# Tradutor de linguagem NAG para Jason

Jonathan Santos e Maria Duarte

# O problema

Necessidade de tradução entre as linguagens

Funcionamento da tradução em ferramentas padrões do Jason

# Solução

- BNF
- Analisador Léxico - Flex
- Analisador Sintático - Bison
- Estruturas de dados
- Implementações

# 1. BNF Base

⟨*programa*⟩ ::= ⟨*Lagentes*⟩

⟨*Lagentes*⟩ ::= (⟨*agente*⟩ ‘‘%’’ )+

⟨*agente*⟩ ::= ‘‘#’’NAME crencas: ⟨*Lcrencas*⟩ objetivos: ⟨*Lobjetivos*⟩ planos: ⟨*Lplanos*⟩

⟨*Lcrencas*⟩ ::= (‘‘{’’ ⟨*nomeCrenca*⟩ ‘‘;’’* ‘‘}’’)

⟨*nomeCrenca*⟩ ::= NAME

⟨*Lobjetivos*⟩ ::= (‘‘{’’ ⟨*nomeObjetivo*⟩ ‘‘;’’* ‘‘}’’)

⟨*nomeObjetivo*⟩ ::= NAME

⟨*Lplanos*⟩ ::= (‘‘{’’ ⟨*nomePlano*⟩ ‘‘;’’* ‘‘}’’)

⟨*nomePlano*⟩ ::= NAME ⟨*tuplaPlano*⟩

⟨*tuplaPlano*⟩ ::= ‘‘(’’ ⟨*eventoGatilho*⟩ ‘‘;’’ ⟨*contexto*⟩ ‘‘;’’ ⟨*corpo*⟩ ‘‘)’’

⟨*eventoGatilho*⟩ ::= NAME

⟨*contexto*⟩ ::= ⟨*expressaoLogica*⟩ | NAME | $\varepsilon$

⟨*expressaoLogica*⟩ ::= NAME ‘‘E’’ NAME | NAME ‘‘OU’’ NAME | ‘‘NAO’’ NAME

⟨*corpo*⟩ ::= ‘‘{’’ ⟨*formulasCorpo*⟩ ‘‘;’’ ‘‘}’’

⟨*formulasCorpo*⟩ ::= NAME

# 2. BNF Final

```
<program> ::= <agents>
<agents> ::=  ε
            | <agent> '%' <agents>
<agent> ::= '#' NAME BELIEFS ':' <beliefs> GOALS ':' <goals> PLANS ':' '{' <plans> '}'
<beliefs> ::= '{' <beliefsName> '}'
<beliefsName> ::=  ε
        | NAME ';' <beliefsName>
<goals> ::= '{' <goalsName> '}'
<goalsName> ::=  ε
        | NAME ';' <goalsName>
<plans> ::= ε
            | <plan> ';' <plans>
<plan> ::=  ε
        | NAME <plansTuple>
<plansTuple> ::= '(' <triggerEvent> ';' <context> ';' <body> ')'
<tiggerEvent> ::= NAME
<context> ::=  ε
      | <logExp>
      | NAME
<logExp> ::= NAME AND NAME
     | NAME OR NAME
     | NOT NAME
<body> ::= '{' <bodysFormula> '}'
<bodysFormula> ::=  ε
      | NAME ';' <bodysFormula>
```

# 3. FLEX

```
%option noyywrap nodefault yylineno

%{
    #include "header.h"
    #include "analise-sintatica.tab.h"
%}

%%

":" |
";" |
"{" |
"}" |
"(" |
"%" |
"#" |
")"                 { return yytext[0]; }
"E"                 { return AND; }
"OU"                { return OR; }
"NAO"               { return NOT; }
"crencas"           { return BELIEFS; }
"objetivos"         { return GOALS; }
"planos"            { return PLANS; }
[a-z][a-zA-Z0-9]*   { yylval.s = strdup(yytext); return NAME; }
[ \t\r]+            { }
[\n]                { }
[\\\n]              { }
.                   { }

%%
```

# 4. Bison

```
%{
    #include<stdio.h>
    #include<stdlib.h>
    #include "header.h"
%}

%union {
    struct agents *a;
    struct believes *b;
    struct goals *g;
    struct plans *p;
    struct body *bo;
    struct planContent *pc;
    void *v;
    char *s;
}

%token  <s> NAME BELIEFS GOALS PLANS
%token  <s> OR NOT AND

%type <a> agent agents
%type <s> triggerEvent context logExp
%type <b> beliefs beliefsName
%type <g> goals goalsName
%type <p> plans plan
%type <pc> plansTuple
%type <bo> body bodysFormula
%type <v> initial;

%start initial

%%
```

