DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 3

Paradigma de Objetos.

Paradigmas de Lenguajes

Grupo Two and a Half Blondes

Integrante	LU	Correo electrónico
De Sousa Bispo, Germán	359/12	germandesousa@gmail.com
Fernandez, Esteban	691/12	esteban.pmf@gmail.com
Wright, Carolina	876/12	wright.carolina@gmail.com

Reservado para la cátedra

Instancia	Fecha	Docente	Nota
Primera entrega			
Segunda entrega			

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Código

1.1. PropositionalFormula

```
Object subclass: #PropositionalFormula
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'PLP-TP3'!
```

1.1.1. Métodos de instancia

```
not
  ^ Negation of: self.

& aFormula
  ^ Conjunction of: self and: aFormula.

| aFormula
  ^ Disjunction of: self and: aFormula.

==> aFormula
  ^ Implication of: self and: aFormula.

= aFormula
  ^ SubclassResponsibility

hash
  ^ SubclassResponsibility

allPropVars
  ^ SubclassResponsibility

negate
  ^ SubclassResponsibility

toNNF
  | formulaWithoutImplications |
  formulaWithoutImplications := self withoutImplications.
  ^ formulaWithoutImplications organizeNegations.

withoutImplications
  ^ SubclassResponsibility

organizeNegations
  ^ SubclassResponsibility

organizeNegationsFromNegation: aNegationFormula
  ^ SubclassResponsibility

operatorAsString
  ^ SubclassResponsibility

asString
  ^ SubclassResponsibility

printString
  ^ self asString
```

```
asStringWithParenthesis: aFormula
  ^ '( ', aFormula asString, ' )'.

asStringWithoutParenthesis: aFormula
  ^ aFormula asString.

representationAsStringIn: aBinaryFormula
  ^ SubclassResponsibility.

representationAsStringInNegation: aNegationFormula
  ^ aNegationFormula asStringWithParenthesis: self.
```

1.1.2. Métodos de Clase

```
PropositionalFormula class
  instanceVariableNames: ''

of: aFormula and: anotherFormula
  ^ SubclassResponsibility.

of: aFormula
  ^ SubclassResponsibility
```

1.2. BinaryPropositionalFormula

```
PropositionalFormula subclass: #BinaryPropositionalFormula
        instanceVariableNames: 'firstFormula secondFormula operator'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'PLP-TP3'
```

1.2.1. Métodos de Instancia

```
initWith: aFormula and: anotherFormula
  firstFormula := aFormula.
  secondFormula := anotherFormula.

setOperator: aOperator
  operator := aOperator

value: aValuation
  |firstResult secondResult msg |
  firstResult := firstFormula value: aValuation.
  secondResult := secondFormula value: aValuation.
  msg := Message selector: operator argument: secondResult.
  ^ msg sendTo: firstResult.

= aFormula
  "Had to implement this short-circuit evaluation.
   If not, should have implemented my own Boolean
   If didn't do this, UnaryPropositionalFormula and PropositionalVariables
   would have fail to understand secondFormula message
   It's not their responsibility to even know that secondFormula message exists"

  (self class == aFormula class) ifFalse: [ ^ false ].
  ^ (firstFormula = (aFormula firstFormula)) and:
       (secondFormula = (aFormula secondFormula))
```

```
hash
  ^ self class hash + firstFormula hash + secondFormula hash

firstFormula
  ^ firstFormula.

secondFormula
  ^ secondFormula.

allPropVars
  ^ firstFormula allPropVars union: secondFormula allPropVars.

negate
  ^ SubclassResponsibility

withoutImplications
  |firstResult secondResult msg |
  firstResult := firstFormula withoutImplications.
  secondResult := secondFormula withoutImplications.
  msg := Message selector: operator argument: secondResult.
  ^ msg sendTo: firstResult.

organizeNegations
  |firstResult secondResult msg |
  firstResult := firstFormula organizeNegations.
  secondResult := secondFormula organizeNegations.
  msg := Message selector: operator argument: secondResult.
  ^ msg sendTo: firstResult.

organizeNegationsFromNegation: aNegationFormula
  ^ aNegationFormula organizeByNegating: self.

asString
  | firstFormulaAsString secondFormulaAsString |
  firstFormulaAsString := firstFormula representationAsStringIn: self.
  secondFormulaAsString := secondFormula representationAsStringIn: self.
  ^ firstFormulaAsString, self operatorAsString, secondFormulaAsString

representationAsStringIn: aBinaryFormula
  ^ aBinaryFormula asStringWithParenthesis: self.
```

1.2.2. Métodos de Clase

```
BinaryPropositionalFormula class
  instanceVariableNames: ''

of: aFormula and: anotherFormula
  ^ self new initWith: aFormula and: anotherFormula

of: aFormula
  ^ ShouldNotImplement
```

1.3. Conjunction

```
BinaryPropositionalFormula subclass: #Conjunction
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
```

```
        category: 'PLP–TP3'
```

1.3.1. Métodos de Instancia

```
operatorAsString
  ^ ' & '

negate
  ^ firstFormula negate | (secondFormula negate).
```

