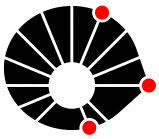

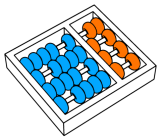

Frontend para ferramenta de ordenação de arquivos SEG-Y

F. M. Rappa *H. C. Yviquel*

Relatório Técnico - IC-PFG-23-07
Projeto Final de Graduação
2023 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Frontend para ferramenta de ordenação de arquivos SEG-Y

Frederico Meletti Rappa* Hervé Cédric Yviquel†

Resumo

Este trabalho representa um relatório do projeto realizado em parceria com o LSC, cujo objetivo é desenvolver uma ferramenta capaz de ordenar e filtrar dados sísmicos representados em formato SEG-Y utilizando computação de alto desempenho.

A partir do trabalho desenvolvido, foi possível desenvolver uma linguagem de query, que pode ser utilizada para operar sobre os dados, a partir de seu parsing e conversão para SQL a fim da sua utilização em outra ferramenta desenvolvida em C++ e Rust.

1 Introdução

Este relatório descreve o projeto de desenvolvimento de uma Domain Specific Language que representa operações em arquivos SEG-Y, um padrão de arquivos binários comumente utilizados para representar dados sísmicos coletados periodicamente em áreas de interesse para geofísicos. Apesar da existência de softwares que permitam a visualização destes arquivos, a necessidade de manipulação de forma simples de um grande volume de dados motivou o desenvolvimento do projeto *SegOrd*.

SegOrd, cujo escopo não será detalhado neste relatório, utiliza de bibliotecas de computação de alto desempenho, como OpenMP para operar sobre conjuntos de dados sísmicos de forma paralela. Nele, arquivos SEG-Y são convertidos para Apache Parquet [1] – formato tabular que permite armazenamento e manipulação eficiente de dados – e manipulados a partir de comandos inseridos pelo usuário. Para permitir uma interface entre usuários e o projeto, foi criada uma linguagem de query simples capaz de definir as operações realizadas sobre os dados, "SEG-Y Query Plan". A partir dos comandos inseridos em um arquivo ou no terminal, a query é validada e transformada em um programa equivalente em SQL, para que possa ser utilizada para a operação nos dados em Parquet pelo *SegOrd*. O compilador foi desenvolvido em Python, utilizando Sly [2], biblioteca para escrita de parsers LALR(1).

2 Referencial Teórico

Esta seção apresenta conceitos para melhor entendimento do projeto.

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP
†Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP

2.1 Domain Specific Language

Kosar et al. [3] define uma Domain Specific Language (DSL) como uma linguagem desenvolvida para conter apenas sintaxe relevante a conceitos de domínio específico. De acordo com [4], DSLs permitem melhor representação de conceitos de negócio, e podem permitir o desenvolvimento de código-fonte de melhor qualidade e manutenibilidade.

2.2 SEG-Y

SEG-Y é um formato de arquivos desenvolvido em 1973 pela *Society of Exploration Geophysicist*, como maneira de representar dados sísmicos. Um arquivo corresponde a uma sequência de *traces*, cada um correspondendo a um conjunto de dados coletados, que podem descrever valores em duas ou três dimensões. Além dos dados, arquivos em formatos SEG-Y contêm três header: um textual, que permite uma descrição legível do arquivo; um binário, que contém dados referentes a todo o arquivo; e *trace headers*, que contêm informações relevantes ao dado, como coordenadas ou momento da captura, por exemplo.

O tamanho e formato de cada dado, no que se refere a valores numéricos e número de bits, é informado nos headers binários, de modo que não é possível definir a dimensão dos dados antes da leitura do arquivo. Além disso, a depender do tipo de dados, como em duas ou três dimensões, os dados são organizados em ordens distintas.

