Grupo Magneto

Conversor de Autômatos

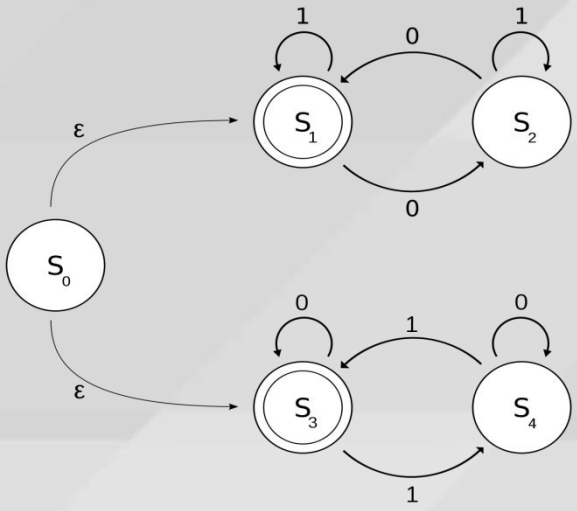
(AFND, AFD e ER)

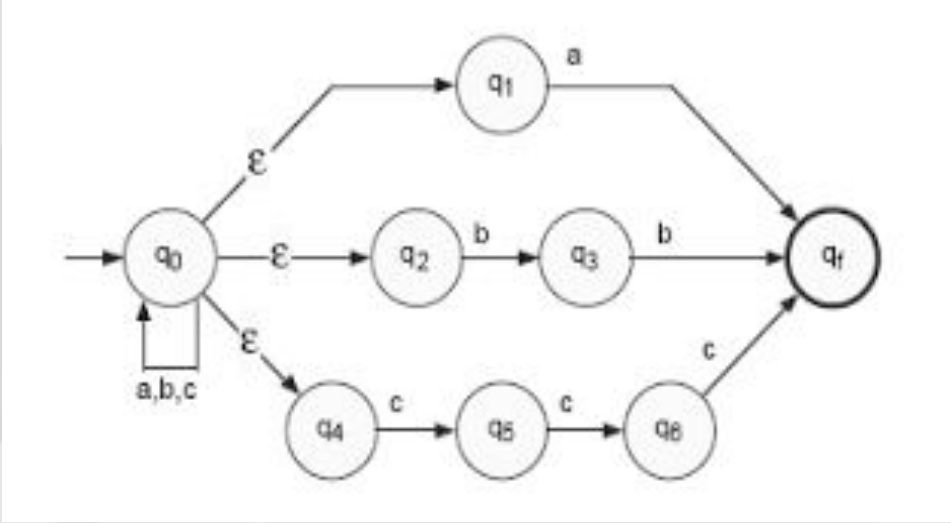
Integrantes:
Bruno Schwamborn, Christopher Coutinho, Maria Cecília Galvão, Davi Austregesilo, Eduardo Lupa, Rodolfo Prazim, Lara Nogueira

POLI ESCOLA POLITÉCNICA DE PERNAMBUCO
UPE UNIVERSIDADE DE PERNAMBUCO

Conversão AFND para AFD

- AFND (autômato finito não determinístico) é um autômato que detém de diversos estados e possíveis estados próximos que o levam a um estado final.
- O AFD é possui estados possíveis de aceitação ou não de cadeia simbólicas, podendo descobrir seu estado final.





Função principal (main(void))

Loop principal para verificação de *flags*

```
int convert = 0; /* controle do tipo de conversao */
int entrada = 1; /* variavel de controle de tipo de entrada 0 - para arquivo e 1 - para entrada manual */
```

```
  while((opt = getopt(argc, argv, "hcnvf:")) != EOF)
      switch(opt)
      {
          case 'h':
              help();
              break;
          case 'c':
              copyr();
              break;
          case 'n':
              convert = 1; /* conversao vai ser do tipo AFND -> AFD */
              break;
          case 'v':
              verb++;
              break;
          case 'f':
              strncpy(name_file, optarg, SBUFF); /* recebe o nome de um arquivo com maximo de tamanho de SBUFF (256) */
              entrada = 0; /* entrada 0 representa entrada por arquivo */
              break;
      case '?':
          help();
      default:
          help();
          printf("Type\n\t$man %s\nor\n\t$%s -h\nfor help.\n\n", argv[0], argv[0]);

          return EXIT_FAILURE;
  }
```

Organização da conversão AFND->AFD

```
    /* ------------------------------------------- */
    /* ------------------Conversao----------------- */
    switch(convert)
    {
        case 1: /* conversao vai ser do tipo AFND - AFD */
            if(entrada == 0) /* entrada por file */
                recebe_afnd(file, &afnd);   /* funcao recebe afnd do arquivo */
            afd = calcular_afd(afnd);       /* funcao retorna a traducao */
            print_quintupla(afd);           /* print afd na tela */
            break;
        default: /* caso nao tenha conversao imprime a help */
            help();
            return EXIT_FAILURE;
    }

    /* ------------------------------------------- */
    /* ------------------------------------------- */

    return EXIT_SUCCESS;
}
```

Função recebe_afnd()

Entrada sendo a quíntupla de um arquivo exemplo dado

```
while(fgets(c_file, 256, file) != NULL)         /* le todas as linhas do arquivo */
{
    if(opt == 1 && strstr(c_file,"#") == NULL)
        copia->K = atoi(c_file);
    if(opt == 2 && strstr(c_file,"#") == NULL)
        copia->A = c_file[0];
    if(opt == 3 && strstr(c_file,"#") == NULL)
        copia->S = atoi(c_file);
    if(opt == 4 && strstr(c_file,"#") == NULL)
    {
        copiatoken = strdup(c_file);
        while((token = strsep(&copiatoken," ")) != NULL)
            inserir_final_st(token,&copia);
    }
    if(opt == 5 && strstr(c_file,"#") == NULL)
    {
        inserir_delta_st(c_file,&copia);
    }
    if(strstr(c_file,"#K")) /* ao encontrar #K proxima linha sera lida como numero de estados */
        opt = 1;
    if(strstr(c_file,"#A")) /* ao encontrar #A proxima linha sera lida como ultima letra do alfabeto */
        opt = 2;
    if(strstr(c_file,"#S")) /* ao encontrar #S proxima linha sera lida como estado inicial */
        opt = 3;
    if(strstr(c_file,"#F")) /* ao encontrar #F proxima linha sera lida como lista de estados finais */
        opt = 4;
    if(strstr(c_file,"#D")) /* ao encontrar #D proxima linha sera lida como lista da funcao de transicao d(ei, le, ef) */
        opt = 5;
}

*cabeca = copia; /* passa a nova quintupla copia para a variavel cabeca */
```

