Trabalho de Compiladores - Documentação

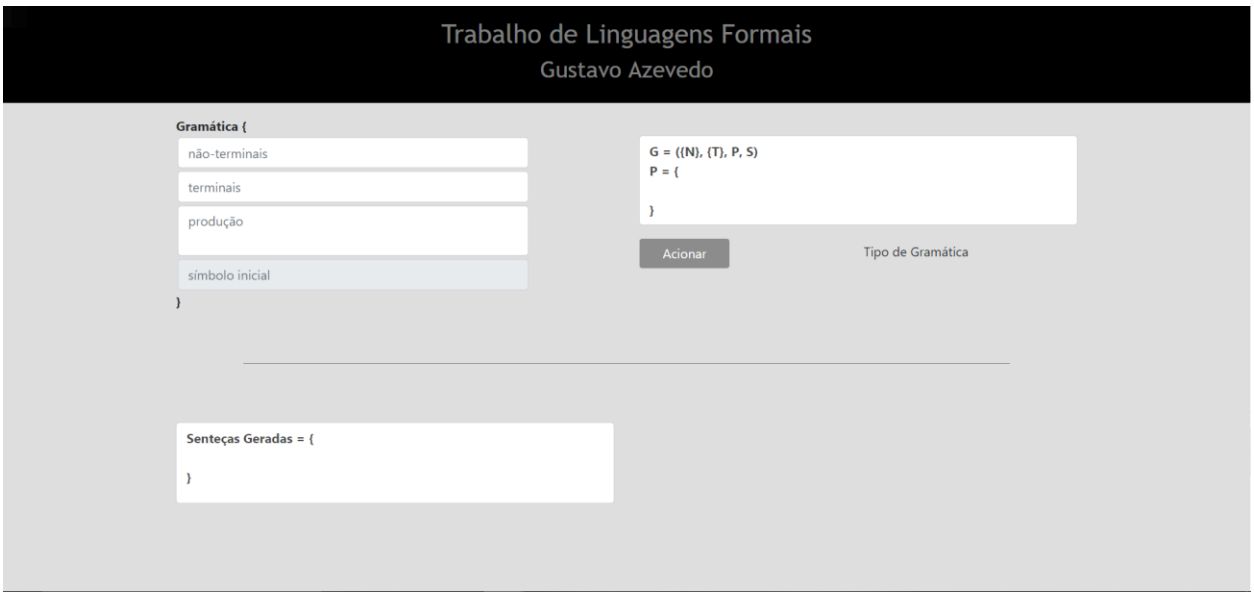
Gustavo Azevedo (m91999) e Gabriel Konzen Silveira (m73780)

O manipulador de gramáticas foi desenvolvido em Java Script e utilizou das seguintes ferramentas: Bootstrap, Pure CSS e JQuery. A interface foi desenvolvida em HTML e CSS.

Versão 1.0:

1. Interface

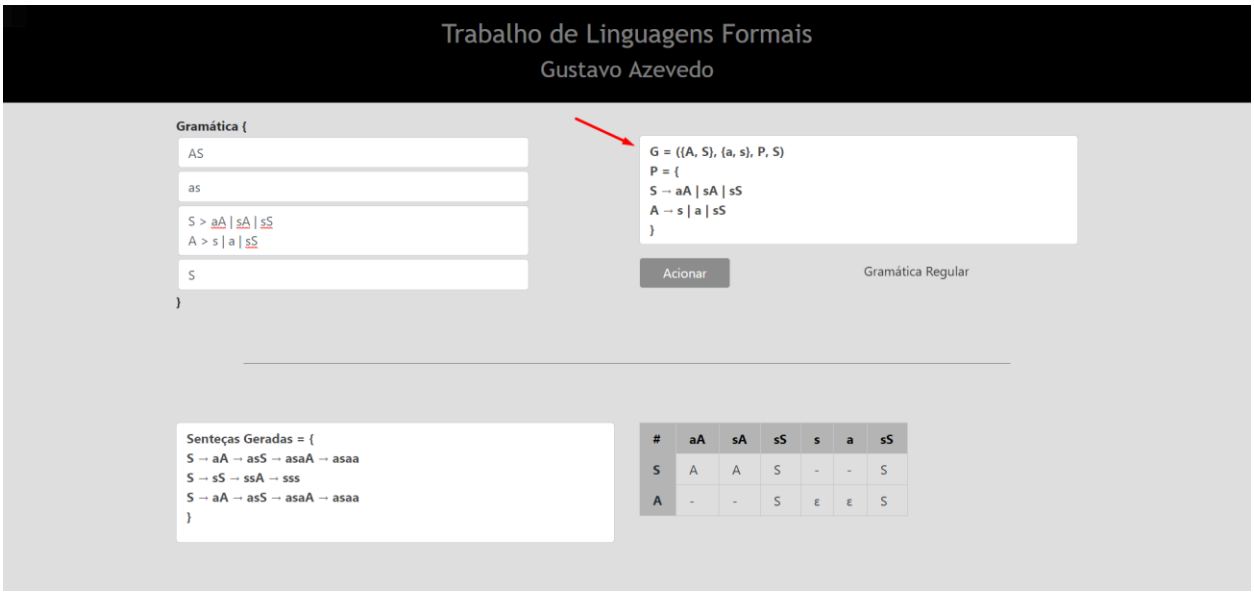
Segue abaixo uma imagem do layout da interface.



