Nomes:	Bruno Rafael Aricó	**Nº USP:**	8125459
	Isabela Blücher		9298170
	Luís Felipe de Melo Costa Silva		9297961
	Nícolas Nogueira Lopes da Silva		9277541

Relatório do Exercício-Programa I de MAC0239

1 Problema

Tivemos que implementar um sistema de provas para o cálculo proposicional usando os tableaux semânticos (com as expansões α e β) para provar validade ou não de um sequente, da seguinte forma:

$$A_1, A_2, ..., A_n \vdash B, \text{ onde:}$$

As fórmulas $A_i, i = 1, 2, ..., n$ são as premissas e B é a consequência. No caso de a fórmula ser inválida, devemos mostrar um contraexemplo.

2 Solução

O principal problema encontrado foi a análise léxica da entrada. Uma linguagem para as fórmulas foi definida no enunciado. Os parênteses são usados em todas as fórmulas e os átomos são do tipo p, q (letras minúsculas), etc. Os operadores foram definidos como:

- .I. = $\rightarrow$
- .A. = $\wedge$
- .O. = $\vee$
- .N. = $\neg$

No entanto, não sabíamos como fazer um programa que entendesse essa linguagem. Para nos ajudar, usamos um analisador análogo ao FLEX para o C, o ANTLR.

2.1 ANTLR

O ANTLR (ANother Tool for Language Recognition) é um gerador de códigos para leitura, processamento, execução ou tradução de textos estruturados ou arquivos binários.

É bastante usado para construir linguagens, ferramentas e arcabouços. A partir de uma gramática, ele gera um *parser* que consegue construir e percorrer *parse trees*.

A motivação para usá-lo é que o ANTLR consegue gerar imagens a partir da linguagem e da entrada. Segue exemplo:

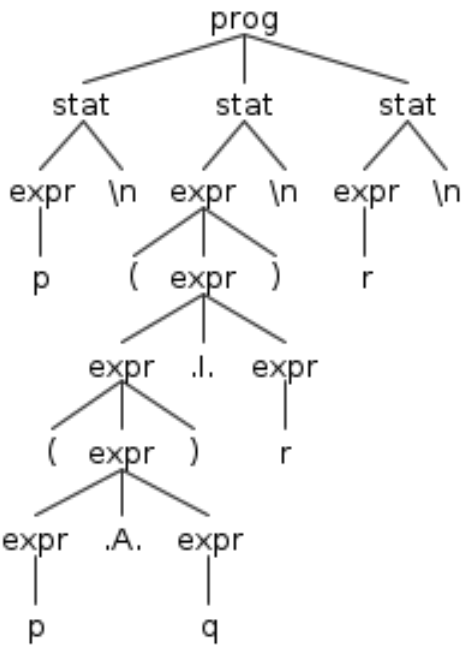

Figure 1: Gerada com o comando `$ java org.antlr.v4.gui.TestRig Expr prog -gui input`, onde `input` é o arquivo com a entrada (\entradas\invalidas\8.txt)

2.2 Implementação

O arquivo EvalVisitor foi criado por nós e nele estão definidas as regras de expansão do Tableau. Atacamos o problema da seguinte forma:

Usamos o ANTLR como base principal do nosso trabalho. A partir dos pedidos da ferramenta, criamos nossa gramática, definida no arquivo **Expr.g4**. Ele gera os arquivos necessários para o reconhecimento da linguagem. Os arquivos gerados tem a forma Expr*.java.

A ferramenta cria a árvore com os componentes do analisador léxico derivados de gramática. Ele faz o ExprBaseVisitor que define algumas funções que serão executadas quando visitamos alguma estrutura na árvore gerada por ele. Um exemplo disso é o **Progrule**, que executa o código quando visitamos a estrutura raiz da nossa gramática (visitProgrule).

Cada estrutura de **expr** dessa árvore possui um valor associado no *ParseTreeProperty* VALORES (é uma estrutura que associa algo a um nó da árvore) que pode ser **T** (true)

ou **F** (false).

Todas as estruturas **expr** foram colocadas em uma pilha. Todas as premissas **expr** foram valoradas como **T** e a conclusão B como **F**. O valor da estrutura desempilhada foi devolvido para a saída.

De acordo com a estrutura visitada (recursivamente, com as funções geradas pelo ANTLR) pôde-se definir o que fazer quando:

- se entra em situação de operação entre dois elementos ($\rightarrow$, $\wedge$ e $\vee$ - **visitOp2Atom**).
- se entra em situação de negação ($\neg$ - **visitOpNot**).
- se visita um elemento átomo (**visitAtom**).
- é encontrado um parênteses (**visitParen**).
- é encontrada uma **expr** comum (**visitPrintExpr**).

Quando uma expansão α é realizada:

- valoramos cada ramo da árvore na estrurura filha em VALORES (do **ParseTreeProperty**) de acordo com a tabela da expansão β;
- empilhamos a estrutura filha da direita na pilha global;
- retornamos o valor da visita na estrutura filha da esquerda.

Quando uma oexpansão β é realizada:

- copiamos a pilha global;
- valoramos cada ramo da árvore na estrurura filha em VALORES (do **ParseTreeProperty**) de acordo com a tabela da expansão β;
- colocamos em uma variável temporária o resultado da visita no ramo esquerdo;
- substituimos a pilha pelá cópia anterior;
- colocamos em outra variável temporária o resultado da visita no ramo direito ;
- devolvemos um AND entre as duas saídas.

Quando temos uma operação de negação:

- valoramos a única expressão filha como o valor contrário ao valor associado a essa expressão em VALORES;
- visitamos a expressão filha.

Quando temos um parêntese ou expressão sozinha:

- valoramos a expressão filha em VALORES como valor da expressão atual;
- visitamos a expressão filha.

Quando chegamos em um átomo, verificamos se aquele átomo já possui alguma valor na **TreeMap** ATOMOS. Caso já possua algum valor nessa **TreeMap**, o programa verifica se o valor associado àquela estrutura no VALORES e no ATOMOS são diferentes. No caso em que são diferentes então encontramos uma contradição (o que não descarta que a expressão é inválida) e então temos um ramo fechado e retornamos um **T**. Caso não sejam diferentes então vemos se existe algo na pilha, associando um valor de retorno para a tirada de um elemento da pilha. Ao terminar, se o valor de retorno for F então encontramos uma situação em que o sequente é inválido, exibimos todos os átomos e seus valores em ATOMOS e encerramos o programa.