Relatório de Engenharia do Conhecimento

Fase 3

Engenharia Informática

Grupo 21

João David n49448

João Marques n49038

Luís Moreira n49531

Índice

Script para gerar ficheiro com ontologia em OWL 2 DL

O ficheiro F2CSPtoOWL.py corresponde ao script responsável por interpretar o ficheiro F2CSP e gerar o ficheiro ".owl".

O ficheiro de entrada F2CSP convêm estar na mesma directoria do script, visto que ao utilizar determinadas formas de caminhos para o ficheiro poderá causar algum erro, desta forma, ao ter o ficheiro de entrada junto do script, basta escrever o seu nome, o mesmo acontece quando definir o nome do ficheiro de saída.

Ao correr o script, é pedido ao utilizador o nome do ficheiro F2CSP, e o nome do ficheiro de saída (como exemplo, considere o nome do ficheiro de saída "out"), após o script interpretar o ficheiro F2CSP, será criado o ficheiro out.owl que pode posteriormente ser aberto utilizando o Protege.

O script espera que o ficheiro F2CSP fornecido esteja corretamente escrito, caso contrário terá um comportamento imprevisível e os ficheiros de saída não deveram ser considerados.

Implementação do script F2CSPtoOWL

Na implementação do script foram criadas três classes (todas no mesmo ficheiro, F2CSPtoOWL.py), a class Domain, Constraint e MainRun.

Ao correr a script a class MainRun vai ser executada e inicia a interpretação do ficheiro F2CSP, ao encontrar a label "Domains:" sabe que estão definidos em baixo os domínios, por cada domínio encontrado, é criado um objecto Domain e adiciona-o a um dicionário (key:"D1", value:Domain), quando posteriormente encontra a label "Variables:" procede à analise das variáveis que estão definidas na forma "V11 D1", para adicionar esta variável ao domínio definido anteriormente, acede-se ao dicionário com a chave do domínio, neste caso "D1", e adiciona-se "V11" à lista de variáveis pertencentes ao respectivo domínio.

Ao encontrar a label "Constraints:", o MainRun sabe que a partir dai vai encontrar todas as constraints do F2CSP, portanto, para cada constraint encontrada vai criar um objecto "Constraint", e enquanto analisa a mesma constraint, vai adicionando informação ao objecto, nomeadamente as variáveis envolvidas, o tipo de constraint (Reject ou Accept) e os valores associados a cada variável que serão rejeitados ou aceites com base no tipo de constraint.

Interpretação da solução desenvolvida

Para simplificar, tome como exemplo o ficheiro de input F2CSP denominado “2x2F2CSP.txt”, que trata de resolver as restrições de um tabuleiro 2x2, em que na mesma linha e na mesma coluna não há valores repetidos, valores esses que estão contidos no único domínio “D1 1..2”.

V11	V12
V21	V22

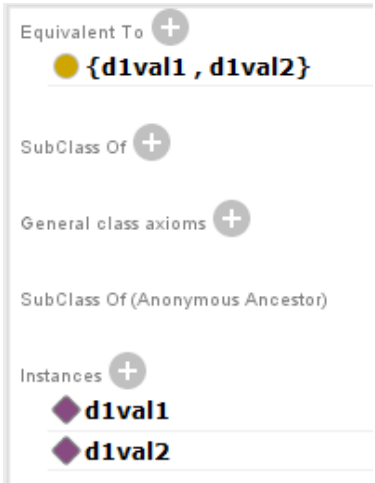
A interface do Protege será utilizada no auxílio da explicação da solução desenvolvida.

Neste caso, serão criadas três Classes, duas classes obrigatórias independentemente do problema (Fml e Var), e uma classe Dx por cada domínio definido no F2CSP.

Classe Dx

Neste caso existe apenas a classe D1. Tendo em conta o número de valores do domínio definido no ficheiro F2CSP, o mesmo número de instâncias desta classe será criado, utilizando a sintaxe “dXvalY”, em que X corresponde ao número do domínio, e Y ao valor respectivo a esse domínio. Por exemplo, o ficheiro F2CSP do problema enunciado anteriormente tem um único domínio “D1 1..2”. Logo será criada apenas uma classe referente aos domínios do problema, a classe D1, e essa classe terá como instância d1val1 e d1val2, visto que o domínio vai de 1 a 2.

Por fim é ainda definido em todas as classes domínio criadas a propriedade “Equivalent to” onde serão escritas todas as instâncias desse mesmo domínio. Esta propriedade permite-nos fechar o mundo, é uma forma de dizer que esta classe apenas tem estas instâncias.



Classe Fml

É nesta classe que serão escritas as restrições do problema, mais precisamente na propriedade "Equivalent To", será criada uma conjunção com todas as constraints definidas no F2CSP, quando uma constraint é do tipo Reject, será aplicado o prefixo not de forma a negar a atribuição desses mesmos valores em conjunto. É também criada uma única instância desta classe com o nome fml onde serão inferidas as object properties assim que se ligar o Reasoner.

