Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Bacharelado em Ciência da Computação

COMPILADORES

Construção de um Tradutor da Linguagem Natural NAG para a Linguagem Jason/AgentSpeak

Francisco Correa Neto
Júlio César Guimarães Costa

PONTA GROSSA
10/2022

SUMÁRIO

1. INTRODUÇÃO

O seguinte relatório de projeto da disciplina de Compiladores da Universidade Tecnológica Federal do Paraná , tem como objetivo implementar um tradutor da linguagem natural NAG para a linguagem JASON/AgentSpeak, utilizando análises léxicas e sintáticas com as ferramentas Flex e Bison.

2. MATERIAL E MÉTODOS

2.1. MATERIAL

No presente trabalho foi realizada a tradução entre as linguagens utilizando o analisador sintático Flex e o analisador léxico Bison, com a utilização de um ASL, uma estrutura de dados projetada e construída com a utilização da linguagem C.

2.2. METODOLOGIA

Para a realização deste projeto tem-se como entrada arquivos em linguagem natural NAG, estes seguindo uma estrutura padrão a ser analisada sintaticamente, destacando o exemplo apresentado no enunciado do projeto (Figura 1). Também foram criados diversos arquivos de entrada diferentes para uma melhor avaliação dos resultados obtidos.

Figura 1 - Arquivo de entrada

```
bob crencas: { estaChovendo ; naotenhoGuardaChuva ; }
objetivos: { comprarGuardaChuva ; naoPegarChuva ; }
planos: { plano1 ( comprarGuardaChuva
; estaChovendo E naotenhoGuardaChuva
;{ sair; procurarLoja; comprarGuardaChuva; } ) ; }
```

Fonte: Os autores (2022).

Como um segunda etapa foi realizada a análise sintática junto ao analisador Flex, e uma análise léxica por meio do analisador Bison, utilizando o sistema operacional Linux, por fim, a saída do algoritmo será um arquivo na extensão ".asl", formato referente a linguagem JASON/AgentSpeak, este, foi testado e compilado no aplicativo jEdit.

Para a compilação do algoritmo foi utilizado um arquivo *MakeFile*, arquivo realizado para uma melhor organização das linhas de comando (Figura 2).

Figura 2 - Arquivo MakeFile

```
all: Tradutor

Tradutor.tab.c Tradutor.tab.h: Tradutor.y
	bison -t -v -d Tradutor.y

lex.yy.c: Tradutor.l Tradutor.tab.h
	flex Tradutor.l

Tradutor: lex.yy.c Tradutor.tab.c Tradutor.tab.h
	gcc Tradutor.h Tradutor.c -o Tradutor Tradutor.tab.c lex.yy.c

clean:
	rm Tradutor Tradutor.tab.h Tradutor.tab.c lex.yy.c Tradutor.output
```

Fonte: Os autores (2022).

3. RESULTADOS E DISCUSSÃO

3.1. REPRESENTAÇÃO DA GRAMÁTICA

A gramática, na Figura 3, está representada no formato BNF.

Figura 3 - Gramática

```
<nome_crenca> ::= ID

<lista_crencas> ::= pal.vazia
                  | <nome_crenca> ; <lista_crencas>
                  | {<lista_crencas>}

<nome_objetivo> ::= ID

<lista_objetivos> ::= pal.vazia
                  | <nome_objetivo> ; <lista_objetivos>
                  | {<lista_objetivos>}

<evento_gatilho> ::= ID

<expressaoLogica> ::= ID OPERADOR ID
                  | OPERADOR ID

<contexto> ::= pal. vazia
        |ID
        |<expressaoLogica>

<formula_corpo> ::= pal.vazia
        |ID ; <formulaCorpo>

<corpo> ::= {<formula_corpo>}

<nome_plano> ::= ID

<lista_planos> ::= pal.vazia
        |<nome_plano> ; <lista_planos>
        |{<lista_planos>}

<nome_agente> ::= ID

<stmt> ::= pal.vazia
        | PLANOS {<lista_planos>}
        | OBJETIVOS {<lista_objetivos>}
        | <nome_agente> CRENCAS {<lista_crencas>}

<lista_stmt> ::= {gera_Jason(agent)}
        | <stmt> EOL <list_stmt>
        | 0
```

Fonte: Os autores (2022).