Nessa segunda imagem, temos o layout completo, após terem sido escritos os campos e rodado o programa.

No quadrante superior esquerdo, se encontra as áreas de input para se inserir os dados da gramática.

Na parte superior do quadrante superior direito, se encontra a escrita da gramática inserida.



Pode-se observar que no quadrante superior direito, na área inferior, indica qual o tipo de gramática que ele reconheceu.

No quadrante inferior da esquerda, se encontra três sentenças randomicamente geradas para a gramatica inserida.

E por final, no quadrante inferior direito, se encontra a tabela do autômato finito para a gramática inserida.

2. Funcionamento

Embaixo segue o código fonte da criar a gramática sendo inserida, ele acontece automaticamente a mediada que os dados são inseridos.

```
//Esses sets pegam os dados escritos nas caixas a
 esquerda, processam eles,
//e escrevem a gramática na caixa da direita
function setNT(nter) {
  let aux = '';
  let aws = nter.split('');
  for (let key in aws) {
    key = aws[key];
    if (/[A-Z]/g.test(key)) {
      aux += key + ', ';
    }
  }
  $('#n1').html(aux.slice(0, -2));
}

function setTer(ter) {
  let aux = '';
  let aws = ter.split('');
  for (let key in aws) {
    key = aws[key];
    if (/[a-z]/g.test(key)) {
      aux += key + ', ';
    }
  }
  $('#t1').html(aux.slice(0, -2));
}

function setSI(si) {
  $('#s1').html(si);
  $('#si').val(si);
}

function setProd(prod) {
  prod = prod.replace(/>/g, '→');
  prod = prod.replace(/&/g, 'ε');
  prod = prod.replace(/(?:\r\n|\r|\n)/g, '<br>');
  $('#p1').html(prod);
}
```

Abaixo se encontra o código para reconhecer o tipo de gramática, que roda quando o botão é clicado.

```
function runProgram() {
  //teste para descobrir o tipo de gramática
  let [gr, glc, gsc, gi] = new Array(4).fill(true);

//limpa os dados para mais facilmente poder utiliza-los n
o futuro
  let prod = $('#prod').val();
  prod = prod.replace(/ /g, '');
  let linhas = prod.split('\n');
  let esquerda = [];
  let direita = [];
  let dirSimbolo = [];
  for (const key of linhas) {
    let aux = key.split('>');
    direita.push(aux[1]);
    esquerda.push(aux[0]);
  }
  for (const key of direita) {
    let aux = key.split('|');
    for (const i of aux) {
      dirSimbolo.push(i);
    }
  }
  //fim da limpa dos dados


//Procura se possui terminais na esquerda, se sim, não é
 GR nem GLC
  if (/([a-z])(.*>)/g.test(prod)) {
    gr = false;
    glc = false;
  }


//Olha o lado esquerdo, se tiver mais de um caractere, nã
o é GR nem GLC
  for (const key in esquerda) {
    for (const iterator of esquerda[key]) {
      if (iterator.length > 1) {
        gr = false;
        glc = false;
      }
    }
  }


//Procura no lado direito se possui um terminal sozinho o
u um terminal seguido de NT,
  //se fugir dessa regra, não é GR.
```

```
//Procura também por vazios, se encontrar, não é GLC nem
 GSC
  for (const iter of dirSimbolo) {
    for (const key in iter) {
      if (
        /[A-Z]/g.test(iter.charAt(key)) &&
        !/[a-z]/g.test(iter.charAt(key - 1))
      ) {
        gr = false;
      } else if (
        /[a-z]/g.test(iter.charAt(key)) &&
        iter.charAt(key + 1) == '' &&
        iter.charAt(key - 1) != ''
      ) {
        gr = false;
      }
    }
    if (iter.includes('&')) {
      glc = false;
      gsc = false;
    }
  }


//testa se o lado esquerdo possui o mesmo tamanho ou meno
r que o lado direito,
  //se não for, não é GSC
  if (gsc) {
    for (const key of linhas) {
      let [esq, dir] = key.split('>');
      let aux = dir.split('|');
      for (const i of aux) {
        if (i.length < esq.length) {
          gsc = false;
          break;
        }
      }
    }
  }


//Testa se tem um NT na esquerda, se não tiver, não é uma
 produção valida
  for (const iterator of esquerda) {
    if (!/[A-Z]/g.test(iterator)) {
      gi = false;
      gsc = false;
      gr = false;
      glc = false;
    }
  }
```