1.3.2. Métodos de Clase

```
Conjunction class
        instanceVariableNames: ''

of: aFormula and: anotherFormula
        | formula |
        formula := super of: aFormula and: anotherFormula.
        ^ formula setOperator: #&.
```

1.4. Disjunction

```
BinaryPropositionalFormula subclass: #Disjunction
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'PLP–TP3'
```

1.4.1. Métodos de Instancia

```
operatorAsString
  ^ ' | '

negate
  ^ firstFormula negate & (secondFormula negate).
```

1.4.2. Métodos de Clase

```
Disjunction class
        instanceVariableNames: ''

of: aFormula and: anotherFormula
        | formula |
        formula := super of: aFormula and: anotherFormula.
        ^ formula setOperator: #|.
```

1.5. Implication

```
BinaryPropositionalFormula subclass: #Implication
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'PLP–TP3'
```

1.5.1. Métodos de Instancia

```
operatorAsString
  ^ ' ==> '

negate
  ^ firstFormula & (secondFormula negate).

withoutImplications
  ^ (firstFormula not | secondFormula) withoutImplications

organizeNegations
  ^ ShouldNotImplement.

organizeNegationsFromNegation: aNegationFormula
  ^ ShouldNotImplement.
```

1.5.2. Métodos de Clase

```
Implication class
        instanceVariableNames: ''

of: aFormula and: anotherFormula
  | formula |
  formula := super of: aFormula and: anotherFormula.
  ^ formula setOperator: #==>.
```

1.6. UnaryPropositionalFormula

```
PropositionalFormula subclass: #UnaryPropositionalFormula
        instanceVariableNames: 'firstFormula'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'PLP-TP3'
```

1.6.1. Métodos de Instancia

```
initWith: aFormula
  firstFormula := aFormula.

value: aValuation
  ^ SubclassResponsibility

= aFormula
  "Had to implement this short-circuit evaluation.
   If not, should have implemented my own Boolean
   If didn't do this, UnaryPropositionalFormula and PropositionalVariables
   would have fail to understand secondFormula message
   It's not their responsibility to even know that secondFormula message exists"
  (self class == aFormula class) ifFalse: [ ^ false ].
  ^ (firstFormula = (aFormula firstFormula))

hash
  ^ self class hash + firstFormula hash

firstFormula
  ^ firstFormula.
```

```
allPropVars
  ^ firstFormula allPropVars.

negate
  ^ SubclassResponsibility

asString
  | formulaAsString |
  formulaAsString := firstFormula representationAsStringInNegation: self.
  ^ self operatorAsString, formulaAsString

representationAsStringIn: aBinaryFormula
  ^ aBinaryFormula asStringWithoutParenthesis: self.
```

1.6.2. Métodos de Clase

```
UnaryPropositionalFormula class
  instanceVariableNames: ''

of: aFormula and: anotherFormula
  ^ ShouldNotImplement

of: aFormula
  ^ self new initWith: aFormula
```

1.7. Negation

```
UnaryPropositionalFormula subclass: #Negation
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'PLP-TP3'
```

1.7.1. Métodos de Instancia

```
value: aValuation
  ^ (firstFormula value: aValuation) not

operatorAsString
  El simbolo de negacion no anda en latex con los plugins usados.

negate
  ^ firstFormula.

withoutImplications
  ^ firstFormula withoutImplications not

organizeNegations
  ^ firstFormula organizeNegationsFromNegation: self.

organizeNegationsFromNegation: aNegationFormula
  ^ aNegationFormula organizeByNegating: self.

notOrganize: aFormula
  ^ self.
```

```
organizeByNegating: aFormula
  ^ aFormula negate organizeNegations.
```

1.7.2. Métodos de Clase

```
Negation class
        instanceVariableNames: ''

of: aFormula and: anotherFormula
        ^ MessageNotUnderstood.
```

1.8. PropositionalVariable

```
Object subclass: #PropositionalVariable
        instanceVariableNames: 'name'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'PLP-TP3'
```

1.8.1. Métodos de Instancia

```
not
  ^ Negation of: self

& aFormula
  ^ Conjunction of: self and: aFormula

| aFormula
  ^ Disjunction of: self and: aFormula

==> aFormula
  ^ Implication of: self and: aFormula

initWith: aName
  name := aName.

value: aValuation
  ^ aValuation includes: name.

= aFormula
  ^ (self class == aFormula class) and: (name = (aFormula name))

allPropVars
  ^ Set with: name

negate
  ^ self not.

withoutImplications
  ^ self

organizeNegations
  ^ self

organizeNegationsFromNegation: aNegationFormula
  ^ aNegationFormula notOrganize: self.

asString
```

```
  ^ name

printString
  ^ self asString

representationAsStringIn: aBinaryFormula
  ^ aBinaryFormula asStringWithoutParenthesis: self.

representationAsStringInNegation: aNegationFormula
  ^ aNegationFormula asStringWithoutParenthesis: self.
```

1.8.2. Métodos de Clase

```
PropositionalVariable class
        instanceVariableNames: ''

named: aName
        ^ self new initWith: aName.
```