

```

1  #----- Domain Class-----
2  class Domain():
3      def __init__(self, name, start, end):
4          self.name = name
5          self.values = range(start,end+1)
6          self.vars = []
7
8      def addVariable(self,variable):
9          self.vars.append(variable.lower())
10
11     def getValues(self):
12         return self.values
13
14     def getVars(self):
15         return vars
16
17     def __str__(self):
18         res = ":" + self.name + " rdf:type :Domain ;\n" + "\t:values "
19         for val in self.values[:-1]:
20             res += str(val) + ", "
21         res += str(self.values[-1]) + " ;\n\t:variables"
22         for var in self.vars[:-1]:
23             res += " : " + str(var) + ", "
24         res += " : " + self.vars[-1] + ".\n\n"
25         return res
26
27
28
29  #----- Constraint Class-----
30  class Constraint:
31      def __init__(self,varDom):
32          self.typeCons = ""
33          self.vars = []
34          self.values = []
35          self.varDom = varDom
36
37      def addVar(self, var):
38          self.vars.append(var)
39
40      def addValue(self, value):
41          self.values.append(value)
42
43      def setTypeCons(self,typeCons):
44          self.typeCons = typeCons
45          if self.typeCons == "Reject:\n":
46              self.first = " != "
47              self.second = " || "
48              self.third = " && "
49          elif self.typeCons == "Accept:\n":
50              self.first = " = "
51              self.second = " && "
52              self.third = " || "
53
54      def __str__(self):
55          print(self.vars)
56          print(self.values)
57          res = ""
58          for i in range(len(self.values)):
59              if self.typeCons == "Reject:\n":
60                  res += " ObjectComplementOf("
61              if len(self.vars) > 1:
62                  res += " ObjectIntersectionOf("
63              for j in range(len(self.vars)):
64                  res += " ObjectHasValue("
65                  res += ":" + self.vars[j].lower()
66                  res += " "
67                  res += ":" + self.varDom[self.vars[j]].lower() + "val" +
68                      str(self.values[i][j]).lower() + ")"
69              if len(self.vars) > 1:
69                  res += ")"
70              if self.typeCons == "Reject:\n":
71                  res += ")"

```

```

72
73     return res
74
75
76
77
78 #----- Script itself -----
79 class MainRun:
80     def __init__(self):
81         self.inFileName = ""
82         self.outFileName = ""
83         self.domains = {}
84         self.varDom = {}
85         self.fileOutOWL = None
86
87     def parseConstraints(self, file, nConst):
88         if nConst > 1:
89             self.fileOutOWL.write(" ObjectIntersectionOf(")
90         nConstParsed = 0
91         for line in file:
92             constraint = Constraint(self.varDom)
93             if("Vars:" in line):
94                 if nConstParsed < nConst:
95                     nVars = int(file.readline())
96                     for x in range(nVars):
97                         var = file.readline().rstrip('\n')
98                         constraint.addVar(var)
99                     typeCons = file.readline()
100                     constraint.setTypeCons(typeCons)
101                     nValues = int(file.readline())
102                     for x in range(nValues):
103                         lineValue = file.readline().rstrip('\n')
104                         constraint.addValue(lineValue.split())
105                     self.fileOutOWL.write(str(constraint))
106                     nConstParsed += 1
107         if nConst > 1:
108             self.fileOutOWL.write(")")
109
110
111     def writeHashTagSeparator(self, title):
112         self.fileOutOWL.write("#####\n")
113         self.fileOutOWL.write("# " + title + "\n")
114         self.fileOutOWL.write("#####\n\n")
115
116
117     def writeObjectProperties(self):
118         self.writeHashTagSeparator("Object Properties")
119         for _, d in self.domains.items():
120             for v in d.vars:
121                 self.fileOutOWL.write("# Object Property: " + v + " (: " + v +
122                                     ")\n\n")
123                 self.fileOutOWL.write("FunctionalObjectProperty(: " + v + ")\n")
124                 self.fileOutOWL.write("ObjectPropertyDomain(: " + v + " :Var)\n")
125                 self.fileOutOWL.write("ObjectPropertyRange(: " + v + " : " + d.name +
126                                     ")\n\n")
127
128     def writeClasses(self, fileIn):
129         self.writeHashTagSeparator("Classes")
130         for _, d in self.domains.items():
131             self.fileOutOWL.write("# Class: " + d.name + " (: " + d.name + ")\n\n")
132             self.fileOutOWL.write("EquivalentClasses(: " + d.name + " ObjectOneOf(")
133             for val in d.values:
134                 self.fileOutOWL.write(": " + d.name.lower() + "val" + str(val) + " ")
135             self.fileOutOWL.write("))\n\n")
136             self.fileOutOWL.write("# Class: :Fml (:Fml)\n\n")
137             self.writeFml(fileIn)
138             self.fileOutOWL.write("# Class: :Var (:Var)\n\n")
139             for _, d in self.domains.items():
140                 for v in d.vars:
141                     self.fileOutOWL.write("EquivalentClasses(:Var
142                                         ObjectExactCardinality(1 : " + v + " : " + d.name + ")\n")
143                     self.fileOutOWL.write("\n\n")