Abaixo se encontra o código para escrever as sentenças, para fazer tal, se utilizou uma função recursiva.

```
//Inicio da criação da sentença
//Pega o valor inicial
let inicio = $('#si').val();
let sentenca = inicio;
let sentencas = '';
for (const key in linhas) {
  if (esquerda[key] == inicio) {
    //Repete o processo três vezes
    for (let a = 0; a < 3; a++) {
      //Lança a função para criar a sentença com
      //os dados (inicio, opções da direita do inicio, sentença anterior)
      criaSentenca(esquerda[key], direita[key], esquerda[key]);
      sentenca.replace(/&/g, ''); //Retira o vazio
      sentencas = sentencas + sentenca + '<br>'; //Concatena as sentenças geradas
      sentenca = inicio; //re-inicia o loop
    }
    //escreve na tela a resposta
    sentencas = `Senteças Geradas = { <br />
        ${sentencas}
        }`;
    $('.resultGramaticas').html(sentencas);
    break;
  }
}

function criaSentenca(nt, t, anterior) {
  let aux = t.split('|');
  let limit = aux.length;
  //Pega uma opção da direita randomicamente e substitui
  let randT = aux[Math.floor(Math.random() * limit)];
  let nova = anterior.replace(nt, randT);
  sentenca = sentenca + ' → ' + nova;
  try {
    //testa se ainda existem NT na sentença, caso existirem,
    //escolha uma randomicamente e repete o processo
    if (/[A-Z]/g.test(nova)) {
      let NT = [];
      for (const key in esquerda) {
        if (nova.includes(esquerda[key])) {
          NT.push(key);
        }
      }
      if (NT.length != 0) {
        let rand = Math.floor(Math.random() * NT.length);
        criaSentenca(esquerda[NT[rand]], direita[NT[rand]], nova);
      } else {
        alert('Erro na produção: Não possui fim');
      }
    }
  } catch (e) {
    alert('Erro na produção: Loop infinito');
  }
}
```

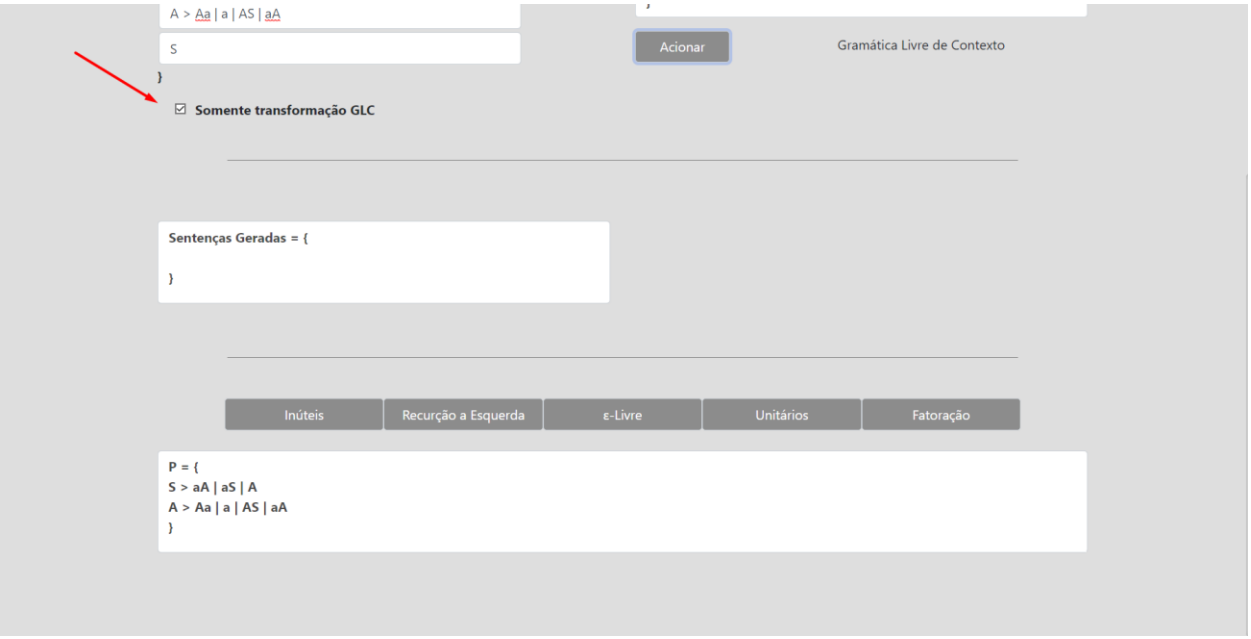
E por fim, está o código para gerar a tabela de autômato finito.

```
// Automato Finito
if (gr) {
  //cria a tabela
  $('#tabela').show();
  let tableHead = `<tr><th scope="col">#</th>`;
  let tableBody = `<tr>`;
  //pega o alfabeto
  let alfabeto = new Set(dirSimbolo);
  for (const i of alfabeto) {
    tableHead += `<th scope="col">${i}</th>`;
  }
  for (const i of linhas) {
    tableBody += `<th class="tableRow" scope="row">${i.split('>')[0]}</th>`;
    for (const a of alfabeto) {
      //pega os conjuntos de estados e testa eles contra o alfabeto
      let regex = new RegExp(`\\b${a}\\b`, 'g');
      if (regex.test(i)) {
        if (/[A-Z]/g.test(a)) {
          let aux = a.replace(/[a-z]/g, '').replace(/(?!^)(?!$)/g, '/');
          tableBody += `<td>${aux}</td>`;
        } else {
          tableBody += `<td>ε</td>`;
        }
      } else {
        tableBody += `<td>-</td>`;
      }
    }
    tableBody += `</tr>`;
  }
  tableHead += `</tr>`;
  //escreve na tela
  $('#tableHead').html(tableHead);
  $('#tableBody').html(tableBody);
} else {
  $('#tabela').hide();
}
```

