Representação do Conhecimento

Utilização de Dados Semânticos

TripleStore

Uma Implementação

A Pesquisa

Pesquisa

- Exemplo usando o método *triples()* de filtragem:
  - Lista de atores dirigidos por um realizador

```
realizador = input("Insira o nome do realizador: ")
idRealizador = _graph.valueSubject("name", realizador)
if idRealizador == None:
    print "Realizador desconhecido!!!"
    return
for s1, p1, o1 in _graph.triples(None, "directed_by", idRealizador):
    print "No filme = ", _graph.valueObject(s1, "name"), "orientou:"
    for s2, p2, o2 in _graph.triples(s1, "starring", None):
        print "Ator = ", _graph.valueObject(o2, "name")
```

Linguagem de Pesquisa

- Apesar da grande utilidade da funcionalidade de filtragem, esta não tira facilmente partido da riqueza do conjunto de relações existente num grafo.

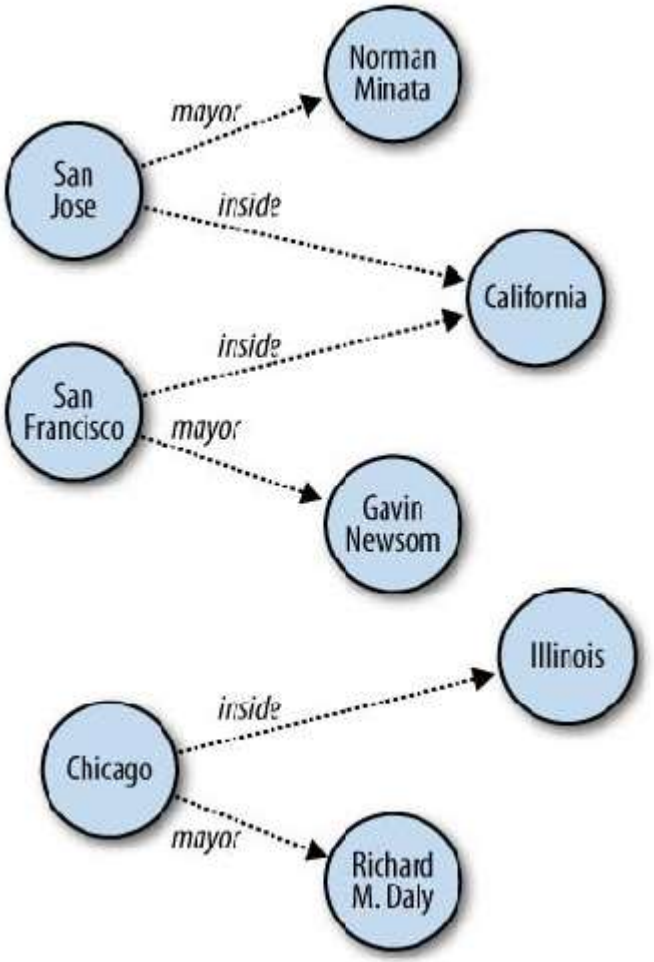
- Para melhorar o acesso à informação disponibilizada pelos grafos de triplos, pode-se criar uma linguagem simples de pesquisa, que permite a abstração do processo elementar visto antes.

Identificadores

- Tendo por base o grafo ao lado, pode-se escrever os seguintes triplos:

("San_Francisco_California", "inside", "California")
("San_Francisco_California", "mayor", "Gavin Newsom")

- Estes triplos usam um identificador para a cidade de “San Francisco”



Identificadores

- A escolha dos identificadores para entidades nos triplos é arbitrária, tal como quando se escolhe nomes para variáveis nos programas.
- O único requisito é o seu uso ser consistente.
- Por exemplo, os triplos anteriores podem ser escritos como a seguir e o seu significado permanece igual:

```
("foo", "inside", "California")
("foo", "mayor", "Gavin Newsom")
```

Identificadores

- É importante não pensar no identificador como o nome da entidade.
- Os identificadores são meros símbolos arbitrários que indicam que múltiplas declarações (triplos) estão relacionadas.
- No exemplo anterior, se se pretender dar nome à entidade cujo mayor é Newsom, então deve existir a declaração:

```
("foo", "name", "San Francisco")
```

Variáveis (i)

- Questão:
  - Quais as entidades presentes no grafo, com predicado “inside” e objeto “California”?

- Resposta:
  - Usando o método *triples()*, usa-se o triplo:
    (None, "inside", "California")
  - O resultado seria um conjunto de triplos e não apenas os valores do sujeito.
  - Mas se, alternativamente, se usa-se:
    ("?city", "inside", "California")
  - “?city” seria uma variável que devolveria todos os valores encontrados

Variáveis (ii)

- Resultados:
  - A variável “?city” teria os valores:

    “San_Francisco_California” e “San_Jose_California”

  - Uma forma de expressar esse conjunto de valores seria através de uma lista de dicionários:

```
[
  {"?city": "San_Francisco_California"},
  {"?city": "San_Jose_California"}
]
```

Variáveis (iii)

- Este tipo de variáveis permite guardar os valores resultado de uma *query* e torna-se possível combinar múltiplas *queries*, usando a interseção ou união dos resultados individuais.