3.2. ANALISADOR LÉXICO FLEX

A seguir, na Figura 4, o código do arquivo Flex.

Figura 4 - Flex

```
%{
        /* Analisador Léxico TradutorNAG_Jason */
    #include "Tradutor.h"
    #include "Tradutor.tab.h"
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
%}

%option yylineno
%option noyywrap

LETTER [a-zA-Z_]
DIGIT [0-9]
ID {LETTER}({LETTER}|{DIGIT})*

%%



"crencas:"                          { return CRENCAS; }
"objetivos:"                    { return OBJETIVOS; }
"planos:"                           { return PLANOS; }
"E"|"OU"|"NAO"                  { yylval.op = strdup(yytext); return OPERADOR; }
"("|")"|"{"|"}"|";"             { return yytext[0]; }
{ID}                                { yylval.id = strdup(yytext); return ID; }
"\n"                            { return EOL; }
<<EOF>>                         { return 0; }
[ \t]                               ;
.                                           { printf("Caracter desconhecido: %s\n", yytext); return 0; }

%%

/* definido pelo analisador léxico */
extern FILE * yyin;

int main(int argc, char ** argv)
{
        /* se foi passado um nome de arquivo */
        if (argc > 1)
        {
                FILE * file;
                file = fopen(argv[1], "r");
                if (!file)
                {
                        exit(1);
                }

                /* entrada ajustada para ler do arquivo */
                yyin = file;
        }
    else
    {
        printf("Nenhum arquivo de entrada encontrado.");
        exit(1);
    }

        yyparse();
}
```

Fonte: Os autores (2022).

3.3. ANALISADOR SINTÁTICO BISON

A seguir, na Figura 5, o código do arquivo Bison.

