UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO (UFERSA)
CENTRO DE CIÊNCIAS EXATAS E NATURAIS (CCEN)
DEPARTAMENTO DE COMPUTAÇÃO
DISCIPLINA: COMPILADORES
ATIVIDADE: TRABALHO PRÁTICO SOBRE ANÁLISE LÉXICA

OBJETIVO: Construir um analisador léxico para reconhecimento de tokens da linguagem OWL2 (Web Ontology Language) no formato Manchester Syntax.

DESCRIÇÃO: A linguagem **OWL** (*Web Ontology Language*) é baseada em **RDF** (*Resource Description Framework*) e **XML** (**eXtensible Markup Language**) para relacionar recursos que têm algo a ver um com o outro na Web. Tudo na Web é um recurso: uma página, um perfil de utilizador de redes sociais, um vídeo, um áudio, um texto, entre outros. O nível de granularidade dos recursos na Web é vasto. Cada recurso pode ser identificado por uma **URI** (*Unified Resource Identifier*). Uma **URL** (*Unified Resource Locator*) é um tipo específico de URI que identifica um endereço único para um determinado recurso na Web. Uma ontologia é um vocabulário que descreve conceitos de uma determinada área do conhecimento. Se esses conceitos forem materializados como recursos na Web, então uma ontologia pode estabelecer relações semânticas entre esses conceitos. Ontologias e dados abertos conectados (*Linked Data*) são as estruturas de dados que ajudam motores de busca, sites de comércio eletrônico e redes sociais a conectarem recursos que têm a ver um com o outro na Web. Agentes inteligentes e serviços Web desenvolvidos com APIs baseadas em Lógica de Descrição (*Description Logics*) podem inferir conhecimento novo a partir de ontologias e dados abertos conectados, e recomendá-lo aos utilizadores de diversos serviços na Web.

OWL é uma linguagem para inferência de máquina, embora ontologistas possam compreendê-la. Abaixo, podemos ver um trecho de uma ontologia de pizzas, o qual descreve as características de uma pizza *Margherita*:

```
<!--  http://www.co-ode.org/ontologies/pizza/pizza.owl#Margherita  -->
<owl:Class rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#Margherita">
      <rdfs:subClassOf rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#NamedPizza"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
          <owl:someValuesFrom
          rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#MozzarellaTopping"/>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
          <owl:someValuesFrom
          rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#TomatoTopping"/>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
          <owl:allValuesFrom>
            <owl:Class>
              <owl:unionOf rdf:parseType="Collection">
                <rdf:Description
                rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#MozzarellaTopping"/>
                <rdf:Description
                rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#TomatoTopping"/>
```

```
            </owl:unionOf>
          </owl:Class>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:label xml:lang="en">Margherita</rdfs:label>
    <rdfs:label xml:lang="pt">Margherita</rdfs:label>
    <skos:altLabel xml:lang="en">Margherita</skos:altLabel>
    <skos:altLabel xml:lang="en">Margherita Pizza</skos:altLabel>
    <skos:prefLabel xml:lang="en">Margherita</skos:prefLabel>
```

Esse mesmo trecho de ontologia pode ser descrito de forma mais palatável para o utilizador humano no formato Manchester Syntax, para descrição de ontologias em Description Logics:

Pizza **that**
hasTopping **some** MozzarellaTopping **and**
hasTopping **some** TomatoTopping **and**
hasTopping **only (**MozzarellaTopping **or**TomatoTopping**)**

Description Logics é a lógica de descrever coisas com base nas relações que elas têm entre si. Uma linguagem baseada em Description Logics é geralmente declarativa, ou seja, não contém métodos ou funções de transformação. Sendo assim, os conceitos de uma ontologia, por exemplo, são descritos sequencialmente, como declarações avulsas. Um motor de inferência (*reasoner*) lê essas declarações e realiza inferências sobre relações implícitas que poderiam conectar ainda mais os conceitos de uma ontologia. Poderíamos também descrever outros conceitos da ontologia usando o mesmo tipo de lógica, por exemplo:

```
<!--  descreve algo que é pizza e tem pelo menos 3 tipos de cobertura -->
```

Pizza **that**
hasTopping **min 3**

```
<!--  descreve algo que é pizza e tem mais de 400 calorias  -->
```

Pizza **that**
hasCaloricContent some **xsd:integer [>="400"]**

```
<!--  descreve algo que é pizza mas não é uma pizza vegetariana  -->
```

Pizza **that**
not VegetarianPizza

```
<!--  descreve algo que é uma cobertura de pizza, que é cobertura de queijo, que é
levemente apimentado e tem a Itália como país de origem  -->
```