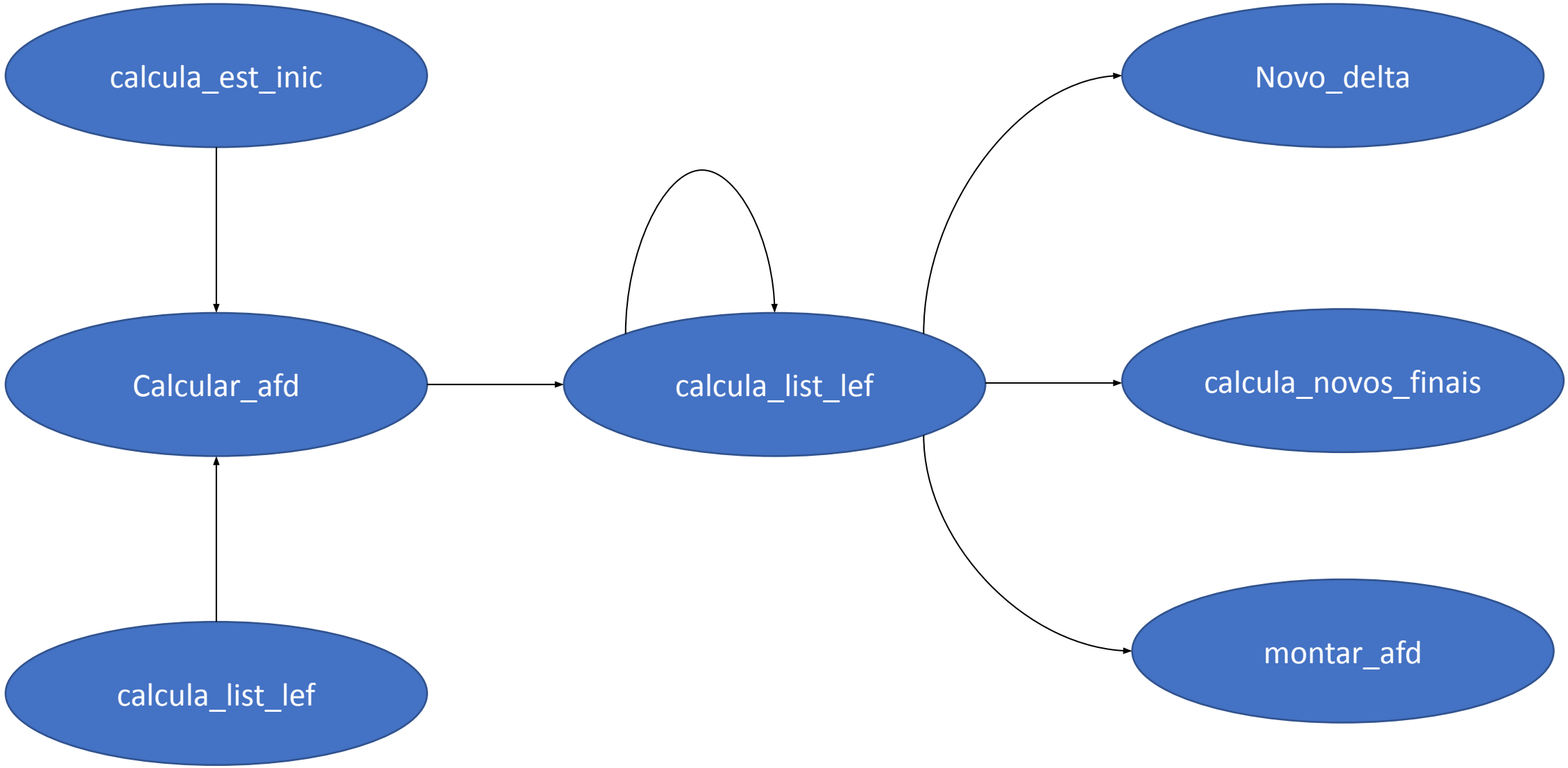
Função calcular_afnd()

- Corpo da conversão

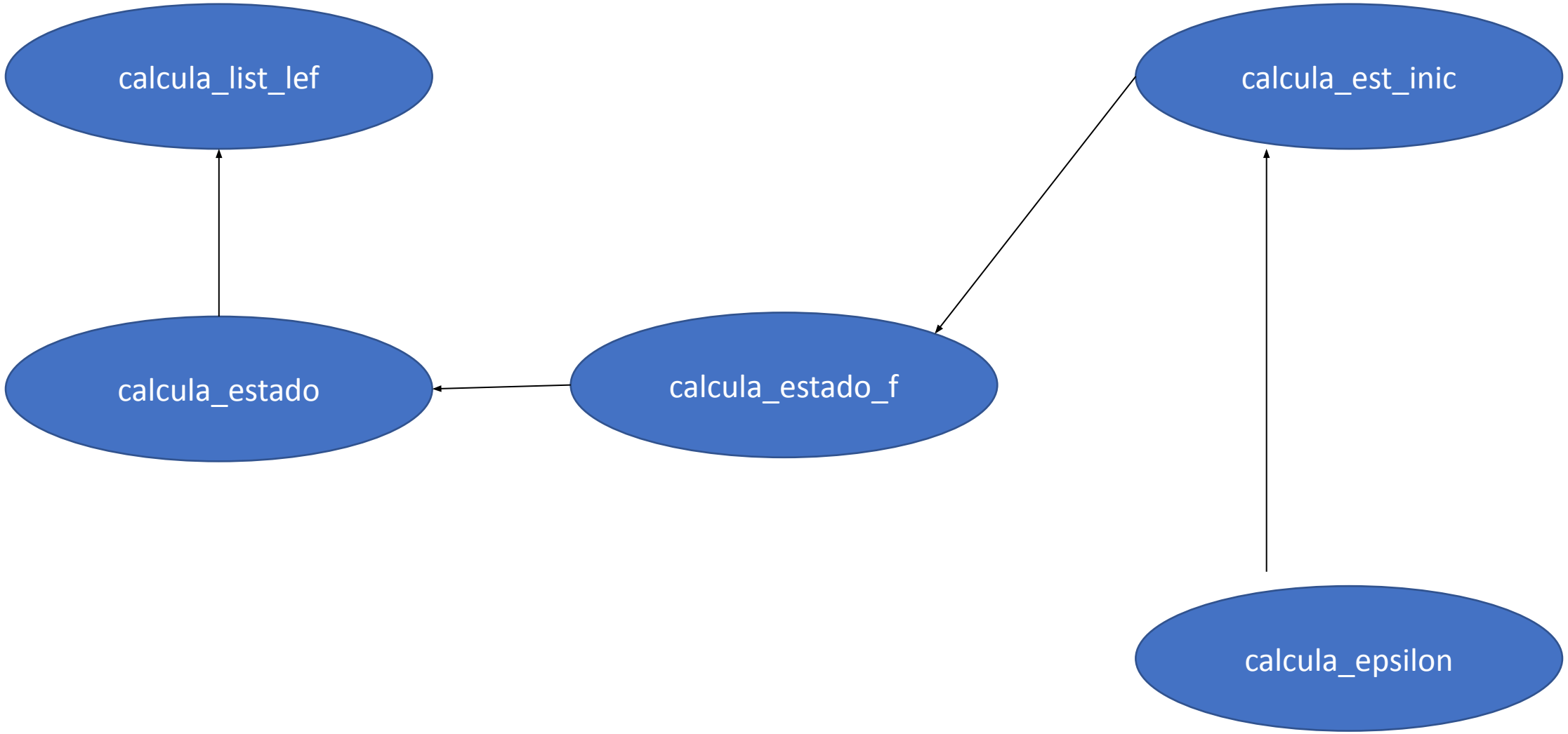
```
t_quintupla *afd = NULL;
t_delta *n_delta = NULL;
t_list_lef *base = NULL;
t_list_lef *ll = NULL;
t_list_lef *calculo = NULL;
t_lef *inic = NULL;
t_lef *n_lef = NULL;

inserir_final(q->S,&inic);          /* adiciona o estado inicial a lista a ser calculada */
inserir_list_final(inic, &base);    /* adiciona a base dos novos estados */
ll = calcular_list_lef(q, inic);    /* calcula os destinos do estados inicial */
while(ll != NULL)   /* enquanto a lista nao for nula */
{
    calculo = calcular_list_lef(q,ll->F);   /* calcular os estados finais de cada componente da lista */
    while(calculo != NULL)  /* enquanto a lista nao for nula */
    {
        if(buscar_list(calculo->F,base) == 0)   /* caso o estado final nao exista na lista base */
            inserir_list_final(calculo->F,&ll); /* adiciona a lista "ll" para calcular seus estados finais */
        calculo = calculo->prox;                /* proximo da lista de calculo */
    }
    inserir_list_final(ll->F,&base);    /* adiciona o valor de estado inicial/destino a lista base */
    ll = ll->prox; /* proximo da lista de "ll" */
}
n_delta = novo_delta(q, base);          /* calcula o novo delta com a nova lista de estados "base" */
n_lef = calcular_novos_finais(q,base);  /* calcula a nova lista de estados finais aceitos */
afd = montar_afd(q,n_delta,base,n_lef); /* monta a quintupla da AFD com os valores calculados anteriormente */
```