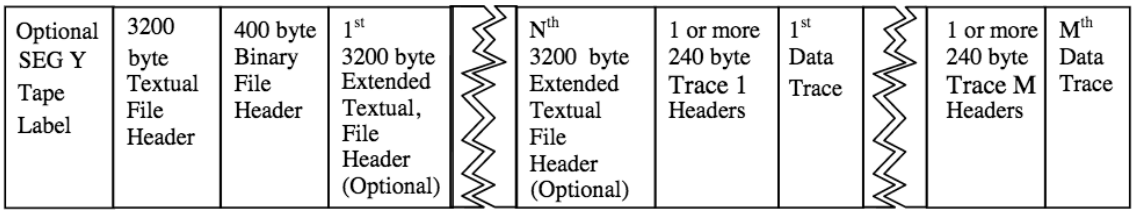

Figura 1: Estrutura de um arquivo SEG-Y

3 Objetivos

Este projeto visa desenvolver uma Domain Specific Language capaz de descrever operações de manipulação em um conjunto de arquivos SEG-Y, como filtragem e ordenação. Mais especificamente, busca-se desenvolver uma linguagem de query de entendimento e uso simples, e um compilador que a possa converter em operações SQL que possam ser utilizadas na manipulação dos dados sísmicos.

4 Desenvolvimento

4.1 SEG-Y Query Plan

Conforme mencionado, um dos objetivos principais da construção da linguagem foi sua simplicidade, de modo que usuários conseguissem interagir com *SegOrd* não programati-

camente, e sem a necessidade de conhecimento de sintaxe SQL. Desse modo, a linguagem possui apenas 10 keywords, 13 símbolos e quatro tipos de dados: dados numéricos – que contêm números inteiros e de ponto flutuante –, booleanos, caracteres e strings. Além disso, os identificadores – que representam headers SEG-Y – e keywords são *case insensitive* e a query independe de identação, para maior simplicidade na escrita. Comentários são trechos entre os sinais "`/-`"e "`-/`"e não influenciam o comportamento.

A linguagem possui duas operações permitidas, filtragem e ordenação, que podem ser dispostas em qualquer ordem entre si. A primeira pressupõe um conjunto de regras que devem ser aplicadas aos dados, de modo que sejam selecionados ao final da operação. Neste contexto, é possível validar igualdade, desigualdade e intervalos entre valores dos tipos mencionados. A segunda operação, por sua vez, define a sequência de headers pela qual os dados serão ordenados. O trecho abaixo representa uma query válida na linguagem.

```
1 /- Example query -/
2
3 ORDER:
4     year, day, hour;
5
6 FILTER:
7     cdp = 1000.6 OR
8     CDP_X >= 100 AND
9     CDP_Y < -500;
```

Listing 1: Exemplo de SEG-Y Query Plan

Nota-se que esta query contêm ambas as operações possíveis. A operação de ordenação, nas linhas 3 e 4, pode ser entendida da seguinte forma: os dados devem ser ordenados de forma crescente pelo header `year`. Caso um conjunto de dados contenha o mesmo valor de `year`, devem ser ordenados de forma crescente por `day` e, caso coincidam em `year` e `day`, ordenados por `hour`. Os valores que serão ordenados devem respeitar a condição estabelecida na operação de seleção, ou filtragem, entre as linhas 6 e 9, que restringe os dados para os que tenham header `CDP` de valor igual a 1000.6 ou `CDP_X >= 100` e `CDP_Y < -500`.

É válido ressaltar que a linguagem proposta não permite operações matemáticas, ou entre caracteres e strings. Esta escolha foi feita uma vez que as otimizações decorrentes como propagação de constantes e *Dead Code Elimination*, por exemplo, já são feitas de maneira eficiente pelo processo de parsing do SQL na manipulação dos dados em formato Parquet.

4.2 SEG-Y Query Plan Parser

Conforme exposto, o compilador foi desenvolvido utilizando Sly, biblioteca Python para construção de parsers. A ferramenta foi escolhida pela robustez e simplicidade de uso, uma vez que permite a representação de keywords como expressões regulares ou regras como código semelhante a BNF.

4.2.1 Estrutura

A estrutura do compilador desenvolvido é semelhante a frontends de compiladores tradicionais, como mostrado na Figura 2.