PizzaTopping **and**
CheeseTopping **that**
hasSpiciness **some** Mild
and hasCountryOfOrigin **value** Italy1

DESAFIO: Especificar um analisador léxico para a linguagem OWL2 no formato Manchester Syntax, considerando as seguintes especificações:

Palavras reservadas:

- *some, all, value, min, max, exactly, that*
- *not*
- *and*
- *or*
- *Class, EquivalentTo, Individuals, SubClassOf, DisjointClasses (todos sucedidos por ":", que indicam tipos na linguagem OWL)*

Identificadores de classes:

- Nomes começando com letra maiúscula, p.ex.: *Pizza.*
- Nomes compostos concatenados e com iniciais maiúsculas, p.ex.: *VegetarianPizza.*
- Nomes compostos separados por *underline*, p.ex.: *Margherita_Pizza.*

Identificadores de propriedades:

- Começando com "has", seguidos de uma string simples ou composta, p.ex.: *hasSpiciness, hasTopping, hasBase.*
- Começando com "is", seguidos de qualquer coisa, e terminados com "Of", p.ex.: *isBaseOf*, *isToppingOf*.
- Nomes de propriedades geralmente começam com letra minúscula e são seguidos por qualquer outra sequência de letras, p.ex: *ssn*, *hasPhone*, *numberOfPizzasPurchased*.

Símbolos especiais:

- Exemplos: [,], {, }, (,), >, <, e ","

Nomes de indivíduos:

- Começando com uma letra maiúscula, seguida de qualquer combinação de letras minúsculas e terminando com um número. Por exemplo: *Customer1*, *Waiter2*, *AmericanHotPizza1*, etc.

Tipos de dados:

- Identificação de tipos nativos das linguagens OWL, RDF, RDFs ou XML Schema, por exemplo: **owl: real**, **rdfs: domain**, ou **xsd: string**.
- **Observação importante:** os tipos podem ser decompostos em duas partes. A primeira parte será identificada por um token denominado **"NamespaceID"**, p.ex.: xsd: , owl: , rdfs: ou rdf:
- A segunda parte consiste no tipo propriamente dito, p.ex.: integer, real, float, date, etc.
- Um namespace tem apenas três ou quatro letras concatenadas com o símbolo ":"

Cardinalidades:

- Representadas por números inteiros, p.ex.: hasTopping min **3**

O analisador léxico deve produzir como saída os tokens de cada elemento da linguagem especificados em uma tabela de símbolos.

FERRAMENTAS:

- Ambientes de desenvolvimento integrado, p.ex.: VS Code
- Ambientes específicos de modelagem para compiladores, p.ex.: Flex (com C++), PLY (com Python) ou JFlex (com Java)

OBSERVAÇÕES IMPORTANTES:

- Para os grupos que forem sorteados para apresentar no seminário da Unidade 1, o trabalho de **implementação** corresponderá a **50%** da nota da **Unidade 1**, enquanto a **apresentação** corresponderá os outros **50% da nota.**
- Para os grupos que não forem sorteados para esta unidade, o trabalho de implementação corresponderá a **100% da nota.**
- Todos os grupos apresentarão seus trabalhos em alguma unidade. Quem apresentou em uma unidade fica desonerado de apresentar nas outras duas unidades.
- O trabalho poderá ser realizado **individualmente** ou em grupos de **2 componentes.**
- O trabalho será testado com outros arquivos de teste diferentes do que está apresentado no final deste documento. A lógica de identificação dos lexemas deverá funcionar para outros testes.
- Data de entrega: **10/12/2024**, até às **23h59.**

REFERÊNCIAS

Horridge, M., Drummond, N., Goodwin, J., Rector, A. L., Stevens, R., & Wang, H. (2006, November). The Manchester OWL syntax. In *OWLed* (Vol. 216). Disponível online em: https://ceur-ws.org/Vol-216/submission_9.pdf

Parr, T. (2013). The definitive ANTLR 4 reference. *The Definitive ANTLR 4 Reference*, 1-326.

McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. *W3C recommendation*, *10*(10), 2004. Disponível online em: https://static.twoday.net/71desa1bif/files/W3C-OWL-Overview.pdf

CÓDIGO DE TESTE:

Class: Customer

EquivalentTo:
Person
and (purchasedPizza **some** Pizza)
and (hasPhone **some** xsd:string)

Individuals:
Customer1,
Customer10,
Customer2,
Customer3,
Customer4,
Customer5,
Customer6,
Customer7,
Customer8,
Customer9

Class: Employee

SubClassOf:
Person
and (ssn **min** 1 xsd:string)

Individuals:
Chef1,
Manager1,
Waiter1,
Waiter2

Class: Pizza

SubClassOf:
hasBase **some** PizzaBase,
hasCaloricContent **some** xsd:integer

DisjointClasses:
Pizza, PizzaBase, PizzaTopping

```
    Individuals:
        CustomPizza1,
        CustomPizza2

Class: CheesyPizza

    EquivalentTo:
        Pizza
         and (hasTopping some CheeseTopping)

    Individuals:
        CheesyPizza1

Class: HighCaloriePizza

    EquivalentTo:
        Pizza
         and (hasCaloricContent some xsd:integer [>= 400])

Class: InterestingPizza

    EquivalentTo:
        Pizza
         and (hasTopping min 3 PizzaTopping)

Class: LowCaloriePizza

    EquivalentTo:
        Pizza
         and (hasCaloricContent some xsd:integer [< 400])

Class: NamedPizza

    SubClassOf:
        Pizza

Class: AmericanaHotPizza

    SubClassOf:
```

NamedPizza,
hasTopping **some** JalapenoPepperTopping,
hasTopping **some** MozzarellaTopping,
hasTopping **some** PepperoniTopping,
hasTopping **some** TomatoTopping

DisjointClasses:
AmericanaPizza, MargheritaPizza, SohoPizza

Individuals:
AmericanaHotPizza1,
AmericanaHotPizza2,
AmericanaHotPizza3,
ChicagoAmericanaHotPizza1

Class: AmericanaPizza

SubClassOf:
NamedPizza,
hasTopping **some** MozzarellaTopping,
hasTopping **some** PepperoniTopping,
hasTopping **some** TomatoTopping

DisjointClasses:
AmericanaHotPizza, MargheritaPizza, SohoPizza

Individuals:
AmericanaPizza1,
AmericanaPizza2

Class: MargheritaPizza

SubClassOf:
NamedPizza,
hasTopping **some** MozzarellaTopping,
hasTopping **some** TomatoTopping,
hasTopping **only**
(MozzarellaTopping **or** TomatoTopping)

DisjointClasses:

AmericanaHotPizza, AmericanaPizza, SohoPizza

Individuals:
MargheritaPizza1,
MargheritaPizza2

Class: SohoPizza

SubClassOf:
NamedPizza,
hasTopping **some** MozzarellaTopping,
hasTopping **some** OliveTopping,
hasTopping **some** ParmesanTopping,
hasTopping **some** TomatoTopping,
hasTopping **only**
(MozzarellaTopping **or** OliveTopping **or** ParmesanTopping **or** TomatoTopping)

DisjointClasses:
AmericanaHotPizza, AmericanaPizza, MargheritaPizza

Individuals:
SohoPizza1,
SohoPizza2

Class: SpicyPizza

EquivalentTo:
Pizza
and (hasTopping **some** (hasSpiciness **value** Hot1))

Class: VegetarianPizza

EquivalentTo:
Pizza
and (hasTopping **only**
(CheeseTopping **or** VegetableTopping))

Class: PizzaBase

DisjointClasses:

