Typedefs

- typedef set< set_str, setstr_compare > set_key

  A type definition for a set of set of string with set_str with the custom compare class setstr_compare.
- typedef set_key::iterator itr_key

  A type definition for a iterator for set_key.

6.1.1 Detailed Description

Includes declaration of classes and definition of comparator class.

This includes the forward declaration of classes from the file typedef.h. It declares the comparator class for set of set of string. It also declares the global type alias for set of the set of string.

Author

Ashish D. Kharde

6.2 dependency.cc File Reference

Includes definitions of the Dependency class members defined in the dependency.h file.

```
#include "dependency.h"
#include "utility.h"
#include <iostream>
#include <algorithm>
```

6.2.1 Detailed Description

Includes definitions of the Dependency class members defined in the dependency.h file.

This file contains definition of the undefined member functions of the class Dependency.

6.3 dependency.h File Reference

Includes declaration for the class Dependency and its members.

```
#include "declaration.h"
#include "utility.h"
#include <set>
#include <iostream>
```

Classes

- class Dependency

  The Dependency class that represents the functional dependency of the relation using the attribute set.

6.3.1 Detailed Description

Includes declaration for the class Dependency and its members.

This file declares the definition of the class Dependency along with its subsequent data members and the member functions prototype.

6.4 relation.cc File Reference

Includes definitions of the Relation class members defined in the relation.h file.

```
#include "relation.h"
#include "utility.h"
#include "dependency.h"
#include "violation.h"
```

6.4.1 Detailed Description

Includes definitions of the Relation class members defined in the relation.h file.

This file contains definition of all the undefined member functions of the class Relation.

6.5 relation.h File Reference

Includes declaration for the class Relation and its members.

```
#include "declaration.h"
#include "dependency.h"
#include <set>
#include <string>
```

Classes

- class Relation

  The Relation class that represents the relation entity.

6.5.1 Detailed Description

Includes declaration for the class Relation and its members.

This file declares the definition of the class Relation along with its subsequent data members and the member functions prototype.

Author

Ashish D. Kharde

6.6 template_def.h File Reference

Contains the definition for various utility template functions.

```
#include "declaration.h"
#include "relation.h"
```

Functions

- template<typename T >
  bool contains (const set< T > &list, const T &val)

  Template function to check whether the second parameter value is present in the set object represented by the first parameter.
- template<typename T >
  bool isEqual (const set< T > &lhs, const set< T > &rhs)

  Template function to check the equality of two sets of any datatype.
- template<typename T >
  bool isSubset (const set< T > &lhs, const set< T > &rhs)

  Template function to check whether the second parameter is subset of first of any datatype.
- template<typename T >
  ostream & printSet (ostream &out, const set< T > &pSet, const Parenthesis &par)

  Template function to insert the set of any type in the output stream using specified parenthesis.

6.6.1 Detailed Description

Contains the definition for various utility template functions.

Definition of different template method declared in the utility.h file. This file is to be included in the utility.h file.

Author

Ashish D. Kharde

6.6.2 Function Documentation

6.6.2.1 template< typename T > bool contains (const set< T > & *list*, const T & *val*) [inline]

Template function to check whether the second parameter value is present in the set object represented by the first parameter.

Parameters

list	Represents the set of any data-type.
val	Represents the value that need to be search from the parameter set object.

Returns

true if the val parameter is found in the set parameter list, false otherwise.

6.6.2.2 template< typename T > bool isEqual (const set< T > & *lhs*, const set< T > & *rhs*) [inline]

Template function to check the equality of two sets of any datatype.

Parameters

lhs	is the set object that represents first parameter.
rhs	is the set object that represents second parameter

Returns

true if both sets are equal, false otherwise.

The function uses the subset method to determine the equality of the two parameter set objects. If lhs is subset of the rhs and rhs is subset of the lhs means both sets are equal.

6.6.2.3 template< typename T > bool isSubset (const set< T > & *lhs*, const set< T > & *rhs*) [inline]

Template function to check whether the second parameter is subset of first of any datatype.

Parameters

lhs	is the first parameter represents the set object from which second parameter is to be look up.
rhs	is the second parameter represents the set object which is to be tested as subset of the first parameter.