Figura 5 - Bison

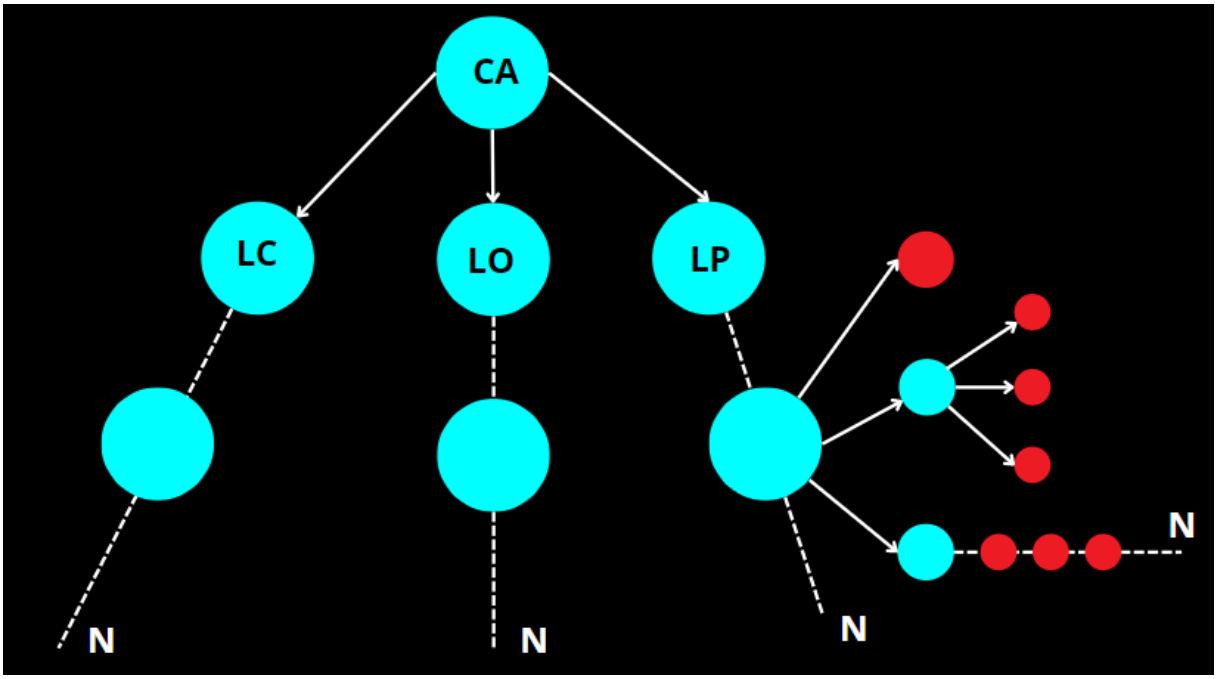
```
%{
        /* Analisador Sintático TradutorNAG_Jason */
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "Tradutor.h"
    #include "Tradutor.tab.h"
    int yylex();

    struct agent_head *agent;
    char *currentEventoGat;
    struct expressao_log *currentExp;
    struct corpo_list *currentCorpo = NULL;
%}

%union {
    char *id;
    char *op;
}

%token <id> ID
%token <op> OPERADOR
%token CRENCAS OBJETIVOS PLANOS
%token EOL

%start list_stmt
%%

/* LIDANDO COM AS CRENCAS */

nome_crenca: ID { add_crenca(agent, $1); }
    ;

lista_crencas: /* Vazio */
        | nome_crenca ';' lista_crencas
    | '{' lista_crencas '}'
        ;

/* LIDANDO COM OS OBJETIVOS */

nome_objetivo: ID { add_objetivo(agent, $1); }
    ;

lista_objetivos: /* Vazio */
        | nome_objetivo ';' lista_objetivos
    | '{' lista_objetivos '}'
        ;

/* LIDANDO COM OS PLANOS */

evento_gatilho: ID              { currentEventoGat = $1; }
    ;

expressaoLogica: ID OPERADOR ID { currentExp = create_exp($1, $2, $3); }
        | OPERADOR ID               { currentExp = create_exp("", $1, $2); }
    ;

contexto: /* Vazio */
    | ID                        { currentExp = create_exp("", "", $1); }
    | expressaoLogica
    ;

formula_corpo: /* Vazio */
    | ID ';' formula_corpo { currentCorpo = addCorpoList(currentCorpo, $1); }
    ;

corpo: '{' formula_corpo '}'
    ;

nome_plano: ID '(' evento_gatilho ';' contexto ';' corpo ')'
            {
                add_plano(agent, $1, currentEventoGat, currentExp, currentCorpo);
                currentCorpo = NULL;
            }
    ;

lista_planos: /* Vazio */
    | nome_plano ';' lista_planos
    | '{' lista_planos '}'
    ;

/* LIDANDO COM OS COMANDOS */

nome_agente: ID { agent = create_agent($1); }
    ;

stmt:
        | PLANOS '{' lista_planos '}'
        | OBJETIVOS     '{' lista_objetivos '}'
    | nome_agente CRENCAS '{' lista_crencas '}'
        ;

list_stmt: { gera_Jason(agent); }
        | stmt EOL list_stmt
        | '0'
        ;

%%
```

Fonte: Os autores (2022).

3.4. AST

Nos baseamos na estrutura da AST, e construímos uma estrutura híbrida com listas encadeadas que melhor se adaptava a solução do problema, ilustrada na Figura 6.

Figura 6 - AST Ilustrada



Fonte: Os autores (2022).

A estrutura parte de “CA”, que ficam os atributos da definição do agente, ela se ramifica em “LC”, “LO” e LP”.

O ramo “LC” é o responsável pela armazenagem de crenças do agente, sendo criada uma nova folha para cada crença inserida.

O ramo “LO” é o responsável pela armazenagem de objetivos do agente, sendo criada uma nova folha para cada crença inserida.

O ramo "LP" é o responsável pela armazenagem de planos do agente, sendo criada uma folha para cada plano inserido. Note que cada plano, se ramifica em três outros critérios, sendo o primeiro, guardar o evento gatilho, o segundo o contexto (E, OU, NAO), sendo esta estratégia variável e portanto, ao receber 2 dados, a primeira folha será tratada como nula. E por fim o terceiro ramo de cada vértice da LP, armazena o corpo do plano.

Na Figura 7, o código, em C, das estruturas de dados.