Pizza, PizzaBase, PizzaTopping

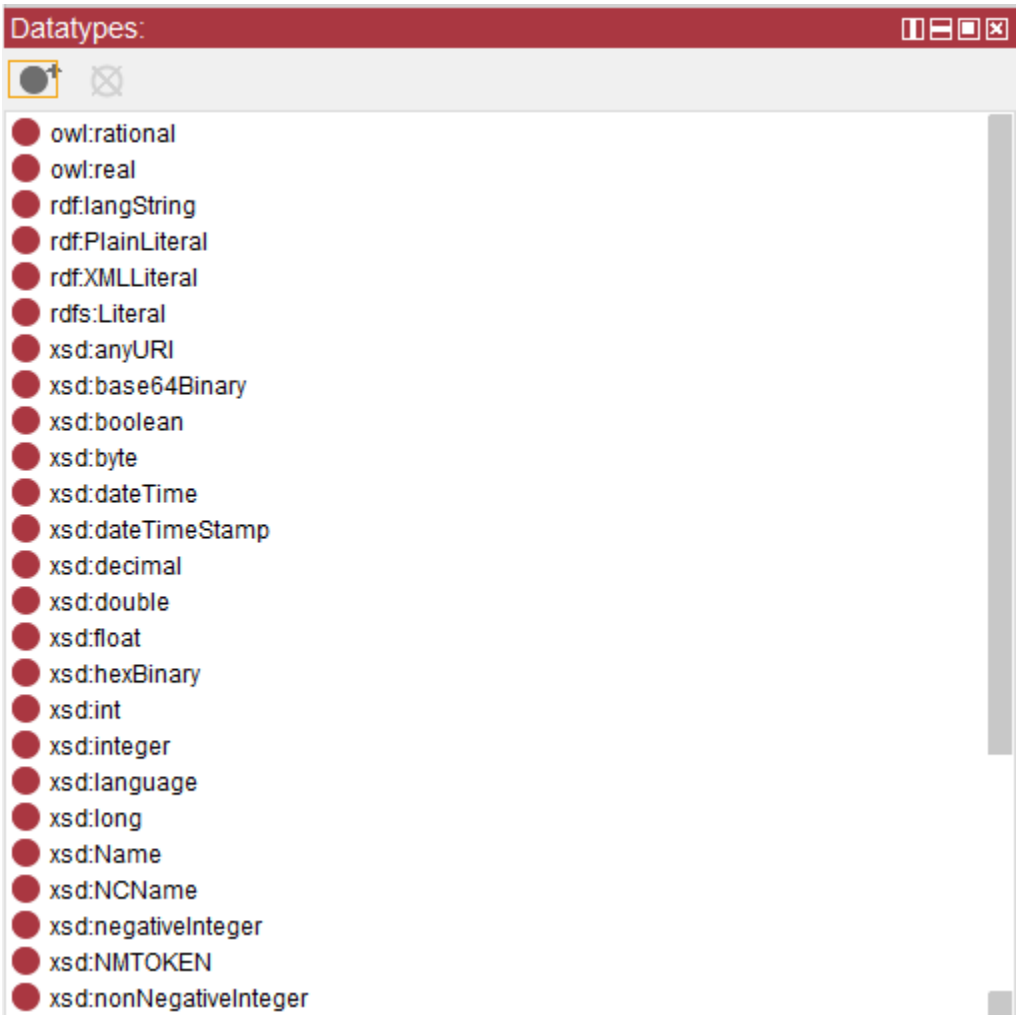
Class: PizzaTopping

DisjointClasses:
Pizza, PizzaBase, PizzaTopping

Class: Spiciness

EquivalentTo:
{Hot1 , Medium1 , Mild1}

TIPOS E NAMESPACES




xsd:language
xsd:long
xsd:Name
xsd:NCName
xsd:negativeInteger
xsd:NMTOKEN
xsd:nonNegativeInteger
xsd:nonPositiveInteger
xsd:normalizedString
xsd:positiveInteger
xsd:short
xsd:string
xsd:token
xsd:unsignedByte
xsd:unsignedInt
xsd:unsignedLong
xsd:unsignedShort

Compilador simples para *Manchester Syntax*

O seguinte projeto tem como objetivo implementar um compilador simples para um conjunto restrito da Manchester Syntax (usada na OWL2).

Sumário

Analisador Léxico

O analisador léxico até agora é capaz de reconhecer alguns lexemas que foram pré-definidos, atribuindo um *token* a cada um. O código principal foi deslocado para o arquivo .y, que chamam o lexer para analisar cada lexema e passá-lo para o parser. O armazenamento feito em uma *unordered list* realizado no arquivo .l é legado da versão antiga que apenas reconhecia os lexemas. Posteriormente, pensa-se em reutilizar ele para exibir, novamente, a contagem de cada ocorrência de token.

Analisador Sintático

O analisador sintático verifica se a gramática da linguagem permanece correta para o reconhecimento de 6 tipos de classes diferentes, sendo essas:

- Classe Primária
- Classe Definida
- Classe com axioma de fechamento
- Classe Aninhada
- Classe Enumerada
- Classe Coberta

Veja a descrição das classes que foi usada clicando aqui.
É possível acessar os arquivos de teste utilizados no diretório "tests".

Analisador Semântico

O analisador semântico analisa um conjunto de regras definida para este projeto para que o sentido se mantenha nas declarações de classe. Para este corte da *OWL Manchester Syntax*, exercitou-se 3 tipos de análises semânticas:

- Análise de precedência dos operadores de cabeçalho
    - Precedência das palavras-chave
    - Axioma de fechamento
- Verificação estática de tipos por coerção
    - Restrição na declaração de xsd:integer e xsd:float
    - Cardinalidade após operadores *min*, *max* e *exactly*
- Verificação estática de tipos por sobrecarga
    - Identificação de *Data properties* e *Object properties*

Veja a descrição das análises que foi usada clicando aqui.

