Relatório de Engenharia do Conhecimento

Fase 2

Engenharia Informática

Grupo 21

João David n49448

João Marques n49038

Luís Moreira n49531

Índice

Script para gerar query SPARQL

O ficheiro F2CSPtoSPARQL.py corresponde ao script responsável por gerar o ficheiro .ttl que contem o conhecimento dos domínios e respectivas variáveis do problema F2CSP e também por gerar o ficheiro .rq que contém a query SPARQL que resolve o problema com base nas constraints do ficheiro F2CSP.

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, vão ser criados dois ficheiros, um denominado out.ttl e outro out.rq, para obter a solução do problema basta utilizar estes dois ficheiros com o SPARQL.

Implementação do script F2CSPtoSPARQL

Na implementação do script foram criadas três classes (todas no mesmo ficheiro, F2CSPtoSPARQL.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, por fim, escreve no ficheiro out.rq a condição dessa constraint na zona do FILTER, quando isto acontece, há que ter em conta que operadores lógicos utilizar, visto que o tipo de constraint influência a forma como a condição é escrita, o seguinte tópico do relatório irá explicar como é feita a escrita da query.

Escrita da query

Para simplificar, tomemos como exemplo um ficheiro de input F2CSP, que é um tabuleiro 2x2, em que na mesma linha e na mesma coluna não há valores repetidos.

Assim que é iniciado a escrita do ficheiro out.rq, são escritos os PREFIX utilizados, e de seguida escrito o SELECT para todas as variáveis de todos os domínios envolvidos no ficheiro F2CSP.

```
PREFIX : <http://www.w3.org>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?v11 ?v12 ?v21 ?v22
```

Depois segue-se a escrita do WHERE, onde se escreve para cada domínio, as suas variáveis em cada linha seguindo o padrão utilizado no ficheiro out.ttl

No ficheiro out.ttl temos:

```
:D1 rdf:type :Domain ;
    :values 1, 2 ;
    :variables :V11, :V12, :V21, :V22.
```

No ficheiro out.rq temos:

```
WHERE {
    :D1 :values ?V11.
    :D1 :values ?V12.
    :D1 :values ?V21.
    :D1 :values ?V22.
```

Chegamos então a parte onde a "magia" acontece, a escrita do FILTER, onde estão as restrições de todas as constraints lidas do ficheiro F2CSP, como dito anteriormente, sempre que é lido uma constraint, o seu tipo (Accept ou Reject) influência a forma como é escrito a condição.

Tome como exemplo a escrita da seguintes constraints, em que a única diferença é no seu tipo:

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

```
C2:
Vars:
2
V11
V12
Accept:
2
1 1
2 2
```

No caso do Reject, a condição será:

```
( (?V11 != 1 || ?V12 != 1) && (?V11 != 2 || ?V12 != 2) )
```

Enquanto que no Accept será:

```
( (?V11 = 1 && ?V12 = 1) || (?V11 = 2 && ?V12 = 2) )
```

A única diferença entre estas duas condições são os operadores utilizados (coloridos com amarelo, castanho e verde), foi com base neste padrão que foi definido o método que define o tipo de constraint na class Constraint.

```
class Constraint:
    def setTypeCons(self,typeCons):
        self.typeCons = typeCons
        if self.typeCons == "Reject:\n":
            self.first = " != "
            self.second = " || "
            self.third = " && "
        elif self.typeCons == "Accept:\n":
            self.first = " = "
            self.second = " && "
            self.third = " || "
```

As condições são escritas para cada constraint presente no ficheiro F2CSP e adicionado ao filter da query sparql, cada condição escrita é sempre separada por &&.

Considere o ficheiro ".rq" do jogo 2x2 mencionado anteriormente, em que em cada linha e coluna não há números repetidos.

```
PREFIX : <http://www.w3.org>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?V11 ?V12 ?V21 ?V22
WHERE {
    :D1 :values ?V11.
    :D1 :values ?V12.
    :D1 :values ?V21.
    :D1 :values ?V22.
    FILTER (
        ( (?V11 != 1 || ?V12 != 1) && (?V11 != 2 || ?V12 != 2) )
        &&
        ( (?V11 != 1 || ?V21 != 1) && (?V11 != 2 || ?V21 != 2) )
        &&
        ( (?V12 != 1 || ?V22 != 1) && (?V12 != 2 || ?V22 != 2) )
        &&
        ( (?V21 != 1 || ?V22 != 1) && (?V21 != 2 || ?V22 != 2) )
    )
}
```