# 4. Bison

```
initial: agents { printList($1); }
    ;
agents:  { $$ = NULL; }
    | agent '%' agents { $$ = prependAgent($3, $1); }
    ;
agent: '#' NAME BELIEFS ':' beliefs GOALS ':' goals PLANS ':' '{' plans '}' { $$ = createAgent($2, $5, $8, $12);}
    ;
beliefs: '{' beliefsName '}'  { $$ = $2; }
    ;
beliefsName: { $$ = NULL; }
    | NAME ';' beliefsName { $$ = prependBelieve($3, $1); }
    ;
goals: '{' goalsName '}'  { $$ = $2; }
    ;
goalsName: { $$ = NULL; }
    | NAME ';' goalsName { $$ = prependGoal($3, $1); }
    ;
plans:  { $$ = NULL; }
    |  plan ';' plans  { $$ = prependPlan($3, $1);}
    ;
plan:  { $$ = NULL; }
    | NAME plansTuple { $$ = createPlan($1, $2); }
    ;
plansTuple:  '(' triggerEvent ';' context ';' body ')' { $$ = createContent($2, $4, $6); }
    ;
triggerEvent: NAME { $$ = $1; }
    ;
context: { $$ = NULL; }
    | logExp { $$ = $1; }
    | NAME { $$ = $1; }
    ;
logExp: NAME AND NAME { $$ = newExp($1, $3, "E");}
    | NAME OR NAME { $$ = newExp($1, $3, "OU"); }
    | NOT NAME { $$ = newExp(NULL, $2, "NAO");  }
    ;
body: '{' bodysFormula '}' { $$ = $2; }
    ;
bodysFormula:  { $$ = NULL; } |
    NAME ';' bodysFormula { $$ = prependBody($3, $1); }
    ;
%%
```

# Estrutura de dados

- Estrutura de listas
  - Maior facilidade
- Estrutura dividida
- Junção da lista no final

# 5. Estrutura de dados

```c
struct believes
{
  char *believes;
  struct believes *next;
};

struct goals
{
  char *goals;
  struct goals *next;
};

struct body
{
  char *body;
  struct body *next;
};

struct planContent
{
  char *triggerEvent;
  char *context;
  struct body *body;
};
struct plans
{
  char *name;
  struct planContent *planContent;
  struct plans *next;
};

struct agents
{
  char *name;
  struct believes *believes;
  struct goals *goals;
  struct plans *plans;
  struct agents *next;
};
```

# Implementação

- Inserção no início
- Armazenamento primeiro, depois escrita
- Múltiplos agentes escritos em arquivos diferentes de saída

# 6. Implementações

```c
// believes
struct believes *prependBelieve(struct believes *believes, char *newBelieve);
// goals
struct goals *prependGoal(struct goals *goals, char *newGoal);
// planContent
struct planContent *createPlanContent(char *triggerEvent, char *context, struct body *body);

struct body *prependBody(struct body *body, char *newBody);
// plans
struct plans *createPlan(char *name, struct planContent *planContent);

struct plans *prependPlan(struct plans *plans, struct plans *newPlan);

struct planContent *createContent(char *triggerEvent, char *context, struct body *body);
struct agents *createAgent(char *name, struct believes *believes, struct goals *goals, struct plans *plans);
struct agents *prependAgent(struct agents *agents, struct agents *newAgent);
void printAgent(struct agents *agents);
void printList(struct agents *list);
void printAgentInFile(struct agents *agent);
// exp
char *newExp(char *leftSide, char *rightSide, char *operator);
```

# 6. Implementações

```
struct believes *prependBelieve(struct believes *believes, char *newBelieve)
{
  if (newBelieve == NULL)
  {
    return believes;
  }

  struct believes *new = (struct believes *)malloc(sizeof(struct believes));

  new->believes = (char *)malloc(sizeof(char) * strlen(newBelieve) + 2);

  new->believes = newBelieve;

  strcat(new->believes, ".");
  new->next = believes;

  return new;
}
```

```
struct goals *prependGoal(struct goals *goals, char *newGoal)
{
  if (newGoal == NULL)
  {
    return goals;
  }

  struct goals *new = (struct goals *)malloc(sizeof(struct goals));

  char *formatedGoal = (char *)malloc(sizeof(char) * strlen(newGoal) + 4);
  strcat(formatedGoal, "!");
  strcat(formatedGoal, newGoal);
  strcat(formatedGoal, ".");

  new->goals = formatedGoal;
  new->next = goals;

  return new;
}
```

```
struct body *prependBody(struct body *body, char *newBody)
{
  if (newBody == NULL)
  {
    return body;
  }
  if (body == NULL)
  {
    struct body *new = (struct body *)malloc(sizeof(struct body));

    new->body = (char *)malloc(sizeof(char) * strlen(newBody) + 100);

    strcat(new->body, " .printf(\"");
    strcat(new->body, newBody);
    strcat(new->body, "\").");
    new->next = body;

    return new;
  }

  struct body *aux = body;

  struct body *new = (struct body *)malloc(sizeof(struct body));

  new->body = (char *)malloc(sizeof(char) * strlen(newBody) + 100);

  strcat(new->body, " .printf(\"");
  strcat(new->body, newBody);
  strcat(new->body, "\");");
  new->next = body;

  return new;
}
```