Returns

true if the rhs is subset of the lhs, false otherwise.

The function checks whether every objects from the parameter rhs is present in the parameter lhs.

6.6.2.4 template< typename T > ostream& printSet (ostream & *out*, const set< T > & *pSet*, const Parenthesis & *par*)

Template function to insert the set of any type in the output stream using specified parenthesis.

Parameters

out	is the output stream object where the set data is to be inserted.
dispSet	is the set objects of any data type T.
par	is the enumeration object of Parenthesis representing the type of parenthesis to be used when formating the set objects in output stream. Default value is NO_BRAC in which case no parenthesis will be used to separate set objects.

Returns

The reference of the out parameter.

This function will insert all the set objects to output stream separated by the parenthesis if necessary. The output operator << should be overloaded for the datatype of the object to insert its content in the output stream.

6.7 typedef.h File Reference

Defines the global type alias for data-types.

```
#include <set>
#include <string>
```

Typedefs

- typedef std::string **string**
- typedef set< Relation > set_rel

  A type definition for a set of Relation.
- typedef set< Dependency > set_dep

  A type definition for a set of Dependency.
- typedef set< string > set_str

  A type definition for a set of strings.
- typedef set_rel::iterator itr_rel

  A type definition for a iterator of set of Relation.
- typedef set_dep::iterator itr_dep

  A type definition for a iterator of set of Dependency.
- typedef set_str::iterator itr_str

  A type definition for a iterator of set of string.

Variables

- const unsigned short int WIDTH = 12

  A global constant variable to specify the value of width used in formatting the stream output.

6.7.1 Detailed Description

Defines the global type alias for data-types.

Declaration of global type alias and the forward declaration of the classes.

Author

Ashish D. Kharde

6.8 user_interface.cc File Reference

Includes definitions of the UserInterface class members defined in the user_interface.h file.

```
#include "user_interface.h"
#include <iostream>
#include <regex>
#include <vector>
#include <sstream>
#include <iterator>
#include <functional>
#include <algorithm>
#include <cstdlib>
```

Macros

- #define PRINT std::cout<<std::setw(WIDTH) <<std::right<< ""

Functions

- string **getText** (Relation::Normal n)
- void **print** (std::vector< string > &vec)

6.8.1 Detailed Description

Includes definitions of the UserInterface class members defined in the user_interface.h file.

This file contains definition of the undefined member functions of the class UserInterface.

6.8.2 Macro Definition Documentation

6.8.2.1 #define PRINT std::cout<<std::setw(WIDTH) <<std::right<< ""

Defines the global output format using overloaded output with standard ostream object cout.

6.9 user_interface.h File Reference

Includes declaration for the class UserInterface and its members.

```
#include <iomanip>
#include <vector>
#include "declaration.h"
#include "relation.h"
```

Classes

- class UserInterface

  The UserInterface class that represents the interface class which interacts with the user using command line interface and performs the actions accordingly.

6.9.1 Detailed Description

Includes declaration for the class UserInterface and its members.

This file declares the definition of the class UserInterface along with its subsequent data members and the member functions prototype.

6.10 utility.cc File Reference

Specifies the definitions for the global non-member functions defined in utility.h header file.

```
#include "utility.h"
#include "dependency.h"
#include "relation.h"
#include <iomanip>
```

Functions

- ostream & operator<< (ostream &out, const set_key &dispSet)
  Insert set_key object into stream.
- ostream & operator<< (ostream &out, const set_rel &dispSet)
  Insert set_rel objects into stream.
- ostream & operator<< (ostream &out, const set_dep &dispSet)
  Insert set_dep objects into stream.
- ostream & operator<< (ostream &out, const set_str &dispSet)
  Insert set_str objects into stream.
- ostream & operator<< (ostream &out, const Dependency &d)
  Insert Dependency object into stream.
- ostream & operator<< (ostream &out, const Relation &rel)
  Insert Relation object into stream.
- set_key & operator-= (set_key &lhs, const set_key &rhs)
  Subtract and assign the set_key objects.
- set_str & operator-= (set_str &lhs, const set_str &rhs)
  Subtract and assign the set_str objects.
- bool testSet (const set_key &lhs, const set_str &rhs, bool super)
  Checks whether the set_str object is superset or subset for any set_str object of the set_key.