Calculo da AFD



Calculo_list_lef



Função recebe_afd()

- adiciona todos estados e parametros(K, S, F e D) para a struct da quintupla

```
while(fgets(c_file, 256, file) != NULL)         /* le todas as linhas do arquivo */
{
    if(opt == 1 && strstr(c_file,"#") == NULL)
        copia->K = atoi(c_file);
    if(opt == 2 && strstr(c_file,"#") == NULL)
        copia->A = c_file[0];
    if(opt == 3 && strstr(c_file,"#") == NULL)
        copia->S = atoi(c_file);
    if(opt == 4 && strstr(c_file,"#") == NULL)
    {
        copiatoken = strdup(c_file);
        while((token = strsep(&copiatoken," ")) != NULL)
            inserir_final_st(token,&copia);
    }
    if(opt == 5 && strstr(c_file,"#") == NULL)
    {
        inserir_delta_st(c_file,&copia);
    }
    if(strstr(c_file,"#K")) /* ao encontrar #K proxima linha sera lida como numero de estados */
        opt = 1;
    if(strstr(c_file,"#A")) /* ao encontrar #A proxima linha sera lida como ultima letra do alfabeto */
        opt = 2;
    if(strstr(c_file,"#S")) /* ao encontrar #S proxima linha sera lida como estado inicial */
        opt = 3;
    if(strstr(c_file,"#F")) /* ao encontrar #F proxima linha sera lida como lista de estados finais */
        opt = 4;
    if(strstr(c_file,"#D")) /* ao encontrar #D proxima linha sera lida como lista da funcao de transicao d(ei, le, ef) */
        opt = 5;
}

*cabeca = copia; /* passa a nova quintupla copia para a variavel cabeca */
```

Função afd2er()

- adiciona novos estados ei e ef e atualiza os parametros(K, S, F e D)

```
//define estado final e insere na lista d
while(lef != NULL)
{

    //define e insere ao final de ef
    inserir_delta(lef->f, " ", cabeca->K, &d);
    //porem como os estados sao enumerados a partir de 0, entao apenas K.

    //vai para o prox elemento F
    if(lef->prox == NULL) break;
    else lef = lef->prox;
}

//insere novo ei e novo ef a t_quintupla
q = cabeca;//vai p/ inicio da lista
q->K = q->K + 2; //add 2 estados novosS
i = 0;
lef = cabeca->F;//vai p/ inicio da lista
while(lef != NULL)//ve quantos elementos tem na lista lef
{
    ++i;//conta elementos
    if(lef->prox == NULL) break;//evita segmentation fault
    else lef = lef->prox;//chama prox da lista lef
}
if(i == 1)//lista com apenas 1 elemento
{
    q->F->f = q->K - 1; //define fase final como a quantidade total de fases(K) -1,
    //pois as fases iniciam contagem a partir de 0
}
else//lista com mais de um elemento
{
    int j = 0;
    do //elimina elementos da lista enquanto quantidade de elementos for maior que 1(j>1)
    {
        lef = cabeca->F;
        while(lef != NULL)//remove elementos
        {
            remover_lef(1, &lef);//chama funcao que remove
            lef = lef->prox;//aponta p/ prox elemento de lef
        }

        lef = cabeca->F;//aponta p/ inicio de lef
        j = 0; //reinicia contador
```

```
0   0
0 a 0
0 b 0
1 a 0
1 b 1
1   2
```

```
#S
0
#F
1
```

Função afd2er()

- adiciona novos estados ei e ef e atualiza os parametros(K, S, F e D)

```
        lef = cabeca->F;//aponta p/ inicio de lef
        j = 0; //reinicia contador
        while(lef != NULL)//conta quantos elementos tem na lista
        {

            lef = lef->prox;//chama proximo elemento de lef
        }
    }
    while(j > 1);
    lef = cabeca->F;//aponta p/ inicio da lista
    lef->f = q->K - 1; //define fase final como a quantidade total de fases(K) -1,
                       //pois as fases iniciam contagem a partir de 0
}

//add 1 a todos, pois um novo estado inicial foi adicionar
while(d != NULL)
{
    if(d->ei == 1000)//novo estado inicial
    {
        //se tiver mais de um estado inicial isso deve ser alterado
        d->ei = 0;
        d->ef = 1;
    }
    else
    {
        //if removivel, corrigindo erro que nao entendo
        if(d->ei == 2 && d->ef == 1)
            strcpy(d->le, "a");
        d->ei = d->ei + 1;
        d->ef = d->ef + 1;
    }
    d = d->prox;
}

//concatena ciclos infinitos com *   (ex.:a*)
d = cabeca->D; //aponta p/ o inicio da lista
while(d != NULL)
{
    if(d->ei == d->ef)
    {
        sprintf(bufferLE, "%s*", d->le);
        d->le = strdup(bufferLE);
    }
    d = d->prox;
}
```

```
0    1
1 a* 1
1 b* 1
2 a  1
2 b* 2
2    3
```

Função afd2er()