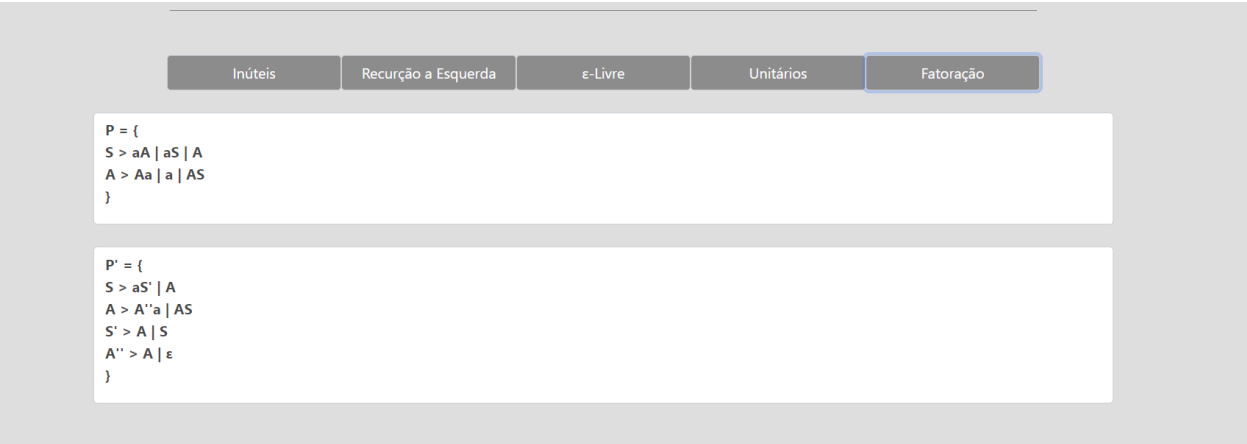
Parte 2:

Interface:

Foi adicionado um marcador para poder usar somente as funcionalidades de transformações.



Quando clicar em acionar, se tiver sido marcado ou ter sido reconhecido como GLC, abre uma nova área para trabalhar, nessa área existem 5 botões, cada um irá realizar a transformação indicada e irá desenhar abaixo o resultado.