# 6. Implementações

```c
struct planContent *createContent(char *triggerEvent, char *context, struct body *body)
{
  struct planContent *new = (struct planContent *)malloc(sizeof(struct planContent));

  if (new == NULL)
  {
    yyerror("out of memory");
    exit(0);
  }

  new->triggerEvent = (char *)malloc(sizeof(char) * strlen(triggerEvent) + 4);
  strcat(new->triggerEvent, "+!");
  strcat(new->triggerEvent, triggerEvent);
  strcat(new->triggerEvent, ":");
  new->context = (char *)malloc(sizeof(char) * strlen(context) + 3);
  strcat(new->context, context);
  strcat(new->context, " <-");
  new->body = body;

  return new;
}
```

```c
struct plans *createPlan(char *name, struct planContent *planContent)
{
  struct plans *new = (struct plans *)malloc(sizeof(struct plans));

  if (new == NULL)
  {
    yyerror("out of memory");
    exit(0);
  }

  new->name = name;
  new->planContent = planContent;
  new->next = NULL;

  return new;
}

struct plans *prependPlan(struct plans *plans, struct plans *newPlan)
{
  if (newPlan == NULL)
  {
    return plans;
  }

  newPlan->next = plans;

  return newPlan;
}
```

```c
char *newExp(char *leftSide, char *rightSide, char *operator)
{
  char *new = (char *)malloc(1000);
  if (!strcmp(operator, "E"))
  {
    strcpy(new, leftSide);
    strcat(new, " & ");
    strcat(new, rightSide);
  }
  else if (!strcmp(operator, "OU"))
  {
    strcpy(new, leftSide);
    strcat(new, " | ");
    strcat(new, rightSide);
  }
  else if (!strcmp(operator, "NAO") && leftSide == NULL)
  {
    strcpy(new, "not ");
    strcat(new, rightSide);
  }
  else if (!strcmp(operator, "NAO") && rightSide == NULL)
  {
    strcpy(new, "not ");
    strcat(new, leftSide);
  }
  return new;
}
```

# 6. Implementações

```
struct agents *createAgent(char *name, struct believes *believes, struct goals *goals, struct plans *plans)
{
  struct agents *new = (struct agents *)malloc(sizeof(struct agents));

  if (new == NULL)
  {
    yyerror("out of memory");
    exit(0);
  }

  new->name = name;
  new->believes = believes;
  new->goals = goals;
  new->plans = plans;
  new->next = NULL;

  return new;
}
```

```
struct agents *prependAgent(struct agents *agents, struct agents *newAgent)
{
  if (newAgent == NULL)
  {
    return agents;
  }

  newAgent->next = agents;

  agents = newAgent;

  return agents;
}
```

```
void printAgentInFile(struct agents *agent)
{
  struct stat st = {0};

  if (stat("output", &st) == -1)
  {
    mkdir("output", 0700);
  }

  char *agentName = (char *)malloc(sizeof(char) * strlen(agent->name) + 16);
  strcpy(agentName, "output/");
  strcat(agentName, agent->name);
  strcat(agentName, ".asl");
  FILE *f = fopen(agentName, "w");
  struct believes *auxBelieves = agent->believes;
  while (auxBelieves != NULL)
  {
    fprintf(f, "%s\n", auxBelieves->believes);
    auxBelieves = auxBelieves->next;
  }
  fprintf(f, "\n\n");

  struct goals *auxGoals = agent->goals;
  while (auxGoals != NULL)
  {
    fprintf(f, "%s\n", auxGoals->goals);
    auxGoals = auxGoals->next;
  }

  fprintf(f, "\n\n");

  struct plans *auxPlans = agent->plans;
  while (auxPlans != NULL)
  {

    fprintf(f, "%s", auxPlans->planContent->triggerEvent);
    fprintf(f, " %s", auxPlans->planContent->context);

    struct body *aux = auxPlans->planContent->body;

    while (aux != NULL)
    {
      fprintf(f, " %s\n", aux->body);
      aux = aux->next;
    }

    auxPlans = auxPlans->next;
  }

  fprintf(f, "\n");
  free(agentName);
  agentName = NULL;
  fclose(f);
}
```

# Obrigado!