Figura 7 - Header das funções auxiliares

```
extern int yylineno;
void yyerror(char *s, ...);

struct expressao_log {
    char contexto1[50];
    char complemento[50];
    char contexto2[50];
};

struct corpo_list {
    char name[50];
    struct corpo_list *prox;
};

/* Listas Principais */
struct list_node {
    char name[50];
    char evento_gatilho[50];
    struct expressao_log *expressaoCont;
    struct corpo_list *listaCorpo;
    struct list_node *prox;
};

/* Cabeca Agente */
struct agent_head {
    char name_agent[50];
    struct list_node *listaCrencas;
    struct list_node *listaObjetivos;
    struct list_node *listaPlanos;
};

struct agent_head *create_agent(char *name);

void add_crenca(struct agent_head *a, char *name);
void add_objetivo(struct agent_head *a, char *name);
void add_plano(struct agent_head *a, char* nameP, char* eg, struct expressao_log *expLog, struct corpo_list *corpo);

struct expressao_log *create_exp(char *name1, char *comp, char *name2);
struct corpo_list *addCorpoList(struct corpo_list *cl, char *name);

void gera_Jason(struct agent_head *a);
```

Fonte: Os autores (2022).

3.5. TRADUÇÃO

Para a realização da tradução foram utilizadas funções em C, onde as mesmas armazenavam nas estruturas de dados os dados provenientes da entrada, traduziam e criavam e escreviam no arquivo de saída.

Os códigos estão demonstrados nas Figuras 8, 9, 10 , 11 e 12.

Figura 8 - Funções Auxiliares

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "Tradutor.h"
int yyparse();

void yyerror(char *s, ...)
{
    va_list ap;
    va_start(ap, s);
    fprintf(stderr, "%d: error: ", yylineno);
    vfprintf(stderr, s, ap);
    fprintf(stderr, "\n");
}

struct agent_head *create_agent(char* name)
{
    struct agent_head *newAgent = malloc(sizeof(struct agent_head));
    if (!newAgent){
        yyerror("Sem Espaco");
        exit(1);
    }
    strcpy(newAgent->name_agent, name);
    newAgent->listaCrencas = NULL;
    newAgent->listaObjetivos = NULL;
    newAgent->listaPlanos = NULL;
    printf("AGENTE <%s>\n", newAgent->name_agent);
    return newAgent;
}
```

Fonte: Os autores (2022).

Figura 9 - Funções Auxiliares

```
void add_crenca(struct agent_head *a, char* name)
{
    struct list_node *newCrenca = malloc(sizeof(struct list_node));
    if (newCrenca)
    {
        strcpy(newCrenca->name, name);
        newCrenca->prox = a->listaCrencas;
        a->listaCrencas = newCrenca;
        printf("CRENCA <%s>\n", newCrenca->name);
    }
    else
    {
        yyerror("Sem Espaco");
        exit(1);
    }
}

void add_objetivo(struct agent_head *a, char* name)
{
    struct list_node *newObjetivo = malloc(sizeof(struct list_node));
    if (newObjetivo)
    {
        strcpy(newObjetivo->name, name);
        newObjetivo->prox = a->listaObjetivos;
        a->listaObjetivos = newObjetivo;
        printf("OBJETIVO <%s>\n", newObjetivo->name);
    }
    else
    {
        yyerror("Sem Espaco");
        exit(1);
    }
}
```

Fonte: Os autores (2022).

Figura 10 - Funções Auxiliares

```c
void add_plano(struct agent_head *a, char* nameP, char* eg, struct expressao_log *expLog, struct corpo_list *corpo)
{
    struct list_node *newPlano = malloc(sizeof(struct list_node));
    if (newPlano)
    {
        strcpy(newPlano->name, nameP);
        strcpy(newPlano->evento_gatilho, eg);
        newPlano->expressaoCont = expLog;
        newPlano->listaCorpo = corpo;

        newPlano->prox = a->listaPlanos;
        a->listaPlanos = newPlano;

        printf("PLANO <%s> :", newPlano->name);
        printf(" GATILHO -> %s", newPlano->evento_gatilho);
        printf(" | CONTEXTO -> %s %s %s", newPlano->expressaoCont->contexto1, newPlano->expressaoCont->complemento, newPlano->expressaoCont->contexto2);
        printf(" | CORPO -> ");
        struct corpo_list *auxcorpo = newPlano->listaCorpo;
        while (auxcorpo != NULL)
        {
            printf("<%s> ", auxcorpo->name);
            auxcorpo = auxcorpo->prox;
        }
        printf("\n");
    }
    else
    {
        yyerror("Sem Espaco");
        exit(1);
    }
}
```

Fonte: Os autores (2022).

Figura 11 - Funções Auxiliares