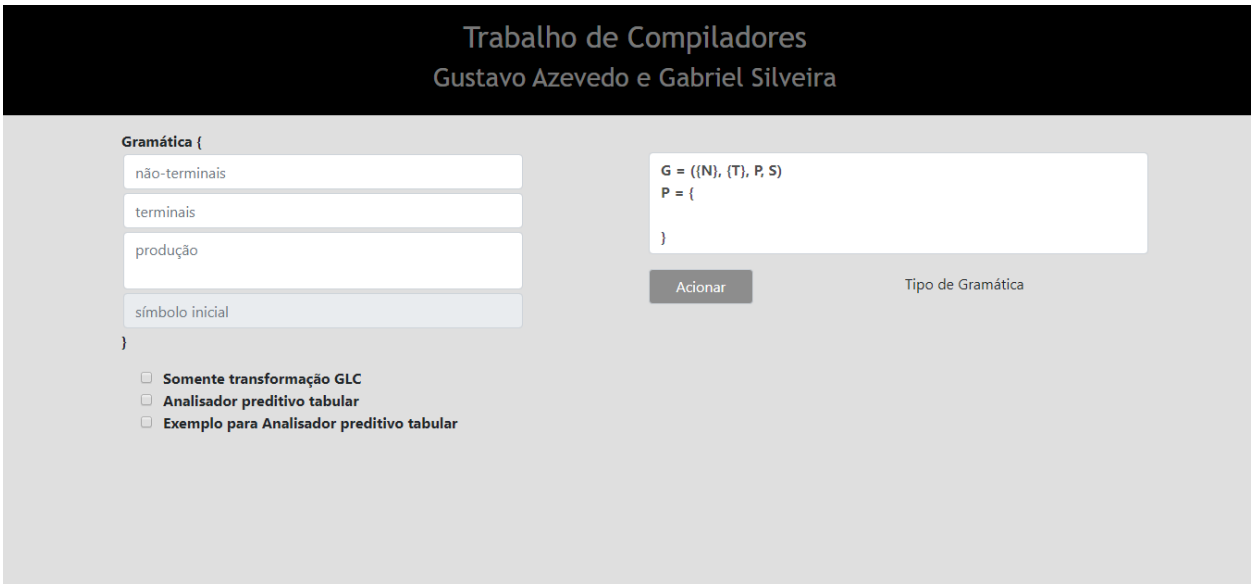


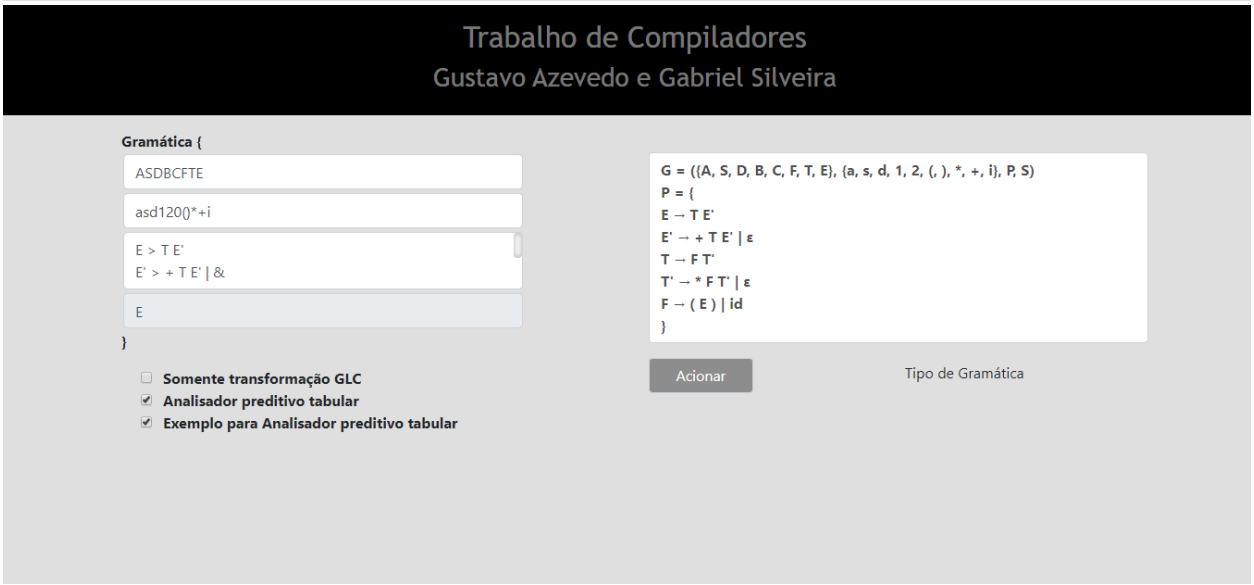
Versão 2.0:

A segunda versão do código aplica a novas funcionalidades requeridas do trabalho de compiladores à ferramenta criada anteriormente para a cadeira de formais.

