Conversor de agentes em JSON para MASPY Flex & Bison

Gabriel A. Posonski[1], **Helena Rentschler**[1]

[1]Departamento de Informática (DAINF)
Universidade Tecnológica Federal do Paraná – Ponta Grossa, PR – Brazil

{gabrielalessi,helenarentschler}@alunos.utfpr.edu.br

Abstract. *This article describes the functioning of a converter that transforms a JSON file containing the description of an agent into Python code using the MASPY framework, in order to meet the requirements of the Compilers course in the Computer Science program at UTFPR-PG. For this purpose, the Flex and Bison programs were used, along with auxiliary functions written in the C language.*

Resumo. *Este artigo descreve o funcionamento de um conversor de um arquivo JSON contendo a descrição de um agente para um código em Python utilizando o framework MASPY, a fim de cumprir os requisitos solicitados na disciplina de Compiladores, do curso de Ciência da Computação da UTFPR-PG. Para tal, foi utilizado os programas Flex e Bison, além de funções auxiliares em linguagem C.*

1. Introdução

É possível definir um agente BDI (*Belief–desire–intention*) utilizando o formato JSON (*Javascript Object Notation*), graças ao seu padrão de pares de nome/valor. Juntamente à análise léxica e sintática, pode-se criar um programa que analise um arquivo nesse formato que contenha a descrição de um agente. Nessa abordagem, tal agente será convertido para um código-fonte em Python que utiliza o framework MASPY-ML[Maspy] para o seu funcionamento.

2. Gramática

Primeiramente, deve-se definir a gramática do conversor, que conterá o padrão que deve ser respeitado pelo arquivo JSON de entrada. Para tal, será utilizado o padrão EBNF[CMU]:

2.1. EBNF

```
   agents ::= {'"agentCode"' :  { ( string ':'  agent )+ } }
    agent ::= { name, beliefs, goal, plans }
     name ::= '"name"' ':'  string
  beliefs ::= '"beliefs"' :  [ string ( ',' string )* ]
     goal ::= '"goal"' :  string
    plans ::= '"plans"' :  { plan ( ',' plan )* }
     plan ::= string :  {
            '"trigger"' :  knowledge,
            '"ctx"' :  knowledge,
            '"body"' :  '['
              ( P_ACTION | P_BELIEF | P_GOAL )
              ( ',' ( P_ACTION | P_BELIEF | P_GOAL ) )* ']' '}'
knowledge ::= P_BELIEF | P_GOAL
   string ::= '"' char* '"'
 P_ACTION ::= '"A_' char* '"'
 P_BELIEF ::= '"B_' char* '"'
   P_GOAL ::= '"G_' char* '"'
```

Nota-se que alguns símbolos terminais estão declarados envoltos por aspas duplas ("), uma característica das limitações de *strings* no formato JSON.

2.2. Diagramas de Sintaxe

Para a construção dos diagramas, foi utilizado o site Railroad Diagram [Railroad] .