- Exemplo:
  - Fazer as 2 *queries* seguintes e intersetar os resultados obtidos.

("?city", "inside", "California")
("?city", "mayor", "Norman Mineta")

Variáveis (iv)

- O resultado é uma cidade que pertence à California e cujo mayor se chama Norman Mineta.

```
[
  {"?city": "San_Jose_California"}
]
```

- Único resultado comum a ambas as *queries.*

Variáveis (v)

- É possível adicionar variáveis às *queries*, por forma a obter mais resultados.

- O seguinte exemplo:

```
("?city", "inside", "California")
("?city", "mayor", "?name_of_mayor")
```

- Dá-nos como resultado, os nomes das cidades e dos seus mayores, pertencentes à California.

```
[
 {"?city": "San_Francisco_California“, "?name_of_mayor": "Gavin Newsom"},
{"?city": "San_Jose_California", "?name_of_mayor“, "Norman Mineta"}
]
```

- As variáveis em cada dicionário encontram-se relacionadas.

A Pesquisa

- Como visto antes, as *queries* na forma de triplos perfazem uma grande *query* em que os seus resultados são a interseção de todos os resultados individuais dos *triple query*.

- Estas interseções são feitas sequencialmente.

- Um exemplo:
  - bg.query([('?empresa', ‘sede', ‘Lisboa'),
    ('?empresa', ‘setor', ‘Investimento Bancário'),
    ('?empresa', ‘doacao', '?donativo'),
    ('?donativo', ‘destinatario',‘Sr. José'),
    ('?donativo', ‘montante', '?euros')])
  - Resultado:
    - Pretende-se saber todos os bancos de Lisboa que fizeram donativos ao Sr. José e em que montantes.

A Pesquisa

- Outro exemplo:
  - bg.query([('?rel1','with','Britney Spears')
    ('?rel1','with','?person'),
    ('?rel1','end','?year'),
    ('?rel2','with','?person'),
    ('?rel2','start','?year')])
  - Resultado:
    - Pretende-se saber as pessoas que iniciaram uma relação no ano em que terminaram uma relação com a Britney Spears.

Método query()

```
# faz um query ao grafo
# passa uma lista de tuplos (triplos restrição)
# devolve uma lista de dicionários (var:valor)
def query(self, clauses):
    bindings = None # resultado a devolver
    for clause in clauses: # para cada triplo
        # dicionário que associa a variável à sua posição no triplo
        bpos = {}
        qc = [] # lista de elementos a passar ao método triples
        # enumeração do triplo, para aceder ao elemento e sua posição
        for pos, x in enumerate(clause):
            if x.startswith('?'): # para as variáveis
                # usa None para o método triples
                qc.append(None)
                # guarda a posição da variável no triplo (0,1 ou 2)
                bpos[x[1:]] = pos
            else:
                # usa o valor dado para o método triples
                qc.append(x)
        # faz a pesquisa com o triplo acabado de construir
        rows = list(self.triples(qc[0], qc[1], qc[2]))
        ...
```

TripleStore
Uma Implementação

A Inferência

htz@ua.pt

Inferência

- A inferência é o processo de derivar nova informação a partir da informação já existente.
  - Isto é, criar novos triplos entre entidades de forma automática.
- Exemplos de tipos de inferência:
  - Simples e determinístico
    - Sabendo que uma pedra pesa 1kg, pode-se inferir que a mesma pedra pesa 1000g
  - Baseado em regras
    - Se uma determinada pessoa tem menos de 18 anos, pode-se inferir que não pode conduzir.

Inferência

- Classificações
  - Se uma empresa se localiza em Aveiro, esta pode ser classificada como empresa do litoral.
- Julgamentos
  - Se uma pessoa tem mais de 1,80m, pode-se referir à mesma como alta.
- Serviços Online
  - Sabendo o endereço de um restaurante, pode-se usar um geocodificador para achar as suas coordenadas num mapa.

Inferência

- Inferência de novos triplos:

```
# aplica inferencia ao grafo
def applyinference(self,rule):
    queries = rule.getqueries()
    bindings=[]
    for query in queries:
        bindings += self.query(query)
    for b in bindings:
        new_triples = rule.maketriples(b)
        for s, p, o in new_triples:
            self.add(s, p, o)
```

Inferência

- Novo módulo inferencerule.py

```
class InferenceRule:
    """
    Regra base para as inferências
    """
    def getqueries(self):
        return None


    def maketriples(self,binding):
        return self._maketriples(**binding)
```

Inferência

- Nova classe de regras

```
class EnemyRule(InferenceRule):
    def getqueries(self):
        partner_enemy = [('?person', 'enemy','?enemy'),
                         ('?rel', 'with', '?person'),
                         ('?rel', 'with','?partner')]
        return [partner_enemy]


    def _maketriples(self, person, enemy, rel, partner):
        return [(partner, 'enemy', enemy)]
```