Por exemplo, no ficheiro "2x2F2CSP.txt" temos quatro constraints:

```
C1:
Vars:
2
V11
V12
Reject:
2
1 1
2 2
```

```
C2:
Vars:
2
V11
V21
Reject:
2
1 1
2 2
```

```
C3:
Vars:
2
V12
V22
Reject:
2
1 1
2 2
```

```
C4:
Vars:
2
V21
V22
Reject:
2
1 1
2 2
```

O desenvolvimento das restrições é o seguinte:

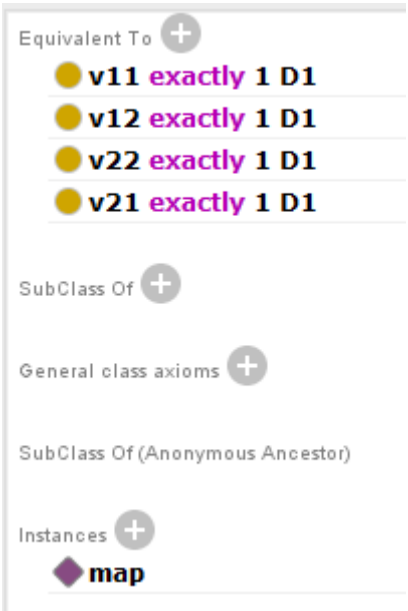
- Satisfy(C1) **and** Satisfy(C2) **and** Satisfy(C3) **and** Satisfy(C4)
    - ((not ((v11 value d1val1) and (v12 value d1val1)))
    - **and**
    - ((not ((v11 value d1val1) and (v21 value d1val1)))
    - **and**
    - ((not ((v12 value d1val1) and (v22 value d1val1)))
    - **and**
    - ((not ((v21 value d1val1) and (v22 value d1val1)))

Estas condições não são suficientes para obter uma solução, visto que para o protege conseguir devolver uma solução válida, o problema necessita de ter apenas uma solução possível, desta forma é necessário neste caso criar uma constraint do tipo Accept: de forma a que só haja uma solução possível. Por exemplo, ao adicionar a seguinte constraint ás anteriores permitiria obter uma solução válida do problema. Se o problema tiver mais que uma solução, o Protege lançará um erro de "inconsistente ontologies".

```
C5:
Vars:
2
V11
V12
Reject:
1
1 2
```

Classe Var

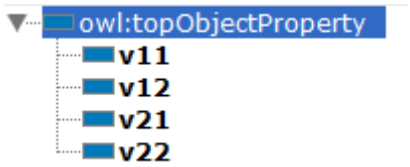
Nesta classe será definido que para cada object property terá apenas um domínio correspondente ao seu domínio definido no ficheiro F2CSP. Isto obriga a que exista apenas um valor do domínio em causa atribuído à variável do problema. É também definida apenas uma instância com nome "map" correspondente ao mapa do problema.

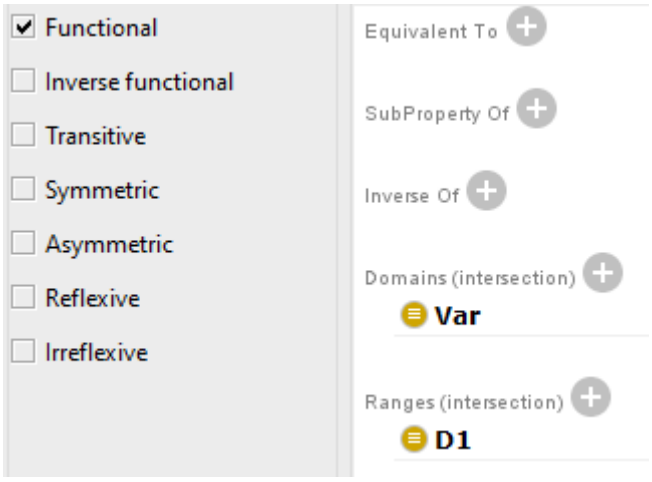


Object Properties

Serão criados “x” Object Properties, em que “x” corresponde ao número total de variáveis definidas no ficheiro F2CSP. Cada object property vai ter como nome a variável que foi definida no ficheiro F2CSP e terá como domain a classe Var e como range a classe domínio a que pertence a variável no ficheiro F2CSP.

No caso do problema referido em cima, existira 4 object properties, cujos nomes serão v11, v12, v21 e v22, cujas definições são como consta em baixo. É ainda definida a característica “Functional” de forma a que haja apenas seja atribuído para cada variável, se o problema tiver mais que uma solução, o reasoner lançará um erro de “inconsistente ontologies”.





São as object properties que efectivamente permitem saber os valores os resultados que se deve atribuir às variáveis do problema de forma a obter a solução. Após o reasoner correr, estas object properties serão inferidas na instância fml.

Entrega e anexos

Os ficheiros entregues são os seguintes:

- F2CSPtoOWL.py
  - Script em Python 3 responsável por converter F2CSP em OWL
- 2x2F2CSP.txt
  - Ficheiro F2CSP referente ao jogo enunciado anteriormente
- SudokuToF2CSP.py
  - Script em Python 3 responsável por gerar sudokus em F2CSP, foi esta script a utilizada para gerar os ficheiros F2CSP usados no desenvolvimento do script
- Ficheiros F2CSP
  - 2x2F2CSP.txt
    - Referente ao jogo enunciado na explicação do desenvolvimento
  - Sudoku4x4_F2CSP_B.txt
    - Sudoku 4x4 com 15 pistas, 1 solução possível
  - Sudoku9x9_F2CSP_C.txt
    - Sudoku 9x9 com 35 pistas, 1 solução possível
  - Sudoku9x9_F2CSP_D.txt
    - Sudoku 9x9 com 40 pistas, 1 solução possível
  - Sudoku9x9_F2CSP_E.txt
    - Sudoku 9x9 com 80 pistas, 1 solução possível