Como usar

Aqui um tutorial simples de uso.
Primeiramente, os requisitos para rodar o programa:

- WSL (*Windows Subsystem for Linux*), Sistema Operacional Linux instalado no PC ou uma VM (máquina virtual) que rode alguma distro do Linux
- Compilador g++
- Bison
- Flex
- CMake
- Visual Studio Code (opcional, para rodar as tasks de build)

Para a instalação das ferramentas, siga o passo a passo
Com os requisitos atendidos, segue o passo a passo do que fazer:

1. Clone o repositório na sua máquina com git clone [url] ou baixe-o
2. Abre o diretório no VS Code
3. Em "Terminal", execute as seguintes três tarefas em sequência em "Executar a tarefa..."
    - Execute a tarefa chamada clean build, ela irá criar uma pasta build se não existir, e se existir vai limpar ela.
    - Execute a tarefa chamada cmake, ela irá gerar arquivos necessários para a compilar o compilador no diretório "build".
    - Execute a tarefa chamada make, ela irá gerar o executável no diretório "build"
4. Com isso feito, o executável foi gerado no diretório build. Navegue até ele via terminal.
5. Ao entrar no diretório build, rode o executável com o comando "./mansyncomp" no terminal. Ele aguardará você digitar as classes para poder identificá-las.

- Invés de digitar manualmente, é possível passar um arquivo como entrada. Para fazer isso, precisará chamar o executável "./mansyncomp ../nome-do-arquivo.txt" ou "./Mansyncomp < ../nome-do-arquivo.txt".
- Caso queira verificar os testes usados, pode usar o comando "./mansyncomp ../tests/nome-do-arquivo.txt"

Descrição dos tipos de classes

Para fins didáticos, vamos definir "expr" como:

```
Unset
    PROPRIEDADE QUANTIFICADOR NOME_DA_CLASSE
```

Classe Primitiva

É uma classe de base na qual suas propriedades podem ser herdadas, mas indivíduos que possuem as mesmas propriedades sem herdar desta não serão classificados como tal. Além disso, é definida com um "SubClassOf" desde que seja apenas ele. (Ver mais)
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    SubClassOf:
    CLASSE,
    expr
    // outras expr separadas por ","
```

Classe Definida

Similar a classe primitiva, mas além de transferir suas características para seus indivíduos por herança também permite que outros indivíduos com as mesmas propriedades sejam classificadas como esta. É definida com um "EquivalentTo".
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE
```

```
    EquivalentTo:
    CLASSE
    and expr
    // outras expr separadas por "and"
```

Classe com axioma de fechamento

Sendo uma classe primitiva ou definida, o que também a caracteriza como classe com axioma de fechamento é a presença do axioma de fechamento (🙄). Isso restringe a classe a ter APENAS aquelas características.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    SubClassOf:
    CLASSE,
    PROPRIEDADE QUANTIFICADOR CLASSE,
    PROPRIEDADE QUANTIFICADOR OUTRA_CLASSE,
    PROPRIEDADE ONLY (CLASSE or OUTRA_CLASSE)
```

Classe Aninhada

Também se aplica tanto a classes primitivas quanto definidas. A característica principal é, na tripla que define uma expressão, ter outra expressão no lugar da classe. Vale ressaltar que esse aninhamento pode se extender por muito.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    EquivalentTo:
    CLASSE
```

```
    and (PROPRIEDADE QUANTIFICADOR (PROPRIEDADE
QUANTIFICADOR CLASSE))
```

Classe Enumerada

Para esse tipo de classe, pode-se definir uma classe a partir de seus possíveis indivíduos.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    EquivalentTo:
    {INDIVIDUO1, INDIVIDUO2, INDIVIDUO3}
```

Classe Coberta

Similar a classe enumerada, mas é uma variação na qual a classe é uma superposição de suas filhas. Sendo assim, indivíduos pertencentes a mãe também vão, necessariamente, pertencer a uma das filhas.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    EquivalentTo:
    CLASSE or OUTRA_CLASSE or OUTRA_CLASSE
```

Descrição das análises semânticas

Análise de precedência de operadores de cabeçalho

Nessa primeira análise, buscamos garantir que a declaração das palavras-chave seguem uma determinada ordem, sendo essa: Class -> EquivalentTo -> SubClassOf -> DisjointClasses -> Individuals. Lembrando que uma classe deve ter pelo menos

