Федеральное государственное бюджетное образовательное учреждение   
высшего образования

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | «Московский государственный технический университет  им. Н.Э. Баумана (национальный исследовательский университет)»(МГТУ им. Н.Э. Баумана) |

ФАКУЛЬТЕТ – Информатика и управления

КАФЕДРА – Информационные системы и телекоммуникации

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе на тему

Веб-сервис хранения данных набора Eclipse Workspaces

Студент группы ИУ3-73 (подпись) 07.02.2017 Я.А. Конов

Руководитель курсовой работы (подпись) 07.02.2017 А.М. Иванов

Москва, 2017

/\*\*

\*

\*/

package net.sf.tweety.commons.tests;

import java.io.IOException;

import java.math.BigInteger;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import org.junit.Assert;

import org.junit.Test;

import net.sf.tweety.commons.BeliefSet;

import net.sf.tweety.commons.Formula;

import net.sf.tweety.commons.Language;

import net.sf.tweety.commons.Parser;

import net.sf.tweety.commons.ParserException;

import net.sf.tweety.commons.Reasoner;

import net.sf.tweety.commons.streams.DefaultFormulaStream;

import net.sf.tweety.commons.streams.FormulaStream;

import net.sf.tweety.commons.util.ConversionTools;

import net.sf.tweety.commons.util.MathTools;

import net.sf.tweety.commons.util.Pair;

import net.sf.tweety.logics.commons.syntax.Constant;

import net.sf.tweety.logics.commons.syntax.Predicate;

import net.sf.tweety.logics.commons.syntax.Sort;

import net.sf.tweety.logics.fol.parser.FolParser;

import net.sf.tweety.logics.fol.syntax.FolSignature;

import net.sf.tweety.logics.pl.NaiveReasoner;

import net.sf.tweety.logics.pl.PlBeliefSet;

import net.sf.tweety.logics.pl.lang.PropositionalLanguage;

import net.sf.tweety.logics.pl.parser.PlParser;

import net.sf.tweety.logics.pl.syntax.Proposition;

import net.sf.tweety.logics.pl.syntax.PropositionalFormula;

import net.sf.tweety.logics.pl.syntax.PropositionalSignature;

/\*\*

\*

\* Tweety commons library tests.

\*

\*/

public class TweetyCommonsTest {

private static final String CARL = "carl";

private static final String MARTIN = "martin";

private static final String KNOWS = "Knows";

private static final String FOL\_FORMULA = KNOWS + "(" + MARTIN + ',' + CARL + ')';

private static final String RAINY = "rainy";

private static final String CLOUDY = "cloudy";

private static final String SUNNY = "sunny";

private static final String FIRST = "First";

private static final String SECOND = "Second";

private static final String AND = "&&";

private static final String OR = "||";

private static final String NOT = "!";

@Test

public void pairTest() {

Pair<String, String> pair = new Pair<>();

pair.setFirst(FIRST);

pair.setSecond(SECOND);

Pair<String, String> otherPair = new Pair<>(FIRST, SECOND);

Assert.assertEquals(pair, otherPair);

}

@Test

public void parserTest() throws ParserException, IOException {

Parser<PlBeliefSet> parser = new PlParser();

PropositionalFormula complexFormula = (PropositionalFormula)parser.parseFormula(NOT + RAINY + OR + CLOUDY);

Proposition propositionCloudy = (Proposition)parser.parseFormula(CLOUDY);

Proposition propositionRainy = (Proposition)parser.parseFormula(RAINY);

Assert.assertTrue(complexFormula.getAtoms().contains(propositionCloudy));

Assert.assertTrue(complexFormula.getAtoms().contains(propositionRainy));

}

@Test

public void reasonerTest() throws ParserException, IOException {

BeliefSet<PropositionalFormula> plBeliefSet = new PlBeliefSet();

Parser<PlBeliefSet> parser = new PlParser();

//today not rainy

plBeliefSet.add((PropositionalFormula)parser.parseFormula(NOT + RAINY));

//it can be rainy or sunny

plBeliefSet.add((PropositionalFormula)parser.parseFormula(RAINY + OR + SUNNY));

Reasoner reasoner = new NaiveReasoner(plBeliefSet);

//is it sunny today?

