UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO (UFERSA)
CENTRO DE CIÊNCIAS EXATAS E NATURAIS (CCEN)
DEPARTAMENTO DE COMPUTAÇÃO
DISCIPLINA: COMPILADORES
PROFESSOR: PATRÍCIO DE ALENCAR SILVA
PROJETO: ANALISADOR SINTÁTICO PARA TEXTUAL ONTOLOGY LANGUAGE

OBJETIVO: Projetar um analisador sintático para verificação da estrutura de declaração de operadores da linguagem TONTO (*Textual Ontology Language*).

CONTEXTUALIZAÇÃO: Conforme estudado na primeira unidade da disciplina de Compiladores, a análise léxica no processo de compilação consiste na identificação dos tipos de lexemas que compõem uma linguagem de máquina, seja esta procedural ou declarativa. Linguagens para definição de ontologias são declarativas, i.e., apenas informam o que as coisas são, mas não contêm instruções de transformação de entradas em saídas, e.g., métodos ou funções. A linguagem OWL (Web Ontology Language), baseada em XML (eXtensible Markup Language), é usada para a definição de ontologias computacionais interpretáveis por agentes inteligentes artificiais. Já a linguagem OntoUML também é usada para a definição de ontologias, mas difere de OWL por ser gráfica e apropriada à interpretação e discussão humanas. Além disso, dispõe de um conjunto de estereótipos (ou metaclasses) para definição de modelos de ontologias com fundamentação filosófica. A linguagem Tonto (Textual Ontology Language) é a versão declarativa e textual de OntoUML, sendo o estudo de caso atualmente estudado na referida disciplina de Compiladores [1-2]. Na fase de análise léxica, criamos um subconjunto da linguagem Tonto com regras específicas de escrita de lexemas para representar classes, relações, instâncias e demais elementos da linguagem. Na fase de análise sintática, teremos de orientar um ontologista a escrever as estruturas que definem cada construto da linguagem de acordo com regras bem definidas de escopo e ordem [3].

DESCRIÇÃO DO PROBLEMA: Projetar um **analisador sintático** para a linguagem TONTO para verificação da corretude da especificação textual de uma ontologia nos seguintes casos:

1. **Declaração de pacotes:** uma especificação Tonto é dividida em pacotes. Cada pacote define uma "visão" de uma ontologia. Vários pacotes ou visões compõem uma ontologia completa. Pacotes funcionam como *namespaces* ou contêineres lógicos de classes, seus respectivos atributos e relações. Cada modelo em Tonto precisa começar com a declaração de um pacote, conforme o exemplo:

```
package myPackage
```

2. **Declaração de classes**: Uma classe é declarada com um estereótipo de classe em OntoUML (e.g., **kind**, **subkind**, **role**, **phase** etc.) seguida de seu nome conforme regras definidas no analisador léxico. A declaração de uma classe pode conter atributos próprios. No exemplo abaixo, temos os atributos ***name*** e ***birthdate***, cujos tipos específicos deverão ser tratados por outro conjunto de regras de produção da gramática do analisador sintático. Uma vez que a classe é declarada, poderá ser usada na modelagem de relações. Uma classe pode conter ainda declarações internas de relações, conforme exemplo posterior.

```
kind Person {
    name: string
    birthDate: date {const}
}

phase Child specializes Person
```

3. **Declaração de tipos de dados**: Tonto contém seis tipos de dados nativos (***number***, ***string***, ***boolean***, ***date***, ***time*** e ***datetime***). É possível construir ou derivar outros tipos mais complexos a partir desses tipos básicos. No exemplo abaixo, temos um novo tipo de dado para representar um endereço (**Address**), o qual contém seus próprios atributos, cada um formatado de acordo com um tipo nativo de Tonto.

```
datatype Address {
    street: string
    number: int
}
```

4. **Declaração de classes enumeradas:** Certas classes podem ser criadas com um tipo finito e pré-definido de instâncias. Essas são as chamadas classes enumeradas (**enumerated classes**). Exemplos de classes enumeradas incluem: dias da semana, planetas de um sistema planetário ou o menu de um restaurante. No exemplo abaixo, temos uma classe enumerada nomeada como **EyeColor**, a qual contém apenas quatro instâncias (ou indivíduos). A declaração da classe é precedida pela palavra reservada **enum**. Note que o analisador léxico construído nesta disciplina previa uma regra diferente para a nomeação de indivíduos, cujos nomes deveriam terminar com um número.

```
enum EyeColor { Blue, Green, Brown, Black }
```