```c
struct expressao_log *create_exp(char *name1, char *comp, char *name2)
{
    struct expressao_log *newExp = malloc(sizeof(struct expressao_log));
    if (newExp)
    {
        strcpy(newExp->contexto1, name1);
        strcpy(newExp->complemento, comp);
        strcpy(newExp->contexto2, name2);

        return newExp;
    }
    else
    {
        yyerror("Sem Espaco");
        exit(1);
    }
}

struct corpo_list *addCorpoList(struct corpo_list *cl, char *name)
{
    struct corpo_list *newCorpoValue = malloc(sizeof(struct corpo_list));
    if (newCorpoValue)
    {
        strcpy(newCorpoValue->name, name);
        newCorpoValue->prox = NULL;
        if (cl == NULL) return newCorpoValue;
        newCorpoValue->prox = cl;
        cl = newCorpoValue;
        return cl;
    }
    else
    {
        yyerror("Sem Espaco");
        exit(1);
    }
}
```

Fonte: Os autores (2022).

Figura 12 - Funções Auxiliares

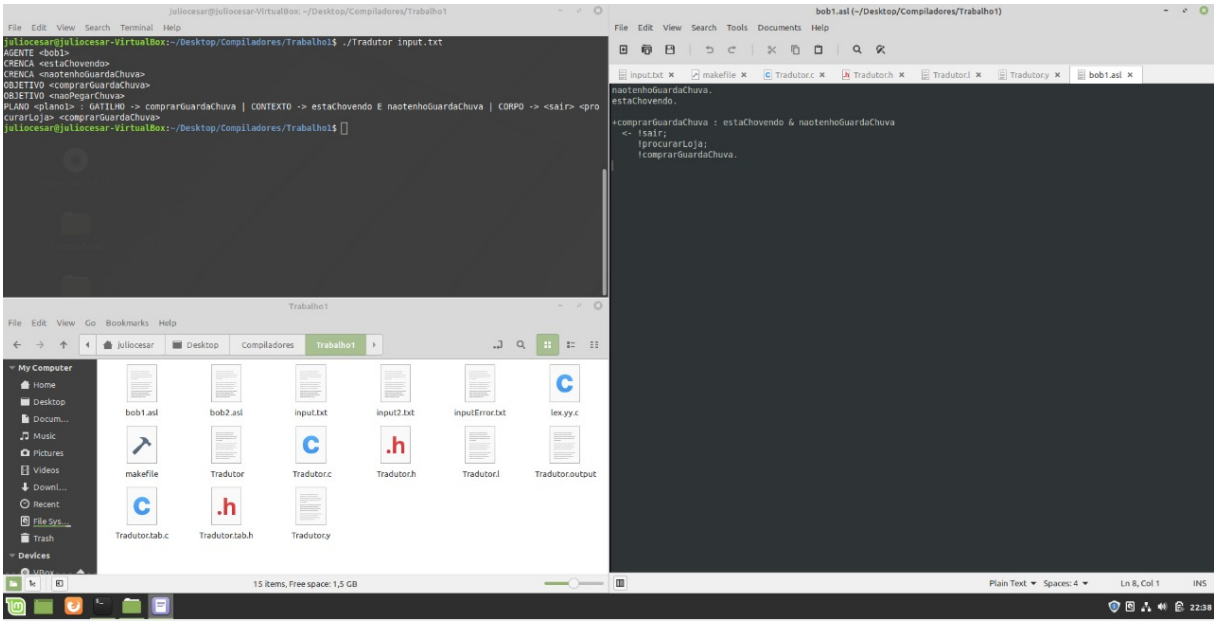
```
void gera_Jason(struct agent_head *a)
{
    FILE *fptr;
    struct list_node *auxCrenca = a->listaCrencas;
    struct list_node *auxObjetivos = a->listaObjetivos;
    struct list_node *auxPlanos = a->listaPlanos;
    struct corpo_list *auxcorpo;

    char finalAddress[100] = "./";
    char termination[] = ".asl";
    strcat(finalAddress, a->name_agent);
    strcat(finalAddress, termination);