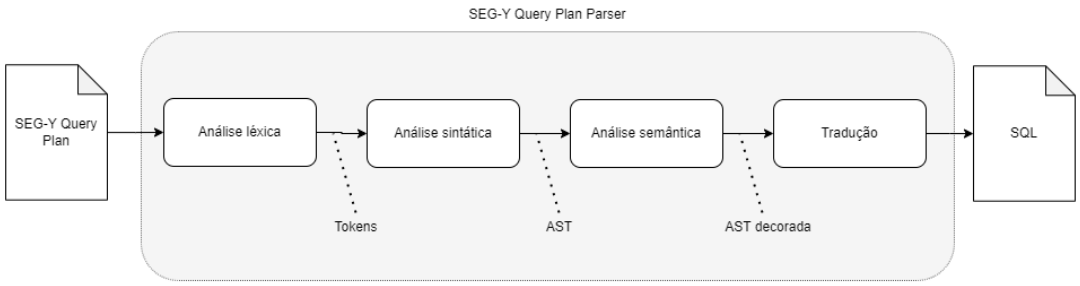

Figura 2: Estrutura do frontend desenvolvido

O compilador recebe uma string representando uma query em SEG-Y Query Plan, a decompõe em tokens, analisa sua estrutura, valida sua semântica e devolve uma query em SQL. As seções a seguir descrevem em detalhes as etapas do processo.

4.3 Análise léxica

A análise léxica corresponde à decomposição e categorização do texto de entrada em um conjunto de *tokens*. [5] define um token como os símbolos elementares de uma gramática . Para o desenvolvimento de SEG-Y Query Plan, os tokens foram divididos de acordo com as categorias expostas na Tabela 1.

Definição	**Representação**	**Descrição**	**Exemplo**
Identificadores	`ID`	representam os *headers* a que serão filtrados os arquivos. Corresponde a um conjunto de caracteres alfanuméricos e "_", iniciados por uma letra.	`CDP`, `TRACE_SAMPLE_COUNT`, `Unassigned1`
Operadores	`PLUS`, `MINUS`, `LT`, `LE`, `GT`, `GE`, `EQ`, `NE`	representam operações booleanas e numéricas	`+`, `=`, `<=`
Palavras chave	`FILTER`, `ORDER`, `AND`, `NOT`, `TRUE`, `FALSE`, `INCL`, `DESC`, `IN`, `RANGE`	representam exatamente as palavras-chave utilizadas	`ORDER`, `range`, `True`
Constantes	`INT_CONST`, `REAL_CONST`, `CHAR_CONST`, `STRING_LITERAL`	representam dados, como números e sequências de caracteres	`10`, `-40.6`, `"1979-05-23"`, `'a'`
Delimitadores	`LPAREN`, `RPAREN`, `COMMA`, `COLON`, `SEMI`	representam sinais de pontuação utilizados na sintaxe	`(`, `:`, `,`

Tabela 1: Tokens possíveis gerados da análise léxica

Desse modo, caso o texto possa ser dividido de acordo com as regras, o processo de compilação continua. Caso contrário, um erro é retornado ao usuário.

4.4 Análise sintática

A partir da identificação correta dos tokens no texto provido, é feita a análise semântica, que identifica se sua organização coincide coma sintaxe definida da linguagem. Esta pode ser definida pelas seguintes regras EBNF:

⟨*program*⟩ ::= ⟨*filter*⟩ ⟨*order*⟩
| ⟨*order*⟩ ⟨*filter*⟩
| ⟨*filter*⟩
| ⟨*order*⟩
| ⟨*empty*⟩

⟨*order_id*⟩ ::= ⟨*id*⟩
| ⟨*id*⟩ DESC

⟨*filter*⟩ ::= FILTER COLON ⟨*expression*⟩ SEMI

⟨*order*⟩ ::= ORDER COLON ⟨*id_list*⟩ SEMI

⟨*id_list*⟩ ::= ⟨*order_id*⟩
| ⟨*id_list*⟩ COMMA ⟨*order_id*⟩

⟨*expression*⟩ ::= ⟨*unary_expression*⟩
| ⟨*expression*⟩ LT ⟨*expression*⟩
| ⟨*expression*⟩ LE ⟨*expression*⟩
| ⟨*expression*⟩ GT ⟨*expression*⟩
| ⟨*expression*⟩ GE ⟨*expression*⟩
| ⟨*expression*⟩ EQ ⟨*expression*⟩
| ⟨*expression*⟩ NE ⟨*expression*⟩
| ⟨*expression*⟩ AND ⟨*expression*⟩
| ⟨*expression*⟩ OR ⟨*expression*⟩
| ⟨*range_expression*⟩