5. **Generalizações (*Generalization sets*):** Ontologias são vocabulários construídos com base em taxonomias, i.e., árvores de conceitos unidos por relações de herança. Conceitos do topo da árvore são mais abstratos, enquanto os conceitos mais próximos da base são mais concretos. As taxonomias também podem ser organizadas em grupos de generalizações, visto que cada conceito pode derivar múltiplas taxonomias, cada uma segundo um aspecto. No exemplo abaixo, vemos duas formas de se declarar uma generalização em Tonto. Na primeira linha, temos um conjunto de generalização denominado ***PersonAgeGroup***, o qual é declarado como tal pela palavra reservada ***genset***. Nesta pequena taxonomia, há uma classe-mãe denominada Person e duas classes filhas, denominadas **Child** e **Adult**, respectivamente. Neste caso, devemos considerar que as três classes já foram declaradas anteriormente no código, mas aqui, especificamente, estão sendo agrupadas em uma pequena taxonomia de generalização. As palavras reservadas **disjoint** e complete que antecedem a palavra ***genset*** denotam que a taxonomia é completa (ou seja, a classe **Adult** nesta classificação de pessoa por idade só tem duas filhas: **Child** e **Adult**) e disjunta (ou seja, as subclasses **Child** e **Adult** não compartilham indivíduos – sua interseção é o vazio). Note que a palavra reservada ***where*** reforça a estrutura. Uma outra forma de declarar a mesma estrutura é abrindo uma outra estrutura limitada por chaves e declarando a classe-mãe com a palavra reservada

general, e as respectivas classes filhas com a palavra reservada ***specifics***. Nesse mesmo caso, é possível acrescentar restrições de disjunção ou completude (e.g. ***disjoint***, ***overlapping***, ***complete*** ou ***incomplete***).

```
disjoint complete genset PersonAgeGroup where Child, Adult specializes Person

genset PersonAgeGroup {
    general Person
    specifics Child, Adult
}
```

6. **Declarações de relações:** Em Tonto, uma relação pode ser declarada interna ou externamente a uma classe. No exemplo abaixo, a classe **University** (estereotipada com a palavra ***kind***) contém uma declaração interna de relação. Neste exemplo, a relação não tem nome, mas aparece apenas com seu estereótipo (**componentOf**) e seus elementos básicos (cardinalidade e a simbologia gráfica de agregação **<>--**). Há também a classe que representa a imagem da relação, i.e., **Department**. O domínio da relação é a própria classe **University**. Na segunda parte do exemplo, uma relação é declarada fora do escopo de qualquer classe. Neste caso, a relação é estereotipada por **@mediation** e denotada pela palavra reservada **relation**. Note que a relação não tem um rótulo ou nome específico. Ainda assim, a declaração está completa por definir o domínio da relação (i.e., **EmploymentContract**) e a imagem da relação (i.e., **Employee**). A declaração é completada pelas cardinalidades e pelo símbolo especial --.

```
// Internal relation
kind University {
    @componentOf [1] <>-- [1..*] Department
}

// External relation
@mediation relation EmploymentContract [1..*] -- [1] Employee
```

REQUISITOS: O trabalho deverá ser elaborado e avaliado de acordo com os seguintes critérios:

- O projeto será parcialmente automatizado com o uso da ferramenta **BISON** e suas aplicações em C++ ou Python. Lembrem-se de que esse framework opera com base em análise ascendente.
- **Testes** estarão disponíveis em: https://github.com/patricioalencar/Compiladores_UFERSA.
- Os trabalhos deverão estar no GitHub, com documentação apropriada.
- A saída da análise sintática deverá compreender duas visualizações: (1) tabela de síntese contendo um resumo dos construtos encontrados (e.g.: quantos e quais pacotes, quais classes estão em cada pacote, quais relações estão em cada classe e quais são externas, quantas e quais declarações de tipos etc.; e (2) relatório de erros da ontologia, com sugestões de tratamento.
- O **vídeo** explicativo de 05 (cinco) minutos continua **obrigatório**.

REFERÊNCIAS

1. Coutinho, M. L. (2024). Leveraging LLMs in text-based ontology-driven conceptual modeling. Disponível online em: https://www.utwente.nl/en/eemcs/fois2024/resources/papers/coutinho-leveraging-llms-in-text-based-ontology-driven-conceptual-modeling.pdf
2. Coutinho, M. L., Almeida, J. P. A., Sales, T. P., & Guizzardi, G. (2024). A Textual Syntax and Toolset for Well-Founded Ontologies. In *14th International Conference on Formal Ontology in Information Systems, FOIS 2024* (pp. 208-222). IOS.
3. Lenke, M., Tonto: A Textual Syntax for OntoUML – A textual way for conceptual modeling. Disponível online em: https://matheuslenke.github.io/tonto-docs/

ANEXO - MODELO COMPLETO DA ONTOLOGIA EM ONTOUML