    fptr = fopen(finalAddress, "w");
    while (auxCrenca != NULL)
    {
        fprintf(fptr, "%s.\n", auxCrenca->name);
        auxCrenca = auxCrenca->prox;
    }
    fprintf(fptr, "\n");
    while (auxPlanos != NULL)
    {
        fprintf(fptr, "+%s : ", auxPlanos->evento_gatilho);
        if (!strcmp(auxPlanos->expressaoCont->complemento, "E")) fprintf(fptr, "%s & %s\n", auxPlanos->expressaoCont->contexto1, auxPlanos->expressaoCont->contexto2);
        if (!strcmp(auxPlanos->expressaoCont->complemento, "OU")) fprintf(fptr, "%s | %s\n", auxPlanos->expressaoCont->contexto1, auxPlanos->expressaoCont->contexto2);
        if (!strcmp(auxPlanos->expressaoCont->complemento, "NAO")) fprintf(fptr, "not %s\n", auxPlanos->expressaoCont->contexto2);
        if (!strcmp(auxPlanos->expressaoCont->complemento, "")) fprintf(fptr, "%s\n", auxPlanos->expressaoCont->contexto2);

        auxcorpo = auxPlanos->listaCorpo;
        int i = 0;
        fprintf(fptr, "  <- ");
        while (auxcorpo != NULL)
        {
            if (i == 0)
            {
                fprintf(fptr, "!%s", auxcorpo->name);
                i = 1;
            }
            else fprintf(fptr, "     !%s", auxcorpo->name);
            if (auxcorpo->prox != NULL) fprintf(fptr, ";\n");
            else fprintf(fptr, ".\n");
            auxcorpo = auxcorpo->prox;
        }
        fprintf(fptr, "\n");
        auxPlanos = auxPlanos->prox;
    }