⟨*unary_expression*⟩ ::= ⟨*primary_expression*⟩
| PLUS ⟨*unary_expression*⟩
| MINUS ⟨*unary_expression*⟩
| NOT ⟨*unary_expression*⟩

⟨*primary_expression*⟩ ::= ⟨*id*⟩
| ⟨*constant*⟩
| LPAREN ⟨*expression*⟩ RPAREN

⟨*unary_expression_range*⟩ ::= ⟨*unary_expression*⟩
| ⟨*unary_expression*⟩ INCL

⟨*range_expression*⟩ ::= IN RANGE LPAREN ⟨*unary_expression_range*⟩ COMMA ⟨*unary_expression_range*⟩
RPAREN

⟨*id*⟩ ::= ID

⟨*constant*⟩ ::= INT_CONST
| REAL_CONST
| CHAR_CONST
| STRING_LITERAL
| TRUE
| FALSE

A partir desta gramática, é possível construir as seguintes regras:

- `program`: Representa todo o Query Plan. Um programa pode conter uma operação de filtragem, uma de ordenação, ambas, dispostas em qualquer ordem, ou nenhuma, representando um arquivo vazio;
- `filter`: A operação de filtragem deve definir as regras que selecionam os dados dos arquivos SEG-Y. Desse modo, a expressão `expression` deve conter um conjunto de expressões booleanas a partir dos valores dos headers. A sintaxe também permite uma expressão de intervalo, que filtra os conjuntos de dados de acordo com valores máximos e mínimos de um header.

- `expression`: representa uma expressão unária, binária ou expressão de intervalo;
- `unary_expression`: conjunto operador unário – `NOT`, `+` ou `-`) – e uma expressão, ou uma `primary_expression`;
- `primary_expression`: representa um identificador, constante ou expressão entre parênteses;
- `range_expression`: identifica um intervalo, delimitado por dois valores numéricos. Um limite pode ser incluído no intervalo com o keyword `INCL`. Desse modo, a expressão `coord_X IN RANGE(-100, 90)` é equivalente a $-100 < coord\_X < 90$, enquanto a expressão `coord_Y IN RANGE(0, 0.5 INCL)` equivale a $0 < coord\_Y \leq 0.5$;
- `order_id`: A operação de ordenação pressupõe uma lista de identificadores, que define a sequência de headers a partir da qual os dados devem ser ordenados. Por padrão, ordena o conjunto de dados de forma ascendente, de modo que para que se ordene de forma descendente, deve ser informada a keyword `DESC`;
- `id_list`: representa a lista de identificadores em uma expressão de ordenação, separadas por vírgula;
- `id`: regra que representa tokens terminais de identificadores;
- `constant`: regra que representa terminais de constantes;

A sequência de tokens é validada de acordo com a estrutura exposta, de modo que seja possível dividir o código em trechos sintáticamente corretos. A partir da divisão, é construída uma *Abstract Syntax Tree* (AST), grafo em que cada nó contém informações sobre a gramática do texto.

4.5 Análise semântica

A partir da AST construída na etapa anterior, é feita a análise semântica da query: cada nó da árvore é percorrido e, se possível, definido um tipo, isto é, valores inteiros ou reais e nós que representam operações sobre estes recebem tipos numéricos, e o mesmo é feito para constantes e operações booleanas.

Entre as validações feitas, pode-se citar verificar se os tipos de expressões correspondem aos argumentos, se os tipos de expressões binárias correspondem ou se os identifiacdores são válidos, isto é, se correspondem aos headers possíveis de arquivos SEG-Y. Desse modo, expressões como `filter: (cdp_X < (cdp_Y > 1000));`, apesar de sintaticamente corretas de acordo com a gramática especificada, retornam mensagens de erro. Da mesma forma, o uso de identificadores diferentes dos headers permitidos também o faz.