1. **Interface**

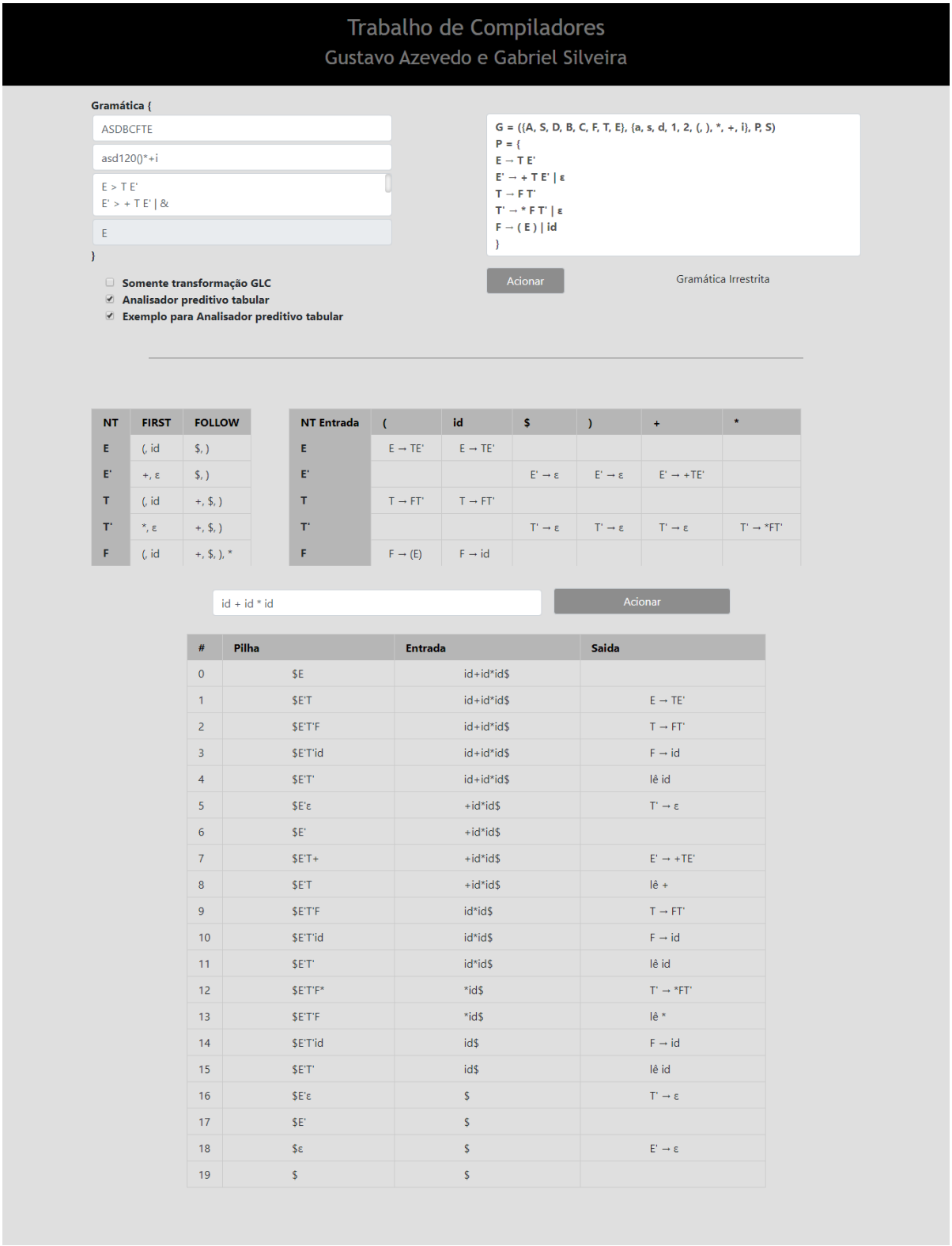
   Foram adicionados checkboxes para a ferramenta, para utilizar as funções novas, é necessário marcar o checkbox “Analisador preditivo tabular”, também pode marcado o checkbox “Exemplo”, que irá auto completar os campos com uma gramática válida.





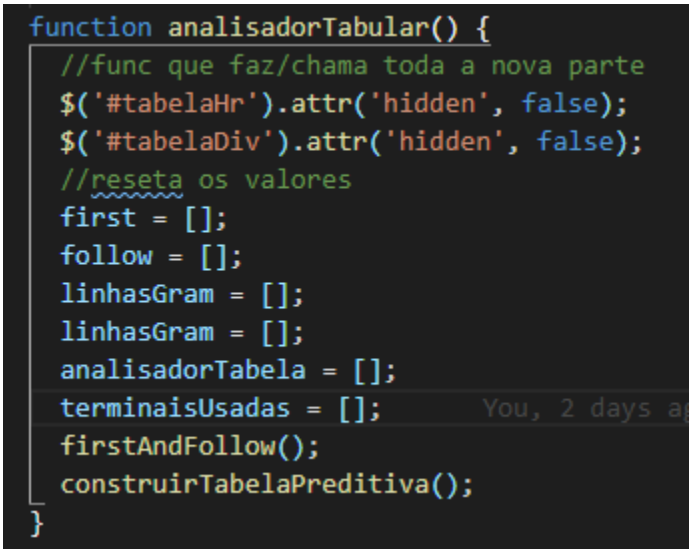
Uma vez que se tenha a gramática inserida (os diferentes itens da produção devem ser separados por espaço, ex.: S > + A id é uma produção com a direita com três itens, sendo +, A, id, pode ser também inserida uma segunda parte, separando as duas por “|”, e o vazio é o

símbolo “&”) se pode acionar o algoritmo a partir do botão “Acionar”, ele irá gerar então uma tabela First Follow, uma tabela de análises e uma área para inserir uma entrada, uma vez que se passe essa entrada, o algoritmo ira usar a tabela de análises para reconhecer dada entrada.