```

```

141 def writeNamesIndividuals(self):
142     self.writeHashTagSeparator("Named Individuals")
143     for _ , d in self.domains.items():
144         for val in d.values:
145             self.fileOutOWL.write("# Individual: ":" + d.name.lower() + "val" +
146                 str(val) + " (::" + d.name.lower() + "val" + str(val) + ")\n\n")
147             self.fileOutOWL.write("ClassAssertion(:" + d.name + " ::" +
148                 d.name.lower() + "val" + str(val) + ")\n\n")
149 self.fileOutOWL.write("# Individual: :fml (:fml)\n\n")
150 self.fileOutOWL.write("ClassAssertion(:Fml :fml)\n\n")
151 self.fileOutOWL.write("SameIndividual(:fml :map)\n\n")
152 self.fileOutOWL.write("# Individual: :map (:map)\n\n")
153 self.fileOutOWL.write("ClassAssertion(:Var :map)\n\n")
154 self.fileOutOWL.write("\n\n")
155 for _ , d in self.domains.items():
156     self.fileOutOWL.write("DifferentIndividuals(")
157     for val in d.values:
158         self.fileOutOWL.write("::" + d.name.lower() + "val" + str(val) + " ")
159     self.fileOutOWL.write(")\n\n")
160
161 def writeFml(self,fileIn):
162     self.fileOutOWL.write("EquivalentClasses(:Fml")
163     n = int(fileIn.readline())
164     self.parseConstraints(fileIn,n)
165     self.fileOutOWL.write(")\n\n")
166
167 def run(self):
168     self.inFileName = input("Enter F2CSP file name:")
169     self.outFileName = input("Enter output file name:")
170     self.fileOutOWL = open(self.outFileName + ".owl","w+")
171
172     self.fileOutOWL.write("Prefix(:=<http://www.semanticweb.org/grupo21/ontologies
173 /2019/4/grupo21#>)\n")
174     self.fileOutOWL.write("Prefix(owl:=<http://www.w3.org/2002/07/owl#>)\n")
175
176     self.fileOutOWL.write("Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns
177 #>)\n")
178     self.fileOutOWL.write("Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)\n")
179     self.fileOutOWL.write("Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)\n")
180
181     self.fileOutOWL.write("Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)\n
182 ")
183
184     self.fileOutOWL.write("\n\nOntology(<http://www.semanticweb.org/grupo21/ontolo
185 gies/2019/4/" + self.outFileName + ">\n\n")
186
187     fileIn = open(self.inFileName, "r")
188     for line in fileIn:
189         if("Domains:" in line):
190             nDomains = int(fileIn.readline())
191             for _ in range(nDomains):
192                 currD = fileIn.readline()
193                 d = currD.split()
194                 self.domains[d[0]] = Domain(d[0], int(d[1][0]),int(d[1][-1]))
195             for d in self.domains.keys():
196                 self.fileOutOWL.write("Declaration(Class(:" + d + "))\n")
197             self.fileOutOWL.write("Declaration(Class(:Fml))\n")
198             self.fileOutOWL.write("Declaration(Class(:Var))\n")
199
200         if("Variables:" in line):
201             nVars = int(fileIn.readline())
202             for _ in range(nVars):
203                 currV = fileIn.readline()
204                 v = currV.split()
205                 self.domains[v[1]].addVariable(v[0])
206                 self.varDom[v[0]] = v[1]
207             for _ , d in self.domains.items():
208                 for v in d.vars:
209                     self.fileOutOWL.write("Declaration(ObjectProperty(:" + v +
210 ") )\n")

```

```
202         for _, d in self.domains.items():
203             for val in d.values:
204                 self.fileOutOWL.write("Declaration(NamedIndividual(:" +
205                                         d.name.lower() + "val" + str(val) + "))\n")
206                 self.fileOutOWL.write("Declaration(NamedIndividual(:fml))\n")
207                 self.fileOutOWL.write("Declaration(NamedIndividual(:map))\n")
208                 self.writeObjectProperties()
209                 self.fileOutOWL.write("\n")
210
211             if("Constraints:" in line):
212                 self.writeClasses(fileIn)
213                 self.writeNamesIndividuals()
214                 self.fileOutOWL.write(" ")
215                 fileIn.close()
216                 self.fileOutOWL.close()
217             print("SCRIPT END")
218
219
220 scriptRun = MainRun()
221 scriptRun.run()
```