Assert.assertTrue(reasoner.query(parser.parseFormula(SUNNY)).getAnswerBoolean());

}

@Test

public void languageTest() throws ParserException, IOException {

Language propositionLanguage = new PropositionalLanguage(new PropositionalSignature());

PlParser plParser = new PlParser();

Assert.assertTrue(propositionLanguage.isRepresentable(plParser.parseFormula(RAINY + OR + SUNNY)));

FolParser folParser = createFolParser();

Assert.assertFalse(propositionLanguage.isRepresentable(folParser.parseFormula(FOL\_FORMULA)));

}

@Test

public void formulaStreamTest() throws ParserException, IOException {

Parser<PlBeliefSet> parser = new PlParser();

Collection<Formula> formulas = new ArrayList<>();

//fill formulas collection with some formulas

formulas.add(parser.parseFormula(NOT + RAINY));

formulas.add(parser.parseFormula(SUNNY));

formulas.add(parser.parseFormula(NOT + RAINY + AND + SUNNY));

formulas.add(parser.parseFormula(CLOUDY));

formulas.add(parser.parseFormula(NOT + RAINY + AND + CLOUDY));

FormulaStream<Formula> formulaStream = new DefaultFormulaStream<>(formulas);

Assert.assertEquals(formulas.size(), getStreamSize(formulaStream));

}

@Test

public void mathToolsTest() {

for (int number = 0; number < 100; number++)

{

Assert.assertEquals(factorial(number), MathTools.faculty(number));

}

int totalSize = 10;

for (int subsetSize = 0; subsetSize < totalSize; subsetSize++)

{

Assert.assertEquals(

Double.valueOf(

factorial(totalSize) / (factorial(totalSize - subsetSize) \* factorial(subsetSize))).intValue(),

MathTools.binomial(totalSize, subsetSize).intValue());

}

}

@Test

public void conversionToolsTest() {

BigInteger maxBigInt = new BigInteger("100000");

for (BigInteger bigInteger = BigInteger.ZERO; bigInteger.compareTo(maxBigInt) < 0; bigInteger = bigInteger.add(BigInteger.ONE))

{

Assert.assertEquals(bigInteger.toString(2), ConversionTools.bigInteger2BinaryString(bigInteger));

}

}

private int factorial(int n) {

int result = 1;

for (int i = 1; i <= n; i++)

{

result \*= i;

}

return result;

}

private Object getStreamSize(FormulaStream<Formula> formulaStream) {

int size = 0;

while (formulaStream.hasNext())

{

formulaStream.next();

size++;

}

return size;

}

private FolParser createFolParser() {

FolParser folParser = new FolParser();

FolSignature signature = new FolSignature();

Sort person = new Sort("Person");

signature.add(person);

List<Sort> argumentsSort = new ArrayList<>();

argumentsSort.add(person);

argumentsSort.add(person);

signature.add(new Predicate(KNOWS, argumentsSort));

signature.add(new Constant(MARTIN, person));

signature.add(new Constant(CARL, person));

folParser.setSignature(signature);

return folParser;

}

}

package net.sf.tweety.commons.tests;

import java.util.HashSet;

import java.util.Set;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import net.sf.tweety.commons.util.DefaultSubsetIterator;

import net.sf.tweety.commons.util.IncreasingSubsetIterator;

import net.sf.tweety.commons.util.MathTools;

import net.sf.tweety.commons.util.RandomSubsetIterator;

import net.sf.tweety.commons.util.SubsetIterator;

public class SubsetIteratorTest {

Set<Integer> integerSet;

@Before

public void setUp() {

integerSet = new HashSet<>();

for (int i = 0; i < 3; i++)

{

integerSet.add(i);

}

}

@Test

public void increasingSubsetIteratorTest() {

SubsetIterator<Integer> iterator = new IncreasingSubsetIterator<>(integerSet);

checkSubsetsSize(iterator, getExpectedSize(integerSet));

}

@Test

public void randomSubsetIteratorTest() {

SubsetIterator<Integer> iterator = new RandomSubsetIterator<>(integerSet, true);

checkSubsetsSize(iterator, getExpectedSize(integerSet));

}

@Test

public void defaultSubsetIteratorTest() {

SubsetIterator<Integer> iterator = new DefaultSubsetIterator<>(integerSet);

checkSubsetsSize(iterator, getExpectedSize(integerSet));

}

private int getExpectedSize(Set<?> set) {

int expectedSize = 0;

for (int subsetSize = 0; subsetSize <= set.size(); subsetSize++)

{

expectedSize += MathTools.binomial(set.size(), subsetSize);

}

return expectedSize;

}

private void checkSubsetsSize(SubsetIterator<?> iterator, int expectedSize) {

int subsetsCount = 0;

while (iterator.hasNext())

{

iterator.next();

subsetsCount++;

}

Assert.assertEquals(expectedSize, subsetsCount);

}

}