6.10.1 Detailed Description

Specifies the definitions for the global non-member functions defined in utility.h header file.

Definition of the all the non-member and non-template global functions defined in the header file utility.h

Author

Ashish D. Kharde

6.10.2 Function Documentation

6.10.2.1 ostream& operator<< (ostream & *out*, const set_key & *dispSet*)

Insert set_key object into stream.

Parameters

ostream	object where set_key objects are inserted.
set_key	object to insert.

Returns

The same parameter of ostream.

Inserts the all the set_str objects from the dispSet parameter into the stream with proper formatting, separating each set_str objects within the parenthesis and putting all the sets in the curly braces. The empty set_key object is indicated by { 0 }.

6.10.2.2 bool testSet (const **set_key** & *lhs,* const **set_str** & *rhs,* bool *super*)

Checks whether the set_str object is superset or subset for any set_str object of the set_key.

Parameters

lhs	The set_key
rhs	
super	To determine the type of check either superset true or the subset false. Default value is false for subset.

Returns

true if the second parameter is subset or superset of the any of first parameter object, false otherwise.

It checks whether the rhs set_str object is the super set or the subset of the any of set_str object. If it is subset or superset then the it return the true value. The check for either superset or the subset will be determined by the parameter super.

6.11 utility.h File Reference

Defines the prototypes for the global non-member functions.

```
#include "declaration.h"
#include <algorithm>
#include <iostream>
#include <set>
#include "template_def.h"
```

Enumerations

- enum Parenthesis {
  BRAC_ROUND, BRAC_SQUARE, BRAC_CURLY, BRAC_ANGEL,
  NO_BRAC }

Functions

- ostream & operator<< (ostream &, const Dependency &)

    Insert Dependency object into stream.
- ostream & operator<< (ostream &, const Relation &)

    Insert Relation object into stream.
- ostream & operator<< (ostream &, const set_key &)

    Insert set_key object into stream.
- ostream & operator<< (ostream &, const set_rel &)

    Insert set_rel objects into stream.
- ostream & operator<< (ostream &, const set_str &)

    Insert set_str objects into stream.
- ostream & operator<< (ostream &, const set_dep &)

    Insert set_dep objects into stream.
- set_key & operator-= (set_key &, const set_key &)

    Subtract and assign the set_key objects.
- set_str & operator-= (set_str &, const set_str &)

    Subtract and assign the set_str objects.
- bool testSet (const set_key &lhs, const set_str &rhs, bool super=false)

    Checks whether the set_str object is superset or subset for any set_str object of the set_key.
- string getBrac (const Parenthesis &sap, bool close)

    Retrieves the string representing the corresponding parenthesis.
- bool contains (const set_key &list, const set_str &val)

    Checks whether the set_str object is present in the set_key.
- template<typename T >
  ostream & printSet (ostream &, const set< T > &, const Parenthesis &sap=NO_BRAC)

    Template function to insert the set of any type in the output stream using specified parenthesis.
- template<typename T >
  bool isEqual (const set< T > &lhs, const set< T > &rhs)

    Template function to check the equality of two sets of any datatype.
- template<typename T >
  bool isSubset (const set< T > &lhs, const set< T > &rhs)

    Template function to check whether the second parameter is subset of first of any datatype.
- template<typename T >
  bool contains (const set< T > &lhs, const T &val)

    Template function to check whether the second parameter value is present in the set object represented by the first parameter.

6.11.1 Detailed Description

Defines the prototypes for the global non-member functions.

Declaration of the prototypes for the global non-member functions, overloaded operators and template functions. Also includes the file template_def.h which includes the definition of the template functions.

Author

Ashish D. Kharde

6.11.2 Enumeration Type Documentation

6.11.2.1 enum **Parenthesis**

The global enumerator used for representing the types of the brackets used while producing the output for different sets.

Enumerator

BRAC_ROUND	Represents round brackets
BRAC_SQUARE	Represents square/box brackets
BRAC_CURLY	Represents curly brackets
BRAC_ANGEL	Represents angel brackets
NO_BRAC	Represents no brackets

6.11.3 Function Documentation

6.11.3.1 bool contains (const **set_key** & *list,* const **set_str** & *val*) `[inline]`

Checks whether the set_str object is present in the set_key.