    fclose(fptr);
}
```

Fonte: Os autores (2022).

3.6. TESTES

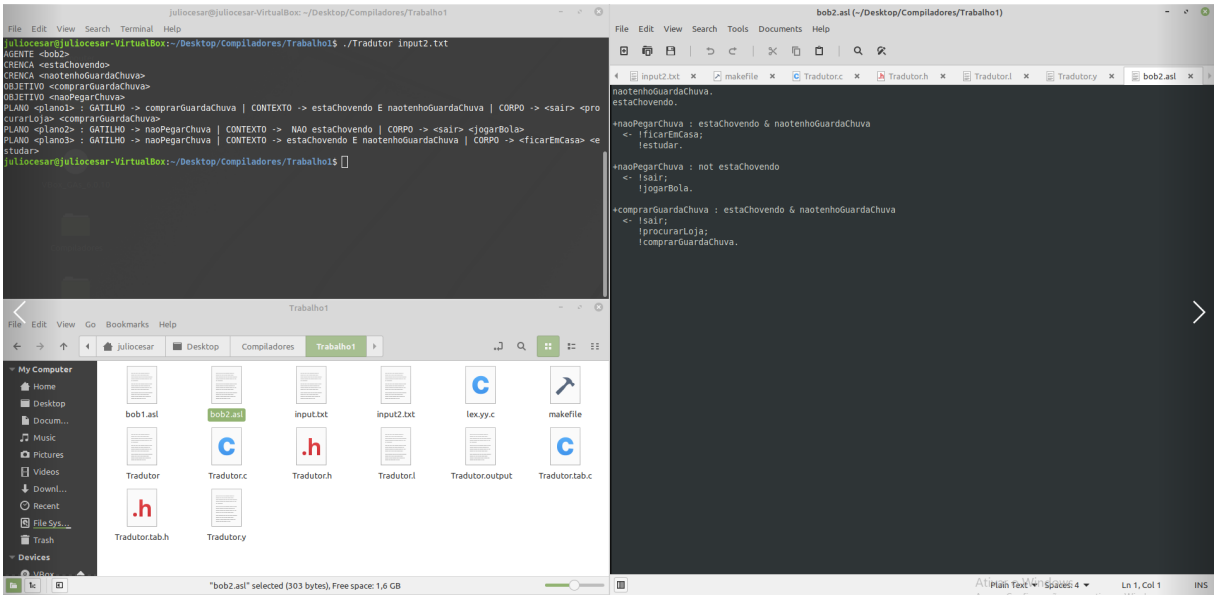
Para a realização dos testes demonstramos o output correto, erros de sintaxe e o output funcionando na IDE jEdit, todos demonstrados nas Figuras abaixo.

Figura 13 - Input com output funcionando



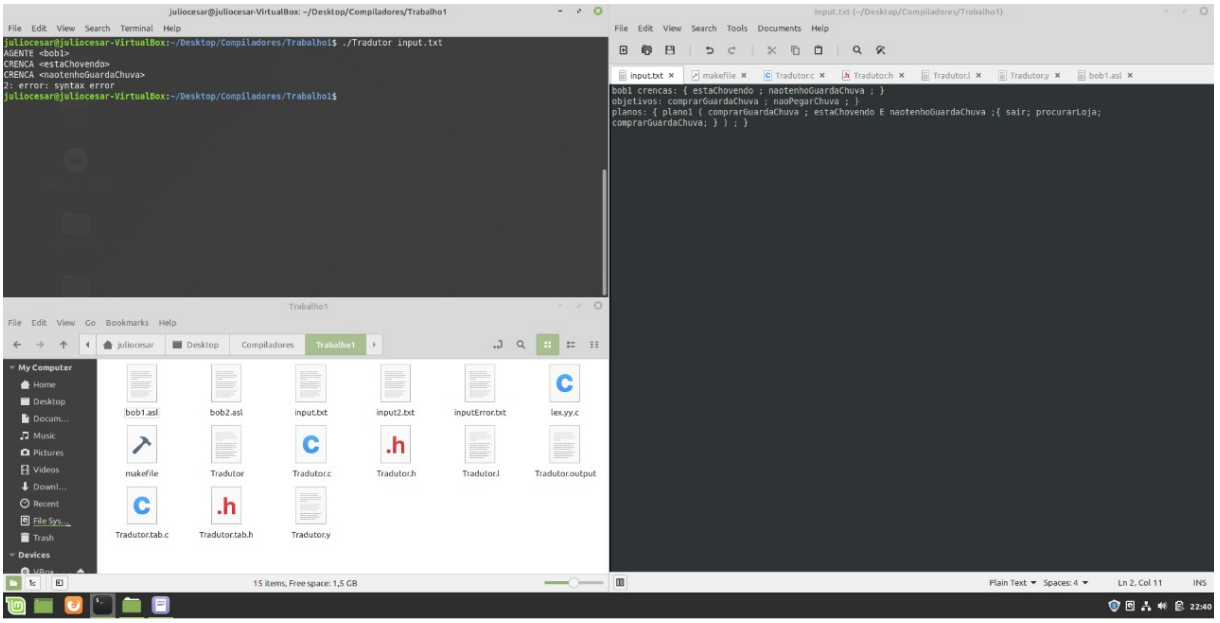
Fonte: Os autores (2022).

Figura 14 - Input alternativo com output funcionando



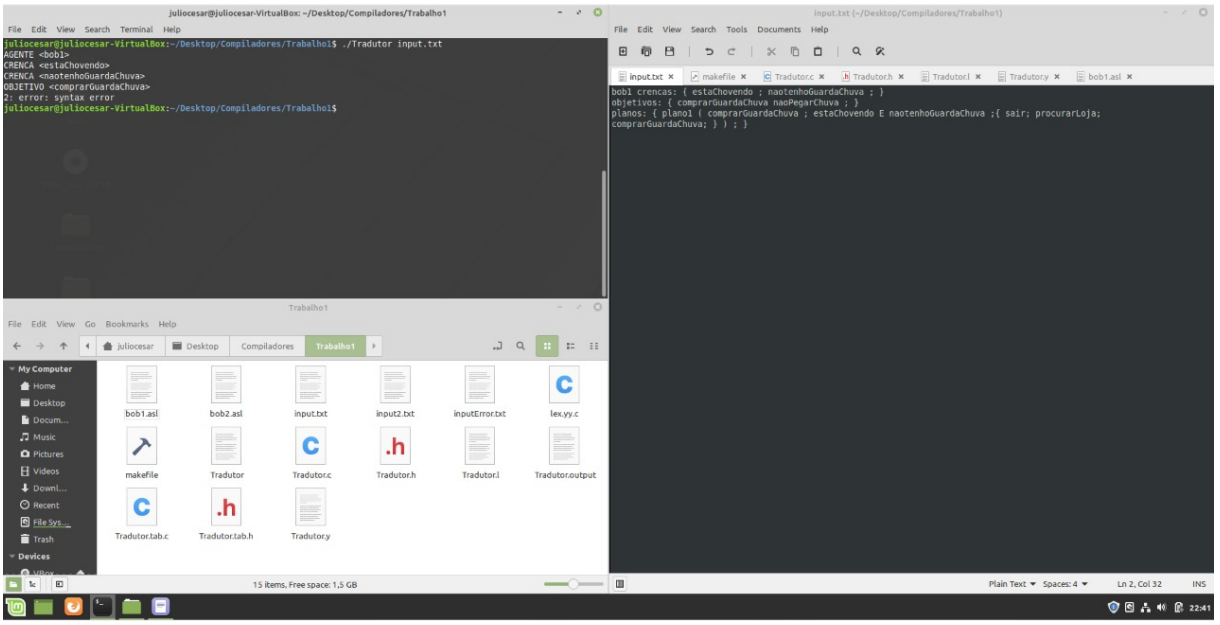
Fonte: Os autores (2022).