- adiciona novos estados ei e ef e atualiza os parametros(K, S, F e D)

```
    //concatena
    d = cabeca->D;
    i = 1; //i=1 indica que concatenara a partir do estado 1
    concatena(d, i, cabeca, 0); //0 eh mandado como parametro, pois sera uma funcao recursiva chamando a si mesma
                                //a funcao eliminaEstados, tambem chama a concatena, mas nesse caso eh enviado 1 como parametro

    //elimina estados
    i = 1; //i=1 indica que eliminara a partir do estado 1
    eliminaEstados(d, i, cabeca);

    //elimina elementos que tem 1000 como parametro
    d = cabeca->D;
    int k = 1;
    int flag = 0;
    t_delta *dSalve = NULL;//salva d anterior
    while(d != NULL)
    {
        flag = 0;
        if(k == 1 && d->ei == 1000)
        {
            cabeca->D = cabeca->D->prox;
            d = cabeca->D;
            --k;
            flag = 1;
        }
        else
            if(d->ei == 1000)
                dSalve->prox = d->prox;
        ++k;
        dSalve = d;
        if(flag == 0)
            d = d->prox;
    }

    //transforma em er
    d = cabeca->D;
    transER(cabeca->D, cabeca);
    free(d);
}
```

Função concatena()

- concatena arcos de ciclos com o arco pro estado final

```
//percorre d e pesquisa ei =i
while(d != NULL)
{
    if(d->ei == i)//encontrou ei =i
    {
        if(d->ei == d->ef)//aponta p/ ele mesmo
            flagApontaEleMesmo = 1;
        if(d->ef == principal->F->f)//aponta p/ FinalGeral
            flagFinal = 1;
    }
    d = d->prox;
}
//encontrou ei = i
if(flagFinal == 1 && flagApontaEleMesmo == 1)
{
    d = principal->D;//volta p/ inicio

    //pesquiso novamente p/ encontrar o mesmo ei=1
    while(d != NULL)
    {

        if(d->ei == i && d->ef == i) //encontrou d->ei=d->ef=i, (ex1.: 2 (a)* 2) ou (ex2.: 3 (b)*3)
        {
            dAux = principal->D; //volta p/ inicio do aux

            //procuro se existe um dAux->ei == i && dAux->ef == estado final geral
            while(dAux != NULL)
            {
                //d deve possuir ei=ef, (ex1.: 2 (a)* 2) ou (ex2.: 3 (b)* 3)
                //dAux deve ter dAux->ei == d->ei && dAux->ef== principal->F->f, (ex1.: 2 " " 4) ou (ex2.: 3 a(a)* 4)
                if(d->ei == d->ef && dAux->ei == d->ef && dAux->ef == principal->F->f && flagFinal == 1)
                {
                    //salva p/ inserir novo caminho
                    inserirEI[contInserir] = d->ei;
                    strcpy(inserirLE[contInserir], d->le);
                    strcat(inserirLE[contInserir], dAux->le);
                    inserirEF[contInserir] = principal->F->f;
                    ++contInserir;

                    //nao se pode apagar d ou dAux agr, pois eles ainda serao utilizados
                    //serao apagados em seguida, depois de inserir todos

                    //ativa flag p/indicar que removeu
                    flagRemoveu = 1;
                    salveFlagRemoveu += flagRemoveu;
                }
```

0		1
1	a*	1
1	b*	1
2	a	1
2	b*	3

Função concatena()

- concatena arcos de ciclos com o arco pro estado final

```
                //chama prox
                dAuxSalve = dAux; //salve dAux anterior
                dAux = dAux ->prox;
            }//FIM while dAux != NULL

            //confere se algo foi removido p/ ver se alteracoes sao necessarias
            if(flagRemoveu == 1)
            {
                dSalve->prox = d->prox;

                //volta posicao de d
                d = dSalve;


                //reinicia flag q indica se houve remocao
                flagRemoveu = 0;
            }
        }
        dSalve = d;//salva posicao anterior /*e se isso nao acontecesse pq eh a primeira
        d = d->prox;//chama prox
    }

    if(salveFlagRemoveu > 0)
    {
        //remove dAux
        dAux = cabeca; //mando p/ inicio
        dAuxSalve = dAux; //salva dAux anterior
        while(dAux != NULL)
        {
            d = cabeca; //precisa ser sempre o mesmo d, pois ele eh a referencia
            if(dAux->ei == i && dAux->ef == principal->F->f && flagFinal == 1)
            {
                dAuxSalve->prox = dAux->prox;
                dAux = dAuxSalve;
            }
            dAuxSalve = dAux;
            dAux = dAux->prox;
        }

        //insiro novos elementos
        for(int j = 0; j < contInserir; j++)
        {
            d = cabeca; //volta p/ inicio
            inserirLEtoken = strdup(inserirLE[j]);
            inserir_delta(inserirEI[j], inserirLEtoken, inserirEF[j], &d);
        }
    }
```

```
                if(dAux->ei == i && dAux->ef == principal->F->f && flagFinal == 1)
                {
                    dAuxSalve->prox = dAux->prox;
                    dAux = dAuxSalve;
                }
                dAuxSalve = dAux;
                dAux = dAux->prox;
            }

            //insiro novos elementos
            for(int j = 0; j < contInserir; j++)
            {
                d = cabeca; //volta p/ inicio
                inserirLEtoken = strdup(inserirLE[j]);
                inserir_delta(inserirEI[j], inserirLEtoken, inserirEF[j], &d);
            }
        }
    }

    //chama eliminaEstados
    if(i <= principal->F->f - 1 && elimina == 1)
        eliminaEstados(principal->D, i, principal);

    //chama recursividade
    if(i < principal->F->f - 1)
        concatena(principal->D, ++i, principal, 0);
}
```

Função eliminaEstados()

- Elimina o estado final enviado

```
if(flagAconteceu == 1)
{
    //roda d
    while(d != NULL)
    {
        //reinicia flag
        flagRemove = 0;

        //encontrou ef==i
        if(d->ef == i)
        {
            //madnda da dAux p inicio
            dAux = principal->D;

            //reinicia contagem p/ inserir
            contInserir = 0;

            //roda dAux
            while(dAux != NULL)
            {
                //confere se existe algum dAux->ei=d->ef
                if(dAux->ei == i)
                {
                    //salvar dAux p/ apagar posteriormente
                    if(flagApagar == 0) //flag evita tentativa de apagar o mesmo mais de uma vez
                    {
                        apagarDAuxei[contApagarDaux] = dAux->ei;
                        strcpy(apagarDAuxle[contApagarDaux], dAux->le);
                        strcpy(apagarDAuxle[contApagarDaux], dAux->le);
                        apagarDAuxef[contApagarDaux] = dAux->ef;
                        ++contApagarDaux;
                    }
                    //copia caminhos p/ inserir nova linha na lista
                    strcpy(bufferLE, d->le); //copia caminho de d->ei ate i
                    strcat(bufferLE, dAux->le); //concatena caminho de d->ei ate dAux->ef
                    //insere na lista
                    inserirEI[contInserir] = d->ei;
                    strcpy(inserirLE[contInserir], bufferLE);
                    inserirEF[contInserir] = dAux->ef;
                    ++contInserir;
                }
                dAux = dAux->prox;
            }
```

0 a* 1
0 b* 1
2 a 1
2 b* 3

Função eliminaEstados()