SubClassOf ou EquivalentTo, mas caso possua as duas o EquivalentTo tem precedência maior, assim como DisjointClasses e Individuals, que são opcionais mas, quando ocorrem, seguem essa precedência.
Confira o exemplo:

```
Unset
    // Classe correta
    Class: NOME_DA_CLASSE

        EquivalentTo:
        CLASSE

    // Classe correta
    Class: NOME_DA_CLASSE

        SubClassOf:
        CLASSE

        DisjointClasses:
        OUTRA_CLASSE

        INDIVIDUALS:
        INDIVIDUO

    // Classe com erro
    Class: CLASSE_ERRO

        SubClassOf:
        CLASSE

        EquivalentTo:
        CLASSE
```

Também temos a precedência do axioma de fechamento para uma propriedade que deve ser declarado somente após as classes relacionadas à propriedade.

```
Unset
    // ...
    SubClassOf:
    CLASSE,
    PROPRIEDADE QUANTIFICADOR EXEMPLO,
    PROPRIEDADE QUANTIFICADOR OUTROEXEMPLO,
    PROPRIEDADE only (EXEMPLO or OUTROEXEMPLO)

    // O fragmento abaixo resulta em erro
    SubClassOf:
    CLASSE,
    PROPRIEDADE only (EXEMPLO or OUTROEXEMPLO),
    PROPRIEDADE QUANTIFICADOR EXEMPLO,
    PROPRIEDADE QUANTIFICADOR OUTROEXEMPLO
```

Verificação estática de tipos por coerção

Nesta análise, o objetivo é garantir que tipos que requerem uma delimitação de intervalo, no caso o xsd:integer e xsd:float, possuam essa restrição após sua declaração, envolta em colchetes, com um operador relacional e uma cardinalidade. Também, garante-se que um tipo inteiro receberá um inteiro como parâmetro e um ponto-flutuante recebera um ponto-flutuante.

```
Unset
    // ...
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE QUANTIFICADOR xsd:integer[>= 400])
    and (PROPRIEDADE QUANTIFICADOR xsd:float[>= 20.5])

    // O fragmento abaixo resulta em erro
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE QUANTIFICADOR xsd:float)
```

Além, é claro, da garantia que um os operadores min, max e exactly sejam sucedidos de uma cardinaldide.

```
Unset
    // ...
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE min 1 CLASSE)

    // O fragmento abaixo resulta em erro
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE exactly CLASSE)
```

Verifição estática de tipos por sobrecarga

A última análise feita identifica e diferencia uma Data property de uma Object property para auxiliar o usuário na hora de debugar. Uma Data property é definida pela tripla (propriedade, quantificador, datatype) enquanto que o Object property se define pela tripla (propriedade, quantificador, classe).

```
Unset
    // ...
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE QUANTIFICADOR xsd:string) // Data
property

    // ...
    SubClassOf:
    CLASSE,
    PROPRIEDADE QUANTIFICADOR CLASSE // Object property
```

Instalação das ferramentas

- Visual Studio Code (Disponível aqui)

- Linux (Instalação do WSL disponível aqui)

Após isso, abra o terminal e realize os seguintes comandos:

```
Unset
    sudo apt update
    sudo apt upgrade
    sudo apt install g++
    sudo apt install make cmake -y
    sudo apt install flex bison -y
```

Voltar para tutorial.

Notas do autor

- Depois de assistir as apresentações, notei que o que eu fiz está um pouco abaixo mediano e percebi que usar e abusar da recursão na gramática, bem como fragmentar ele em pequenas partes para permitir um reuso ao máximo.
- Ele ainda não está perfeito, mas consegue atender aos exemplos usados.

Compilador simples para *Manchester Syntax*

O seguinte projeto tem como objetivo implementar um compilador simples para um conjunto restrito da Manchester Syntax (usada na OWL2).

Sumário

Analisador Léxico

O analisador léxico até agora é capaz de reconhecer alguns lexemas que foram pré-definidos, atribuindo um *token* a cada um. O código principal foi deslocado para o arquivo .y, que chamam o lexer para analisar cada lexema e passá-lo para o parser.

O armazenamento feito em uma *unordered list* realizado no arquivo .l é legado da versão antiga que apenas reconhecia os lexemas. Posteriormente, pensa-se em reutilizar ele para exibir, novamente, a contagem de cada ocorrência de token.

Analisador Sintático

O analisador sintático verifica se a gramática da linguagem permanece correta para o reconhecimento de 6 tipos de classes diferentes, sendo essas:

- Classe Primária
- Classe Definida
- Classe com axioma de fechamento
- Classe Aninhada
- Classe Enumerada
- Classe Coberta

Veja a descrição das classes que foi usada clicando aqui.
É possível acessar os arquivos de teste utilizados no diretório "tests".