2. **Funcionalidades**

O algoritmo funciona de modo que, se e apenas se, o checkbox “Analisador preditivo tabular” estiver marcado, a função principal irá chamar a função “analisadorTabular()”.



```
function analisadorTabular() {
  //func que faz/chama toda a nova parte
  $('#tabelaHr').attr('hidden', false);
  $('#tabelaDiv').attr('hidden', false);
  //reseta os valores
  first = [];
  follow = [];
  linhasGram = [];
  linhasGram = [];
  analisadorTabela = [];
  terminaisUsadas = [];       You, 2 days a
  firstAndFollow();
  construirTabelaPreditiva();
}
```

Essa função, por sua vez chama as funções para construir o first follow e a tabela de análise.

A função first follow funciona de modo que se, primariamente, separe as linhas da gramática em esquerda e direita, com a direita sendo uma matrix com os itens inseridos para cada produção.

```
function firstAndFollow() {
  //separa as linhas para poder trabalhar nelas
  for (const i of linhas) {
    let aux = i.split('>');
    let dir = [];
    for (const a of aux[1].split('|')) {
      let aws = [];
      for (const s of a.split(' ')) {
        if (s != '') {
          aws.push(s);
        }
      }
      dir.push(aws);
    }
    linhasGram.push({
      esquerda: aux[0].replace(/ /g, ''),
      direita: dir
    });
  }
  /*
  -- linhasGram --
  G = {
    S > A D
    A > 1 S | &
    D > 2 A | 2
  }
  Array(3)
    0: {esquerda: "S", direita: Array(1)}
    1: {esquerda: "A", direita: Array(2)}
    2:
      direita: Array(2)
        0: Array(2)
          0: "2"
          1: "A"
        1: Array(1)
          0: "2"
      esquerda: "D"
 */
```

Uma vez feito isso, ele começa a passar as regras do first e cria um array que irá salvar todos os itens de first para cada produção da gramática.

```
//encontra o first
  for (const lin of linhasGram) {
    if (!/[A-Z]/g.test(lin.direita[0][0])) {
      first.push({
        esquerda: lin.esquerda,
        direita: lin.direita,
        first: []
      });
      //regra 1, procura por todos os que começam com terminais
      for (const i of lin.direita) {
        if (
          !/[A-Z]/g.test(i[0]) &&
          !first[first.length - 1].first.includes(i[0])
        ) {
          first[first.length - 1].first.push(i[0]);
        }
      }
    } else {
      first.push({
        esquerda: lin.esquerda,
        direita: lin.direita,
        first: []
      });
    }
  }

// regra 2, decrescente, substitui as terminais pelo first nece
ssários
  for (let i = first.length - 1; i >= 0; i--) {
    if (first[i].first.length == 0) {
      for (const a of first) {
        if (a.esquerda == first[i].direita[0][0]) {
          first[i].first = a.first;
        }
      }
    }
  }
```

Depois disso é feito o mesmo para o follow.

```
// follow
  //regra 1, add $ para o inicial e cria os outros follows
  for (const lin of linhasGram) {
    if (lin.esquerda == inicio) {
      follow.push({
        esquerda: lin.esquerda,
        direita: lin.direita,
        follow: ['$']
      });
    } else {
      follow.push({
        esquerda: lin.esquerda,
        direita: lin.direita,
        follow: []
      });
    }
  }
  //regra 2 e 3
  for (const lin in follow) {
    for (const aws of follow) {
      for (const a in aws.direita) {
        if (aws.direita[a][1] == follow[lin].esquerda) {

//regra 2, se tiver 3 char e não for NT, add para o follow, se não add o firs
t do NT para o follow menos o vazio
          if (aws.direita[a].length == 3) {
            if (!/[A-Z]/g.test(aws.direita[a][2])) {
              follow[lin].follow.push(aws.direita[a][2]);
            } else {
              for (const b of first) {
                if (b.esquerda == aws.direita[a][2]) {
                  follow[lin].follow.push(...b.first);
                  follow[lin].follow = follow[lin].follow.filter(
                    e => e !== '&'
                  );
                }

//regra 3 se length == 3, se o first da NT possuir vazio, add o follow do NT pa
ra o follow
                if (b.first.includes('&')) {
                  if (aws.follow.length > 0) {
                    follow[lin].follow.push(...aws.follow);
                  } else {
                    follow[lin].follow.push(`FOLLOW(${aws.esquerda})`);
                  }
                }
              }
            }
          }
          // regra 3, add o follow do NT para o follow
          if (aws.direita[a].length == 2) {
            if (aws.follow.length > 0) {
              follow[lin].follow.push(...aws.follow);
            } else {
              follow[lin].follow.push(`FOLLOW(${aws.esquerda})`);
            }
          }
        }
      }
    }
  }
```