4.6 Tradução

Na última etapa, de tradução, temos um *Query Plan* com termos válidos, sintática- e semanticamente correto. A AST é percorrida e cada nó é mapeado para um termo equivalente

em SQL, de modo que, ao final do processo, haverá uma string que represente a query equivalente à originalmente inserida.

5 Resultados

O exemplo a seguir representa uma query válida. Ela contêm ambas as operações possíveis, filtragem e ordenação de dados, além de operadores entre números, booleanos, uso de parênteses para denotar precedência, constantes inteiras, reais e de string, além de operadores de intervalo, de ordem descendente e comentários.

```
filter:
    cdp < 1000 and
    (xline >= 100 or
    xline < -500) and
    not (iline != 10) or
    uint1 in range(0.0 incl, 0.5) and
    uint2 = "2020";

/-
This comment will be ignored
-/

order:
    cdp,
    iline desc,
    uint1;
```

Listing 2: Exemplo de SEG-Y Query Plan

A partir do texto anterior, a análise léxica gera a sequência de tokens a seguir. Nota-se que comentários, representados pelos caracteres "`/-`"e "`-/`", são ignorados na geração dos tokens e da query final.

```
FILTER COLON
ID(cdp) LT INT_CONST(1000) AND
LPAREN ID(xline) INT_CONST(100) OR
ID(xline) GT INT_CONST(-500) RPAREN AND
NOT LPAREN ID(iline) NE INT_CONST(10) RPAREN OR
ID(uint1) IN RANGE LPAREN REAL_CONST(0.0) INCL COMMA REAL_CONST(0.5) RPAREN
    AND
ID(uint2) EQ STRING_LITERAL("2020") SEMI

ORDER COLON
ID(cdp) COMMA
ID(iline) DESC COMMA
ID(uint2) SEMI
```

Listing 3: Lista de tokens gerada

A partir da gramática descrita anteriormente, e da sequência de tokens exposta, é possível construir a *Abstract Syntax Tree* na Figura 3. Nela, cada nó tem seu nome em negrito e campos em itálico. Em seguida, é percorrida e verificada se od nós `ID` têm nomes válidos e se os tipos coincidem.