- Elimina o estado final enviado

```
        //apagar d
        if(d->ei == 0)//primeiro membro da lista
        {
            d->ei = 1000;
            d->ef = 1000;
        }
        else
        {
            dSalve->prox = d->prox;
            dSalve = dSalve->prox;
            d = dSalve;
            flagRemove = 1;
        }

        //inserir novas coisas
        for(int j = 0; j < contInserir; j++)
        {
            if(inserirEI[j] == 2000)break;
            inserirLEtoken = strdup(inserirLE[j]);
            inserir_delta(inserirEI[j], inserirLEtoken, inserirEF[j], &d);
            inserirEI[j] = 2000;
        }

        //nao precisa add mais a lista de apagar do dAux, pois sempre serao os mesmo
        flagApagar = 1;
    }

    //incia proximo ciclo
    if(flagRemove == 0) //n ha alteracao a ser feita
    {
        dSalve = d;
        d = d->prox;
    }
    else //nosso dSalve  ja esta apontando p/ o proximo
        d = dSalve;
}

//apagar dAux
for(int j = 0; j < contApagarDaux; j++)
{
    //reinicia flag
    flagRemove = 0;

    //manda p/ inicio da lista
    d = cabeca;
```

```
    //apagar dAux
    for(int j = 0; j < contApagarDaux; j++)
    {
        //reinicia flag
        flagRemove = 0;

        //manda p/ inicio da lista
        d = cabeca;

        //procura na lista p/ remover

        while(d != NULL)
        {
            if(d->ei == apagarDAuxei[j] && d->ef == apagarDAuxef[j] && (strcmp(apagarDAuxle[j], d->le) == 0))
            {
                dSalve->prox = d->prox;
                d = dSalve;
                flagRemove = 1;
            }

            if(flagRemove == 0)
            {
                dSalve = d;
                d = d->prox;
            }
            else
                break;
        }
    }

    //chama concatena
    if(i < principal->F->f - 1)
        concatena(principal->D, ++i, principal, 1);
}
```

Função remover_delta()

- remover 1 lista t_delta

```
void remover_delta(int EI, char *LE, int EF, t_delta **cabeca)
{
    t_delta *copia = *cabeca;
    t_delta *plant = NULL;

    while(copia != NULL)
    {
        if(EI == copia->ei && EF == copia->ef && LE == copia->le)
            break;
        plant = copia;
        copia = copia->prox;
    }
    if(copia == NULL)
        return;
    if(plant != NULL)
        plant->prox = copia->prox;
    else
        *cabeca = copia->prox;
    free(copia);
    return;
}
```

Função transER()

- transformacao final para ER dos arcos que iniciam e finalizam no

```
while(d != NULL)
{
    //copia caminhos p/ string
    strcpy(str1, d->le);
    strcpy(STR1, d->le);

    //reinicia flag q indica alteracao
    salveFlagAconteceu = 0;
    dAux=quint->D;
    while(dAux != NULL)
    {
        //copia caminhos p/ string
        strcpy(str2, dAux->le);
        strcpy(STR2, dAux->le);

        //reinicia flag
        flagAconteceu = 0;

        //se possuírem mesmo tamanho
        if(strlen(str1) == strlen(str2) && d!=dAux)
        {
            for(int i = strlen(str1) - 1; i >= 0; i--)
            {
                //compara se possuir char iguais
                int compara = strcmp(str1, str2);
                if(compara != 0)//str1 possui carac diferente de str2
                {
                    str1[i] = '\0';
                    str2[i] = '\0';
                }
                //achamos trecho semelhante
                else
                    if(compara == 0)
                    {
                        //confirma que alguma alteracao vai ser feita
                        flagAconteceu = 1;

                        //salva flag
                        if(i == 1)
                            salveFlagAconteceu += flagAconteceu;

                        //copiamos trecho semelhante para buffer
                        strcpy(bufferLE, str1);
                        //salvo tamanho do trecho semelhante
                        int tamSemelhante = strlen(str1);
```

```
            }

            //confere se alguma alteracao foi feita
            if(flagAconteceu == 1)
            {
                //conta quantos expressoes serao adicionadas a matrizER
                ++contMatrizER;

                //inclui os pontos
                strcpy(bufferLE, bufferLeFinal);
                strcpy(bufferFinalMesmo, "");
                for(long unsigned int j = 0; j < strlen(bufferLE); j++)
                {
                    //adiciona a letra ao buffer
                    bufferCompara[0] = bufferLE[j];
                    bufferSinais[0] = bufferLE[j + 1];
                    if(strcspn(bufferCompara, "abcdefghijklmnopqrstuvwxyz*") == 0 && strcspn(bufferSinais, "|*) ") == 1)
                    {
                        strcat(bufferFinalMesmo, bufferCompara);
                        strcat(bufferFinalMesmo, ".");
                    }
                    else{
                        strcat(bufferFinalMesmo, bufferCompara);
                    }
                }
                //salva er
                strcpy(matrizER[contMatrizER-1], bufferFinalMesmo);
                if(dAux->prox == NULL)//fim da lista
                {
                    dAuxAnt->prox = dAux->prox;
                }
                else
                {
                    dAuxAnt->prox = dAux->prox;
                    dAuxAnt = dAuxAnt->prox;
                    dAux = dAuxAnt;
                }
            }
        }
        dAuxAnt = dAux;
        dAux = dAux->prox;
    }
```

Função transER()

- transformacao final para ER dos arcos que iniciam e finalizam no mesmo estado

```
//adiciona-se parênteses e barras verticais entre as expressões regulares salvas na matrizER
for(int p=0; p<contMatrizER; p++)
{
    if(p==0)
    {
        sprintf(bufferFinalMesmo, "(%s)",matrizER[p]);
    }
    else
    {
        strcat(bufferFinalMesmo, "|(");
        strcat(bufferFinalMesmo, matrizER[p]);
        strcat(bufferFinalMesmo, ")");
    }
}

//print sem espaco
for(long int unsigned c=0; c<=strlen(bufferFinalMesmo); c++, ++cFinal)
{
    if(c == strlen(bufferFinalMesmo))
    {
        printFinal[cFinal] = '\0';
        break;
    }
    else if(bufferFinalMesmo[c] == ' ')//espaco
    {
       --cFinal;//ainda iremos adicionar algo a casa e nao pode ser espaco
    }
    else
    {
        //mantemno caracter
        printFinal[cFinal] = bufferFinalMesmo[c];
    }
}
printf("%s\n", printFinal);
```

Transformação ER->AFND

Expressões regulares

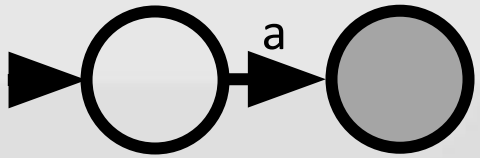
a|a.(a.b|b)*

a|a.(a.b|b)*

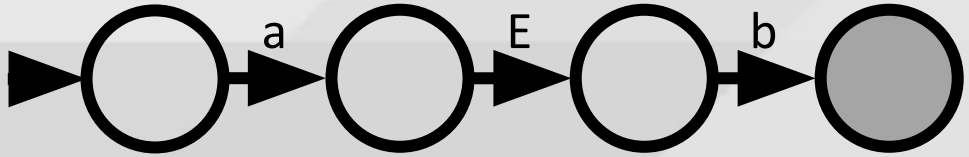
a|(a.(a.b|b))*

Conversão Manual ER->AFND

Exemplos Básicos:

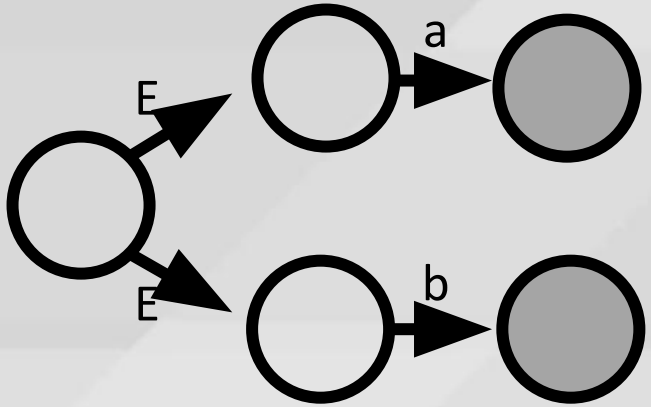


a



a.b

Conjunção

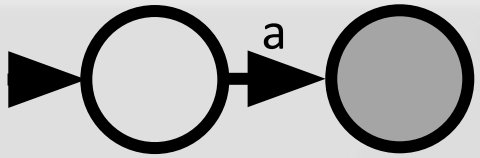


a|b

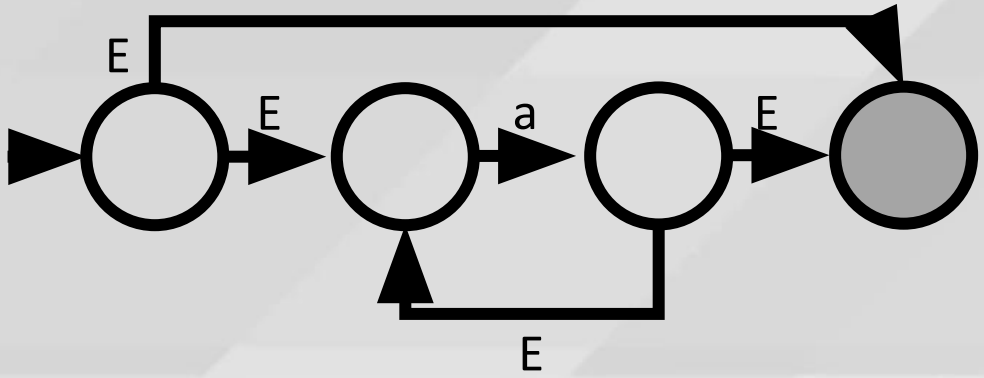
Disjunção

Conversão Manual ER->AFND

Exemplos Básicos:



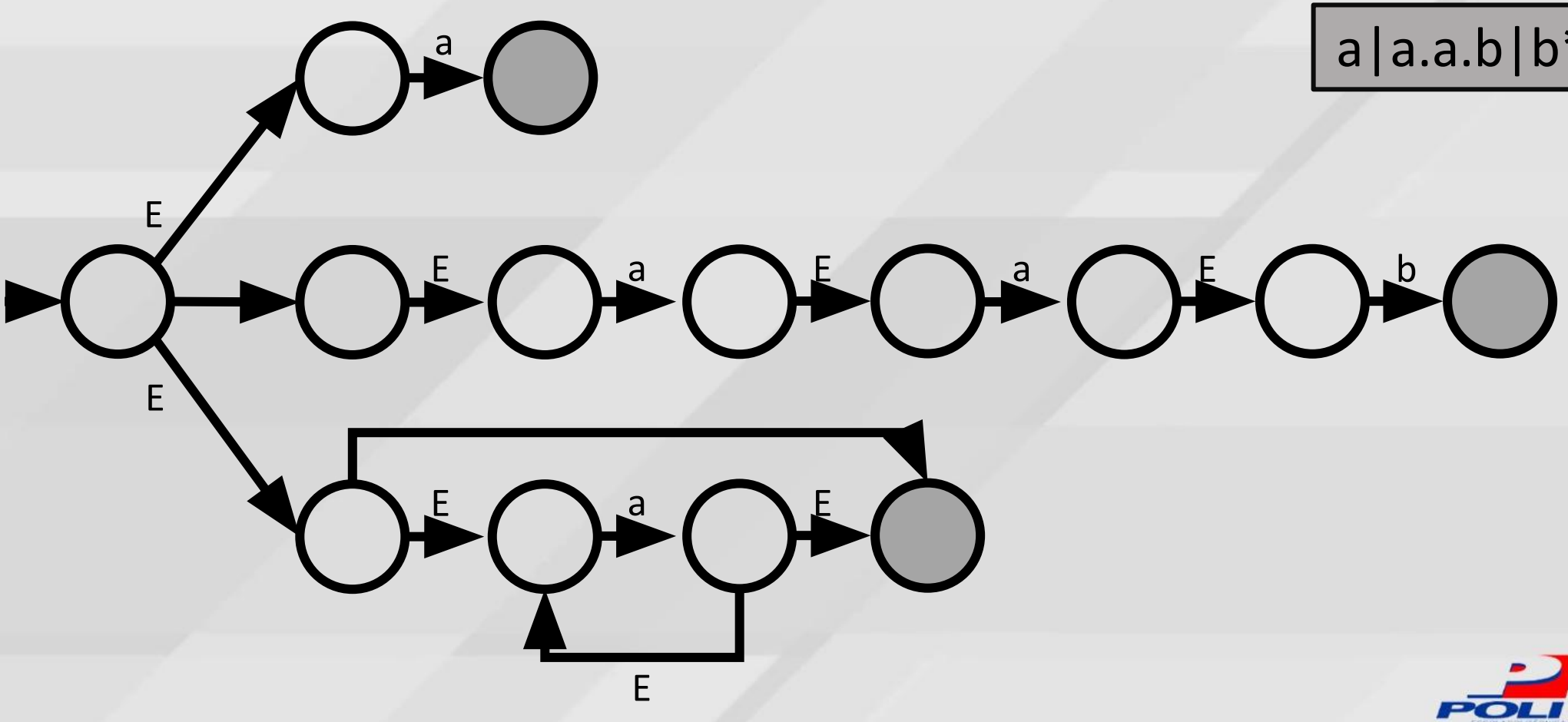
a



a*

Conversão Manual ER->AFND

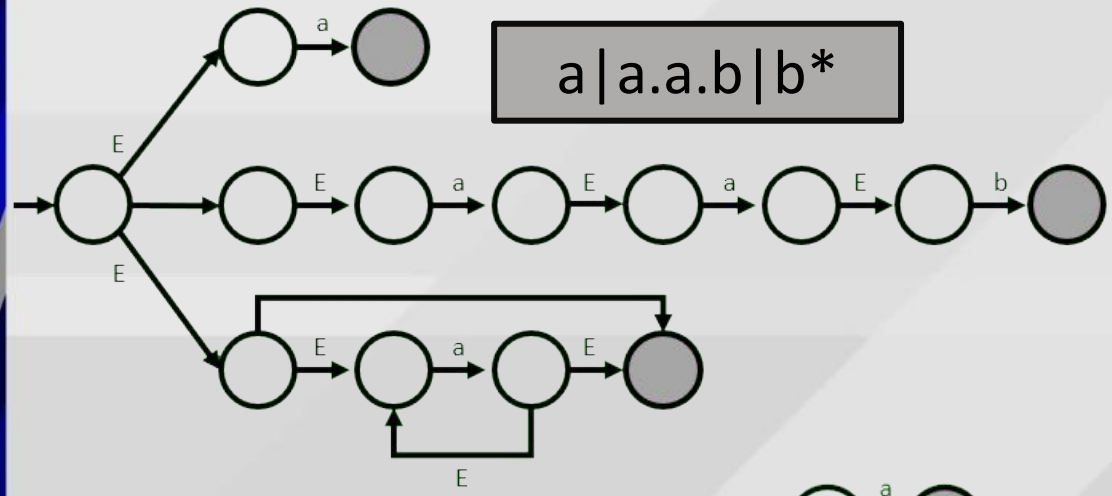
Exemplos Básicos:



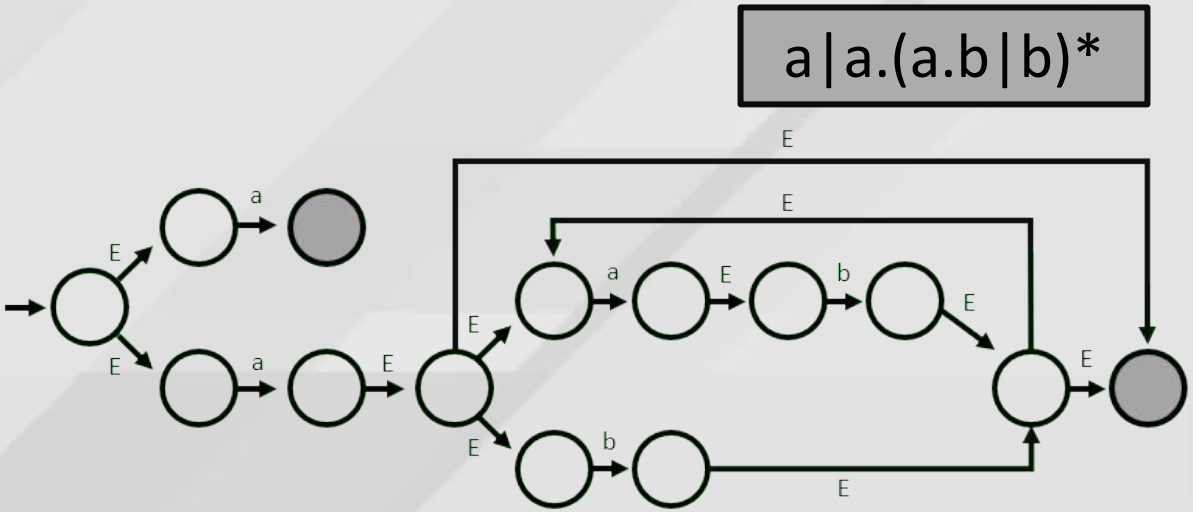
Conversão Manual ER->AFND

Prioridade de Operações:

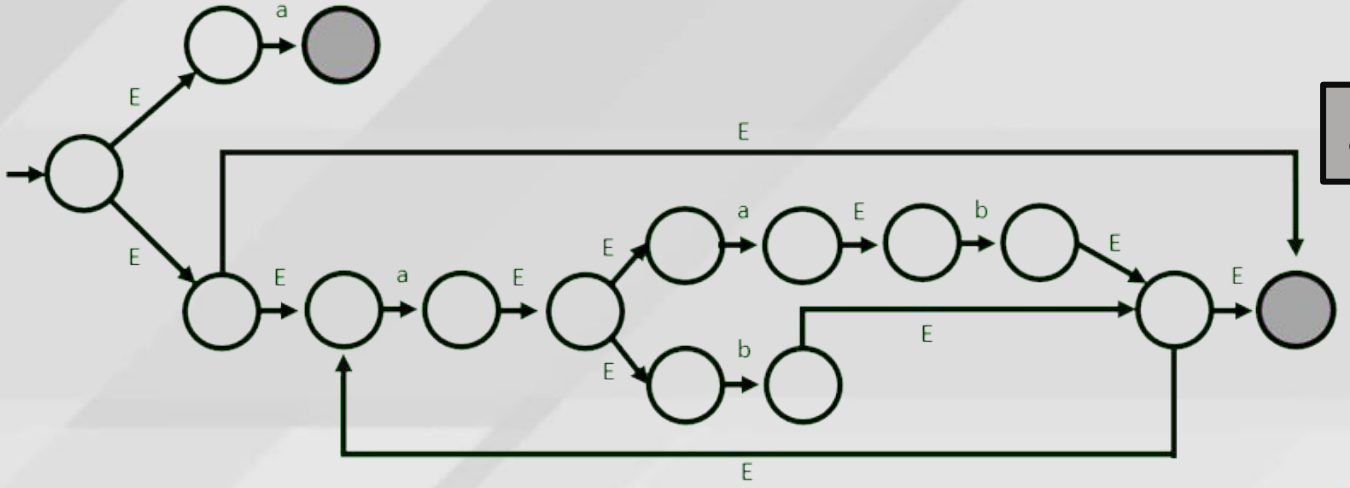
a|a.a.b|b*



a|a.(a.b|b)*



a|(a.(a.b|b))*



Função main()

```c
int main(int argc, char *argv[])
{
    int opt; /* return from getopt() */

    IFDEBUG("Starting optarg loop...");

    /* getopt() configured options:
     *        -h  help
     *        -c  copyright and version
     *        -v  verbose
     *        -f  given file name
     *        -n  entrada manual
     */
    opterr = 0;
    while((opt = getopt(argc, argv, "nvhcf:")) != EOF){
        switch(opt)
        {
            case 'h':
                help();
                break;
            case 'c':
                copyr();
                break;
            case 'n':
                entradaN();
                break;
            case 'v':
                verb++;
                break;
            case 'f':
                entradaF(optarg);
                break;
            case '?':
            default:
                printf("Type\n\t$man %s\nor\n\t$%s -h\nfor help.\n\n", argv[0], argv[0]);
                return EXIT_FAILURE;
        }
    }
```