Após fazer isso o resultado um filtro é passado pra retirar duplicatas e o resultado é escrito na tela.

```
//elimina possíveis duplicatas
  for (const aws of follow) {
    aws.follow = [...new Set(aws.follow)];
  }
  for (const aws of first) {
    aws.first = [...new Set(aws.first)];
  }

  // desenha ma tela
  let tableBody = `<tr>`;
  for (const i in first) {
    tableBody += `<th class="tableRow" scope="row">${
first[i].esquerda}</th>`;
    let a = 0;
    if (a == 0) {
      tableBody += `<td>${first[i].first.join(', ')}`;
      a = first[i].first.length - 1;
    } else {
      tableBody += `${first[i].first[a]}</td>`;
    }

    a = 0;
    if (a == 0) {
      tableBody += `<td>${follow[i].follow.join(', ')}`;
      a = first[i].first.length - 1;
    } else {
      tableBody += `${follow[i].follow[a]}</td>`;
    }
    tableBody += `</tr><tr>`;
  }
  //escreve na tela o First Follow
  tableBody = tableBody.replace(/&/g, 'ε');
  $('#tableBodyFF').html(tableBody);
```

Após fazer isso, é chamada a função “construirTabelaPreditiva()”, a primeira coisa que ele faz é pegar todos os terminais usados (menos o vazio) e salvá-los em um array que será utilizado para testar cada produção para ver se existem traduções.

```
function construirTabelaPreditiva() {
  //pega todos os terminais usados menos o vazio
  analisadorTabela = [];
  for (const i in linhasGram) {
    for (const f of first[i].first) {
      terminaisUsadas.push(f);
    }
    for (const f of follow[i].follow) {
      terminaisUsadas.push(f);
    }
  }

  //tira as duplicatas e o vazio
  terminaisUsadas = [... new Set(terminaisUsadas)];
  terminaisUsadas = terminaisUsadas.filter(e => e !== '&');
```

O resultado final é uma array com extrutura ilustrada abaixo (ex.: no terminal “+”, essa produção traduz recebe “B -> +TE’”), que será então escrito na tela como uma tabela.

```
/*
 analisadorTabela =
 0: {esquerda: "S", tabela: Array(2)}
 1:
   esquerda: "B"
     tabela: Array(0)
       $: ["&"]
       (: []
       ): ["&"]
       *: []
       +: (3) ["+", "T", "E'"]
       id: []
 2: {esquerda: "A", tabela: Array(2)}
 3: {esquerda: "C", tabela: Array(4)}
 4: {esquerda: "F", tabela: Array(2)}
 */
```

Depois de isso ser feito, o usuário pode inserir então uma sentença para testar se ela irá ser reconhecida ou não por aquela gramática. Essa funcionalidade é acionada quando o usuário digita na nova entrada abaixo das tabelas de análise e first follow e acionar o botão, que irá rodar a função “runEntrada()”, essa função irá criar uma fila com a sentença inserida, uma pilha com o final e o inicio e uma array para salvar cada iteração do processo. Uma vez feito isso, o algoritmo irá se repetir até a pilha chegar no final (símbolo “$”) e então escreve na tela na tela.

```
function runEntrada() {
  let entrada = $('#entrada').val();
  entrada = entrada.split(' ');
  entrada.push('$');
  let pilha = ['$', inicio];
  let tabelaEntrada = [];
  tabelaEntrada.push({
    pilha: pilha.join(''),
    entrada: entrada.join(''),
    saida: ''
  });

/* enquanto a pilha não chegar no final, faz um pop da pilha e pega o primeiro da fila, se o primeiro da fila for o mesmo
  que o da pilha, le aquele item, e retira ele da fila */
  while (pilha[pilha.length - 1] != '$') {
    let popPile = pilha.pop(),
      shiftQueue = entrada[0],
      saida = '';
    if (popPile == shiftQueue) {
      tabelaEntrada.push({
        pilha: pilha.join(''),
        entrada: entrada.join(''),
        saida: `lê ${shiftQueue}`
      });
      entrada.shift();
      popPile = pilha.pop();
      shiftQueue = entrada[0];
    }
    // testa por todos os itens da tabela procurando pelo NT correspondente, se achar, add para a pilha e add a saida
    for (const i of analisadorTabela) {
      if (i.esquerda == popPile && i.tabela[shiftQueue].length > 0) {
        saida = `${i.esquerda} > ${i.tabela[shiftQueue].join('')}`;
        for (let index = i.tabela[shiftQueue].length - 1; index >= 0; index--) {
          pilha.push(i.tabela[shiftQueue][index]);
        }
      }
    }
    tabelaEntrada.push({
      pilha: pilha.join(''),
      entrada: entrada.join(''),
      saida: saida
    });
  }
```