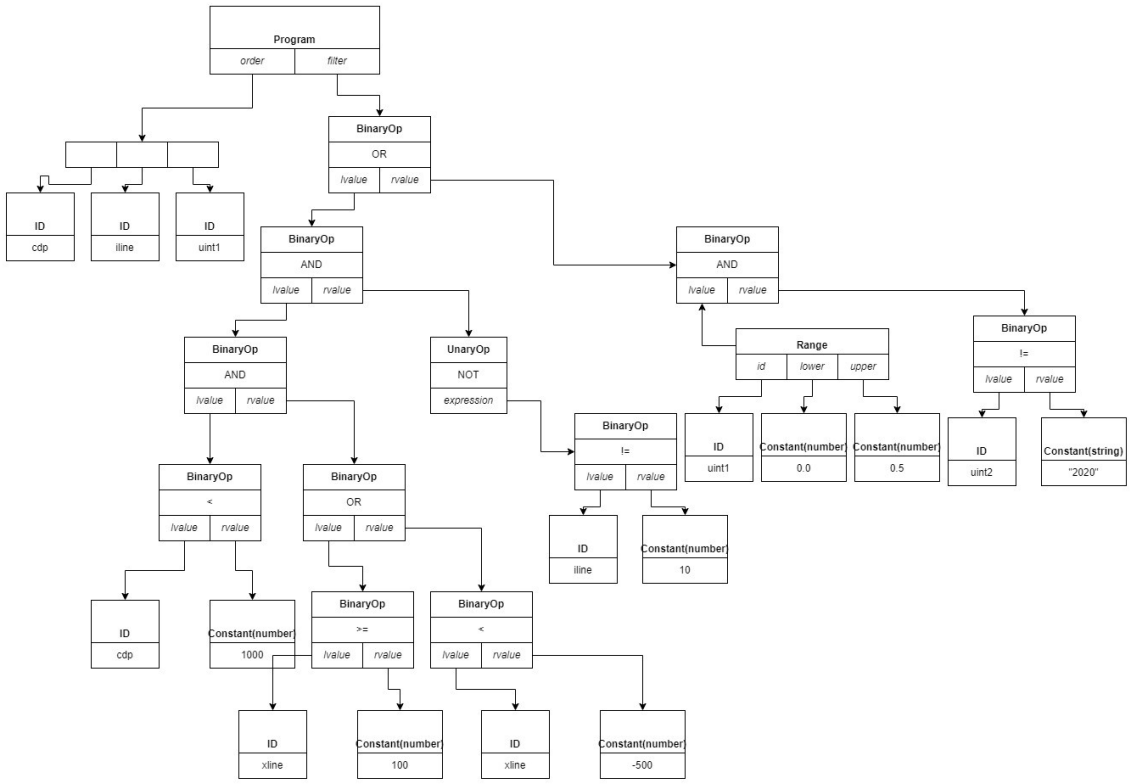

Figura 3: AST construída a partir da query anterior

Por fim, como o trecho é semanticamente válido, é gerado a seguinte query em SQL. O nome da tabela `Table1` é padrão para o funcionamento das operações em Parquet. Nota-se que é equivalente à query original, com a operação `filter:` equivalente a `SELECT` e `order:` a `ORDER BY`.

```sql
SELECT *
FROM Table1
WHERE cdp < 1000
  AND (xline >= 100
       OR xline < -500)
  AND NOT (iline <> 10)
  OR (0.0 <= uint1
      AND uint1 < 0.5)
  AND uint2 = '2020'
ORDER BY cpd,
         iline DESC,
```

```
12          uint1;
```

Listing 4: Query SQL resultante

6 Conclusões e Trabalhos Futuros

Os resultados obtidos neste projeto mostraram que a criação de uma Domain Specific Language (DSL) para manipulação de arquivos SEG-Y foi bem-sucedida. A linguagem de consulta desenvolvida, chamada SEG-Y Query Plan, permite que os usuários especifiquem operações de filtragem e ordenação de forma simples e intuitiva, sem a necessidade de conhecimento prévio de sintaxe SQL.

O compilador desenvolvido, utilizando a biblioteca Sly em Python, mostrou-se eficiente na análise léxica, sintática e semântica da linguagem. Ele é capaz de decompor a consulta em tokens, analisar sua estrutura e validar sua semântica. Além disso, o compilador converte a consulta em SEG-Y Query Plan para uma equivalente em SQL, que pode ser utilizada para manipular os dados sísmicos em formato Parquet.

Trabalhos futuros podem incluir a expansão da DSL para suportar mais operações e funcionalidades, como funções personalizadas e outras operações. Outra área de pesquisa interessante é a integração da DSL e *SegOrd* com ferramentas de visualização de dados, permitindo aos usuários obter visualizações interativas dos resultados das consultas. Isso poderia facilitar a análise e a interpretação dos dados sísmicos, auxiliando os geofísicos em suas pesquisas e tomadas de decisão.

Em resumo, este projeto foi um primeiro passo na criação de uma DSL para manipulação de arquivos SEG-Y, oferecendo aos usuários uma forma simples e poderosa de trabalhar com dados sísmicos.

Referências

[1] Deepak Vohra and Deepak Vohra. Apache parquet. *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*, pages 325–335, 2016.

[2] Beazley. Sly (sly lex yacc), 2022.

[3] Tomaž Kosar, Pablo E Martı, Pablo A Barrientos, Marjan Mernik, et al. A preliminary study on various implementation approaches of domain-specific language. *Information and software technology*, 50(5):390–405, 2008.

[4] Jasper Denkers. A longitudinal field study on creation and use of domain-specific languages in industry. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1152–1155, 2019.

[5] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers: principles, techniques, and tools*, volume 2. Addison-wesley Reading, 2007.