Parameters

list	A constant set of string sets to check from.
val	A constant string set representing the search value.

Returns

true for successful find false otherwise.

A alternate version of the template function contains for set of string set with different comparator i.e. setstr_compare.

6.11.3.2 template<typename T > bool contains (const set< T > & *list,* const T & *val*) `[inline]`

Template function to check whether the second parameter value is present in the set object represented by the first parameter.

Parameters

list	Represents the set of any data-type.
val	Represents the value that need to be search from the parameter set object.

Returns

true if the val parameter is found in the set parameter list, false otherwise.

6.11.3.3 string getBrac (const Parenthesis & *sap*, bool *close*) [inline]

Retrieves the string representing the corresponding parenthesis.

Parameters

sap	a constant Parenthesis enumeration object value to represent the type of braces.
close	a boolean parameter to determine the closing or opening brace. The true value indicates the closing brace and false value indicates the opening brace.

Returns

The string representing the equivalent value of the parenthesis. The default value will be the empty string.

6.11.3.4 template< typename T > bool isEqual (const set< T > & *lhs*, const set< T > & *rhs*) [inline]

Template function to check the equality of two sets of any datatype.

Parameters

lhs	is the set object that represents first parameter.
rhs	is the set object that represents second parameter

Returns

true if both sets are equal, false otherwise.

The function uses the subset method to determine the equality of the two parameter set objects. If lhs is subset of the rhs and rhs is subset of the lhs means both sets are equal.

6.11.3.5 template< typename T > bool isSubset (const set< T > & *lhs*, const set< T > & *rhs*) [inline]

Template function to check whether the second parameter is subset of first of any datatype.

Parameters

lhs	is the first parameter represents the set object from which second parameter is to be look up.
rhs	is the second parameter represents the set object which is to be tested as subset of the first parameter.

Returns

true if the rhs is subset of the lhs, false otherwise.

The function checks whether every objects from the parameter rhs is present in the parameter lhs.

6.11.3.6 ostream& operator<< (ostream & *out*, const set_key & *dispSet*)

Insert set_key object into stream.

Parameters

ostream	object where set_key objects are inserted.
set_key	object to insert.

Returns

The same parameter of ostream.

Inserts the all the set_str objects from the dispSet parameter into the stream with proper formatting, separating each set_str objects within the parenthesis and putting all the sets in the curly braces. The empty set_key object is indicated by { 0 }.

6.11.3.7 template< typename T > ostream& printSet (ostream & *out*, const set< T > & *pSet*, const Parenthesis & *par*)

Template function to insert the set of any type in the output stream using specified parenthesis.

Parameters

out	is the output stream object where the set data is to be inserted.
dispSet	is the set objects of any data type T.
par	is the enumeration object of Parenthesis representing the type of parenthesis to be used when formating the set objects in output stream. Default value is NO_BRAC in which case no parenthesis will be used to separate set objects.

Returns

The reference of the out parameter.

This function will insert all the set objects to output stream separated by the parenthesis if necessary. The output operator << should be overloaded for the datatype of the object to insert its content in the output stream.

6.11.3.8 bool testSet (const set_key & *lhs*, const set_str & *rhs*, bool *super*)

Checks whether the set_str object is superset or subset for any set_str object of the set_key.

Parameters

lhs	The set_key
rhs	
super	To determine the type of check either superset true or the subset false. Default value is false for subset.

Returns

true if the second parameter is subset or superset of the any of first parameter object, false otherwise.

It checks whether the rhs set_str object is the super set or the subset of the any of set_str object. If it is subset or superset then the it return the true value. The check for either superset or the subset will be determined by the parameter super.

6.12 violation.h File Reference

Includes declaration of functor class Violation used to check violation of any normal form by the Dependency for a Relation object.

```
#include "declaration.h"
#include "dependency.h"
#include "relation.h"
```

Classes

- struct Violation

  A unary functor class used to test the dependency for the normal form violation.

6.12.1 Detailed Description

Includes declaration of functor class Violation used to check violation of any normal form by the Dependency for a Relation object.

This file includes the declaration and definition of class Violation and its subsequent methods.

Author

Ashish D. Kharde

Index