Função help(-h) e copyright(-c)

```c
void help(void)
{
    IFDEBUG("help()");
    printf("%s v.%s - %s\n", "exN", VERSION, "Convert AFD into Exp Reg");
    printf("\nUsage: %s [-h|-v|-c|-f|-n]\n", "exN");
    printf("\nOptions:\n");
    printf("\t-h,  --help\n\t\tShow this help.\n");
    printf("\t-c,  --copyright, --version\n\t\tShow version and copyright information.\n");
    printf("\t-v,  --verbose\n\t\tSet verbose level (cumulative).\n");
    printf("\t-f, --file\n\t\tSet the input filename.\n");
    /* add more options here */
    printf("\t-n, --STDIN\n\t\tManual input.\n");
    printf("\n\nNote (-e): input data is a RE (Regular Expression) string to be converted to a NFA
    printf("\nExit status:\n\t0 if ok.\n\t1 some error occurred.\n");
    printf("\nTodo:\n\tLong options not implemented yet.\n");
    printf("\nAuthor:\n\tWritten by %s <%s>\n\n", "Ruben Carlo Benante", "rcb@beco.cc");
    exit(EXIT_FAILURE);
}
```

```c
void copyr(void)
{
    IFDEBUG("copyr()");
    printf("%s - Version %s\n", "exN", VERSION);
    printf("\nCopyright (C) %d %s <%s>, GNU GPL version 2 <http://gnu.org/licenses/gpl.html>. This  is  free  software: you are free to change and redistribute it. There is NO WARRANTY, t
o the extent permitted by law. USE IT AS IT IS. The author takes no responsability to any damage this software may inflige in your data.\n\n", 2016, "Ruben Carlo Benante", "rcb@beco.cc");
    if(verb > 3) printf("copyr(): Verbose: %d\n", verb); /* -vvvv */
    exit(EXIT_FAILURE);
}
```

```c
    if(verb)
        printf("Verbose level set at: %d\n", verb);

    //exN_init(); // global initialization function: ainda a saber para que usar se precisar
    /* ...and we are done */
    /* Write your code here... */
    return EXIT_SUCCESS;
}
```

POLI ESCOLA POLITÉCNICA DE PERNAMBUCO — UPE UNIVERSIDADE DE PERNAMBUCO

Função entradaN

```c
void entradaN(){
char str[SBUFF];
FILE *arq = NULL;
arq = fopen("temp.txt","a");
scanf("%s", str);
fprintf(arq,"%s",str);
fflush(arq);

FILE *read = NULL;
read = fopen("temp.txt","r");

le_arquivo(read);

remove("temp.txt");
return;

}
```

Função entradaF()

```c
void entradaF(char *file){

FILE *arquivo = NULL;

arquivo = fopen(file, "r");

if(arquivo == NULL){
printf("Arquivo nao encontrado\n");
return;
}

le_arquivo(arquivo);

return;
}
```

função le_arquivo()

```
void le_arquivo(FILE *exp_reg){

  char c; //Variável que receberá o caracter da leitura do arquivo de texto.
  char palavra[SBUFF] = ""; //Variável utilizada na chamada do allegro.
  int i = 0;
  int j = 0;
  int k = 0; //Variável que realiza o loop auxiliar.
  int contagem = 96; //Contagem dos caracteres ascii a partir do numero 96 (le
  int fsize = 0; //Variável que contará a qauntidade de caracteres no arquivo
  int p1 = 0; //Parêntese aberto. // ========== NOVA ADIÇÃO ==========
  int p2 = 0; //Parêntese fechado. // ========== NOVA ADIÇÃO ==========

  t_delta conteudo = {0}; //Variável que armazenará o conteúdo do arquivo de t

  if(exp_reg == NULL){ //Caso a leitura do arquivo seja nula, retorna para a f
      return;
```

```
while((c = fgetc(exp_reg)) != EOF){ //Executa o 'while' enquanto não chega ao fim do arquivo
    fsize++; //Incrementa +1 ma contagem de caracteres presentes no arquivo de texto.
    if(fsize == 1){
        conteudo.le = (char *)malloc(sizeof(char)*2); //Alocação de memória.

        conteudo.le[fsize] = '\0'; //Define a posição seguinte do array como final.
    }
    conteudo.le = (char *)realloc(conteudo.le, strlen(conteudo.le)+2); //Realocação de memória.
    conteudo.le[fsize-1] = c; //Subtrai 1 da posição do array, pois a contagem de 'fsize' inicia com '1'
    conteudo.le[fsize] = '\0'; //Define a posição seguinte do array como final.
    if(c != '\n'){
        palavra[k] = c;
    }
    // ========== NOVA ADIÇÃO ==========
    if(c == '('){
        p1++;
    }
    // ========== NOVA ADIÇÃO ==========
    if(c == ')'){
        p2++;
    }

    k++;
}
// ========== NOVA ADIÇÃO ==========
if(p1 != p2){
    exit(1);
}

if(strcmp(palavra, "a|a.a.b|b*") == 0){
    ex1_allegro();
}
if(strcmp(palavra, "a|a.(a.b|b)*") == 0){
    ex2_allegro();
}

if(strcmp(palavra, "a|(a.(a.b|b))*") == 0){
    ex3_allegro();
}

while(conteudo.le[i] != '\0'){
    for(j = 97; j < 123; j++){
        if(conteudo.le[i] == j){
            if(conteudo.le[i] != contagem){
                if(conteudo.le[i] > contagem){
                    contagem = j;
```

Função conversão_er_afnd()

```
if(linha[i] != '\n' && linha[i] != '*' && linha[i] != '.'){ //Caso o caracter lido seja uma letra.
    if(i == 0){
        if(*est != 0){ //Ao entrar nesse 'if', o '*est' terá o mesmo valor dos estados de aceitação.

            // ========== NOVA ADIÇÃO ==========
            pesquisa_ef = buscar_ef(F, *est);
            if(pesquisa_ef == 0){
                inserir_ef(&F, *est);
            }

        }
        *est += 1;
        organiza_listas(&list_auto->D, 0, E, *est);
    }

    if(linha[i+1] == '*'){
        estrela = 1;
        if(linha[i] == ')'){
            organiza_listas(&list_auto->D, *est, E, *est+1);
            *est += 1;

            // ========== NOVA ADIÇÃO ==========
            if(cont_ei == 0){
                montar_loop(list_auto, ei[1],*est);
            }
            // ========== NOVA ADIÇÃO ==========
            else{
                montar_loop(list_auto, ei[cont_ei],*est);
            }
        }
        else{
            montar_estrela(novo_afnd, linha[i], est);
        }
    }
    else{
        if(linha[i] == '(' || linha[i] == ')'){
            if(linha[i] == ')'){
                cont_ei -= 1;
            }
            else{
                organiza_listas(&list_auto->D, *est, E, *est+1);
                cont_ei += 1;
                ei[cont_ei] = *est;
                *est += 1;
            }
        }
```

```
void conversao_er_afnd(char linha[SBUFF], t_quintupla *novo_afnd, unsigned short int *est){

    int i = 0; //Variável que representará a posição de 'linha'.

    int cont_aux;
    int ascii;
    int est_aux;
    int contagem = 0;
    int estrela = 0;
    int pesquisa_ef = 0; // ========== NOVA ADIÇÃO ==========

    int ei[SBUFF]; //Estados iniciais.
    int cont_ei = 0; //Variável que representará a posição de 'ei'.
    char E = 'E'; //Variável que armazena a letra 'E', a qual simula a letra 'Épsilon'.

    t_quintupla *list_auto = novo_afnd;

    while(linha[i] != '\0'){ //Roda o laço até chegar ao fim da palavra.

        // ========== ALTERAÇÃO ==========
        if((linha[i] == '.') || ((linha[i] > 96 && linha[i] < 123) && linha[i+1] == '(') || (linha[i] == ')' && (linha[i+1] > 96 && linha[i+1] < 123)
            organiza_listas(&list_auto->D, *est, E, *est+1);
            *est += 1;
```