Analisador Semântico

O analisador semântico analisa um conjunto de regras definida para este projeto para que o sentido se mantenha nas declarações de classe. Para este corte da *OWL Manchester Syntax*, exercitou-se 3 tipos de análises semânticas:

- Análise de precedência dos operadores de cabeçalho
    - Precedência das palavras-chave
    - Axioma de fechamento
- Verificação estática de tipos por coerção
    - Restrição na declaração de xsd:integer e xsd:float
    - Cardinalidade após operadores *min*, *max* e *exactly*
- Verifição estática de tipos por sobrecarga
    - Identificação de *Data properties* e *Object properties*

Veja a descrição das análises que foi usada clicando aqui.

Como usar

Aqui um tutorial simples de uso.
Primeiramente, os requisitos para rodar o programa:

- WSL (*Windows Subsystem for Linux*), Sistema Operacional Linux instalado no PC ou uma VM (máquina virtual) que rode alguma distro do Linux
- Compilador g++

- Bison
- Flex
- CMake
- Visual Studio Code (opcional, para rodar as tasks de build)

Para a instalação das ferramentas, siga o passo a passo
Com os requisitos atendidos, segue o passo a passo do que fazer:

1. Clone o repositório na sua máquina com git clone [url] ou baixe-o
2. Abre o diretório no VS Code
3. Em "Terminal", execute as seguintes três tarefas em sequência em "Executar a tarefa..."
    - Execute a tarefa chamada clean build, ela irá criar uma pasta build se não existir, e se existir vai limpar ela.
    - Execute a tarefa chamada cmake, ela irá gerar arquivos necessários para a compilar o compilador no diretório "build".
    - Execute a tarefa chamada make, ela irá gerar o executável no diretório "build"
4. Com isso feito, o executável foi gerado no diretório build. Navegue até ele via terminal.
5. Ao entrar no diretório build, rode o executável com o comando "./mansyncomp" no terminal. Ele aguardará você digitar as classes para poder identificá-las.
    - Invés de digitar manualmente, é possível passar um arquivo como entrada. Para fazer isso, precisará chamar o executável "./mansyncomp ../nome-do-arquivo.txt" ou "./Mansyncomp < ../nome-do-arquivo.txt".
    - Caso queira verificar os testes usados, pode usar o comando "./mansyncomp ../tests/nome-do-arquivo.txt"

Descrição dos tipos de classes

Para fins didáticos, vamos definir "expr" como:

```
Unset
    PROPRIEDADE QUANTIFICADOR NOME_DA_CLASSE
```

Classe Primitiva

É uma classe de base na qual suas propriedades podem ser herdadas, mas indivíduos que possuem as mesmas propriedades sem herdar desta não serão classificados como tal. Além disso, é definida com um "SubClassOf" desde que seja apenas ele. (Ver mais)
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    SubClassOf:
    CLASSE,
    expr
    // outras expr separadas por ","
```

Classe Definida

Similar a classe primitiva, mas além de transferir suas características para seus indivíduos por herança também permite que outros indivíduos com as mesmas propriedades sejam classificadas como esta. É definida com um "EquivalentTo".
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    EquivalentTo:
    CLASSE
    and expr
    // outras expr separadas por "and"
```

Classe com axioma de fechamento

Sendo uma classe primitiva ou definida, o que também a caracteriza como classe com axioma de fechamento é a presença do axioma de fechamento (🙄). Isso restringe a classe a ter APENAS aquelas características.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    SubClassOf:
    CLASSE,
    PROPRIEDADE QUANTIFICADOR CLASSE,
```

```
    PROPRIEDADE QUANTIFICADOR OUTRA_CLASSE,
    PROPRIEDADE ONLY (CLASSE or OUTRA_CLASSE)
```

Classe Aninhada

Também se aplica tanto a classes primitivas quanto definidas. A característica principal é, na tripla que define uma expressão, ter outra expressão no lugar da classe. Vale ressaltar que esse aninhamento pode se extender por muito.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    EquivalentTo:
    CLASSE
    and (PROPRIEDADE QUANTIFICADOR (PROPRIEDADE
QUANTIFICADOR CLASSE))
```