Figura 15 - Erro de sintaxe - Falta de Chaves



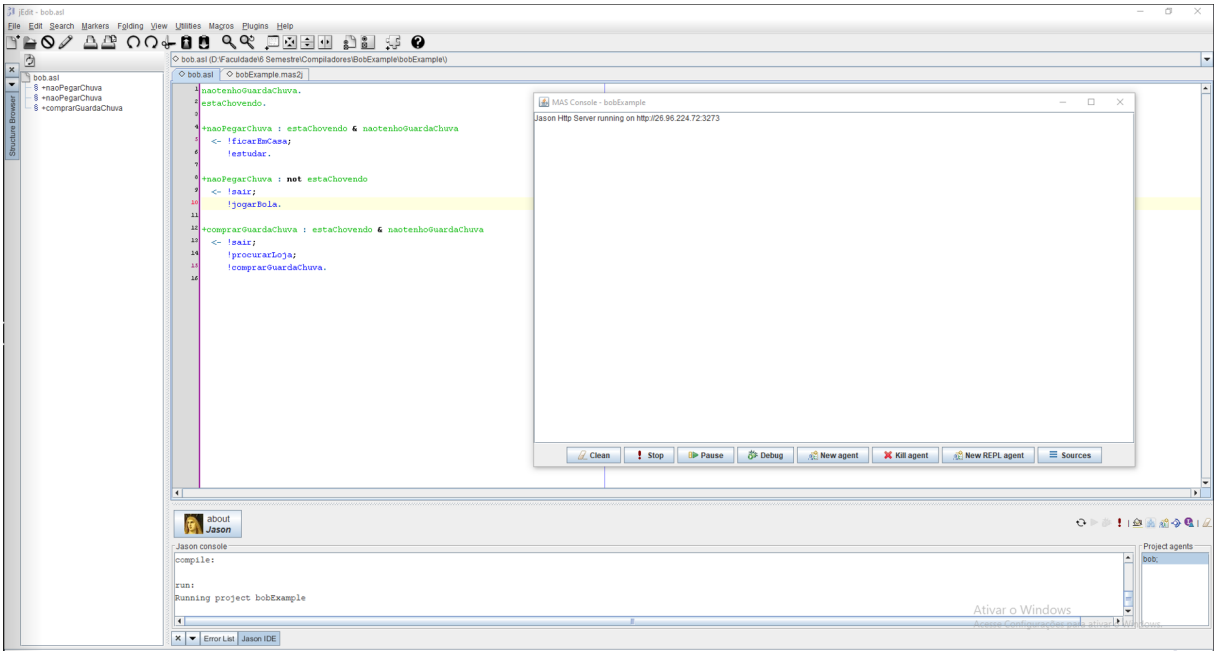
Fonte: Os autores (2022).

Figura 16 - Erro de sintaxe - Falta de Ponto e vírgula



Fonte: Os autores (2022).

Figura 17 - Output compilado e funcionando no jEdit



Fonte: Os autores (2022).

4. REFERÊNCIAS

BORDINI, Rafael H. HUBNER, Jomi F - Jason A Java-based interpreter for an extended version of AgentSpeak, 2007. Disponível em: jason.sourceforge.net/Jason.pdf

JASON DOCUMENTATION, Disponível em: jason.sourceforge.net/doc/f