Classe Enumerada

Para esse tipo de classe, pode-se definir uma classe a partir de seus possíveis indivíduos.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    EquivalentTo:
    {INDIVIDUO1, INDIVIDUO2, INDIVIDUO3}
```

Classe Coberta

Similar a classe enumerada, mas é uma variação na qual a classe é uma superposição de suas filhas. Sendo assim, indivíduos pertencentes a mãe também vão, necessariamente, pertencer a uma das filhas.
Sua estrutura geral se define por:

```
Unset
Class: NOME_DA_CLASSE

    EquivalentTo:
    CLASSE or OUTRA_CLASSE or OUTRA_CLASSE
```

Descrição das análises semânticas

Análise de precedência de operadores de cabeçalho

Nessa primeira análise, buscamos garantir que a declaração das palavras-chave seguem uma determinada ordem, sendo essa: Class -> EquivalentTo -> SubClassOf -> DisjointClasses -> Individuals. Lembrando que uma classe deve ter pelo menos SubClassOf ou EquivalentTo, mas caso possua as duas o EquivalentTo tem precedência maior, assim como DisjointClasses e Individuals, que são opcionais mas, quando ocorrem, seguem essa precedência.
Confira o exemplo:

```
Unset
    // Classe correta
    Class: NOME_DA_CLASSE

        EquivalentTo:
        CLASSE

    // Classe correta
    Class: NOME_DA_CLASSE

        SubClassOf:
        CLASSE

        DisjointClasses:
        OUTRA_CLASSE
```

```
        INDIVIDUALS:
        INDIVIDUO

    // Classe com erro
    Class: CLASSE_ERRO

        SubClassOf:
        CLASSE

        EquivalentTo:
        CLASSE
```

Também temos a precedência do axioma de fechamento para uma propriedade que deve ser declarado somente após as classes relacionadas à propriedade.

Unset

```
    // ...
    SubClassOf:
    CLASSE,
    PROPRIEDADE QUANTIFICADOR EXEMPLO,
    PROPRIEDADE QUANTIFICADOR OUTROEXEMPLO,
    PROPRIEDADE only (EXEMPLO or OUTROEXEMPLO)

    // O fragmento abaixo resulta em erro
    SubClassOf:
    CLASSE,
    PROPRIEDADE only (EXEMPLO or OUTROEXEMPLO),
    PROPRIEDADE QUANTIFICADOR EXEMPLO,
    PROPRIEDADE QUANTIFICADOR OUTROEXEMPLO
```

Verificação estática de tipos por coerção

Nesta análise, o objetivo é garantir que tipos que requerem uma delimitação de intervalo, no caso o xsd:integer e xsd:float, possuam essa restrição após sua declaração, envolta em colchetes, com um operador relacional e uma cardinalidade.

Também, garante-se que um tipo inteiro receberá um inteiro como parâmetro e um ponto-flutuante recebera um ponto-flutuante.

```
Unset
    // ...
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE QUANTIFICADOR xsd:integer[>= 400])
    and (PROPRIEDADE QUANTIFICADOR xsd:float[>= 20.5])

    // O fragmento abaixo resulta em erro
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE QUANTIFICADOR xsd:float)
```

Além, é claro, da garantia que um os operadores min, max e exactly sejam sucedidos de uma cardinaldide.

```
Unset
    // ...
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE min 1 CLASSE)

    // O fragmento abaixo resulta em erro
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE exactly CLASSE)
```

Verifição estática de tipos por sobrecarga

A última análise feita identifica e diferencia uma Data property de uma Object property para auxiliar o usuário na hora de debugar. Uma Data property é definida pela tripla (propriedade, quantificador, datatype) enquanto que o Object property se define pela tripla (propriedade, quantificador, classe).

```
Unset
    // ...
    EquivalentTo:
    CLASSE
    and (PROPRIEDADE QUANTIFICADOR xsd:string) // Data
property

    // ...
    SubClassOf:
    CLASSE,
    PROPRIEDADE QUANTIFICADOR CLASSE // Object property
```

Instalação das ferramentas

- Visual Studio Code (Disponível aqui)
- Linux (Instalação do WSL disponível aqui)

Após isso, abra o terminal e realize os seguintes comandos:

```
Unset
    sudo apt update
    sudo apt upgrade
    sudo apt install g++
    sudo apt install make cmake -y
    sudo apt install flex bison -y
```

Voltar para tutorial.

Notas do autor

- Depois de assistir as apresentações, notei que o que eu fiz está um pouco abaixo mediano e percebi que usar e abusar da recursão na gramática, bem como fragmentar ele em pequenas partes para permitir um reuso ao máximo.
- Ele ainda não está perfeito, mas consegue atender aos exemplos usados.