2.2.1. agents

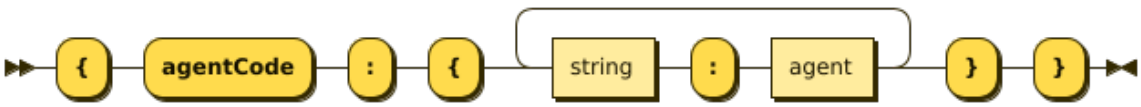

Figure 1. Diagrama agents

2.2.2. agent

Figure 2. Diagrama agent

2.2.3. name

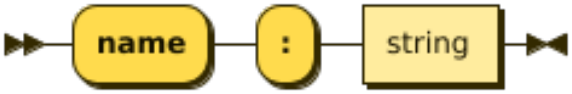

Figure 3. Diagrama name

2.2.4. beliefs

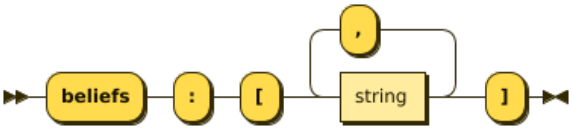

Figure 4. Diagrama beliefs

2.2.5. goal

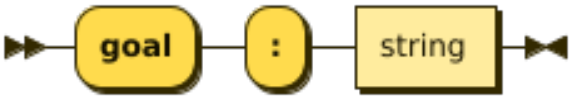

Figure 5. Diagrama goal

2.2.6. plan

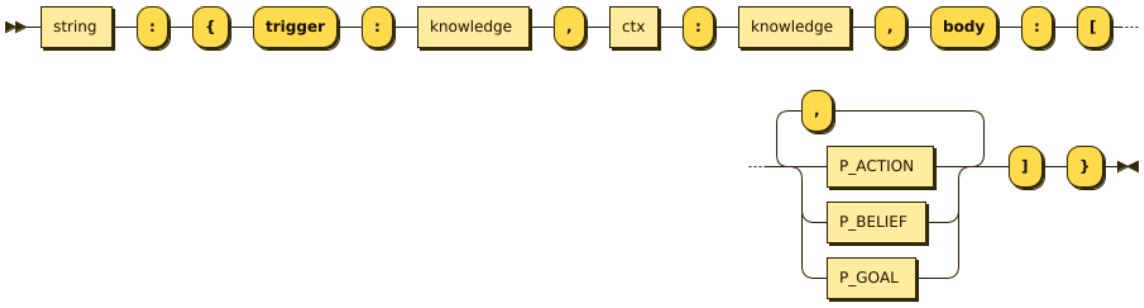

Figure 6. Diagrama plan

2.2.7. knowledge

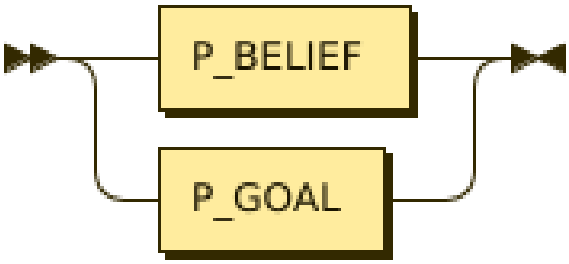

Figure 7. Diagrama knowledge

2.2.8. string

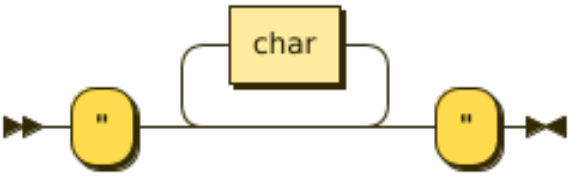

Figure 8. Diagrama string

2.2.9. P_ACTION

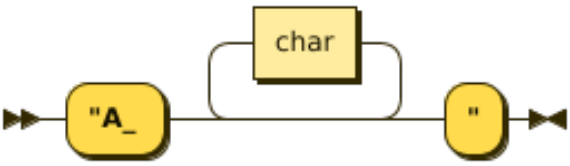

Figure 9. Diagrama P_ACTION

2.2.10. P_BELIEF

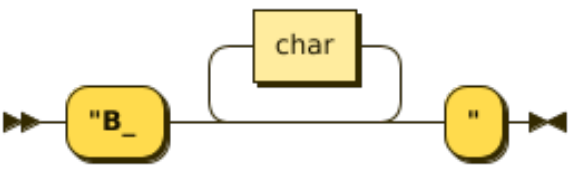

Figure 10. Diagrama P_BELIEF

2.2.11. P_GOAL

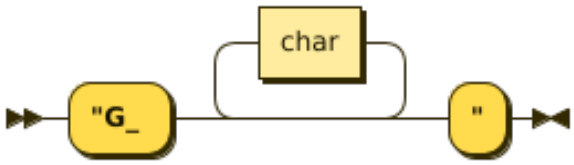

Figure 11. Diagrama P_GOAL

3. Arquivo Lexer

Iniciando na Análise léxica, tem-se o arquivo `lexer.l`:

```
%option noyywrap nodefault yylineno
%{
#include "conversor.h"
#include "parser.tab.h" // Cabeçalho gerado pelo Bison
#include <string.h>     // Para strdup
#include <stdlib.h>
#include <stdio.h>
%}

%%

"\"agentCode\""          { return AGENTCODE; }
"\"beliefs\""            { return BELIEFS; }
"\"goal\""               { return GOAL; }
"\"plans\""              { return PLANS; }
"\"trigger\""            { return TRIGGER; }
"\"ctx\""                { return CTX; }
"\"body\""               { return BODY; }
"\"name\""               { return NAME; }

"{"                      { return '{'; }
"}"                      { return '}'; }
"["                      { return '['; }
"]"                      { return ']'; }
":"                      { return ':'; }
","                      { return ','; }

\"A_([^\"]*)\"     { yylval.s = strndup(yytext + 3, yyleng - 4);
 ↪  printf("Token: P_ACTION (%s)\n", yylval.s); return P_ACTION; }
\"B_([^\"]*)\"     { yylval.s = strndup(yytext + 3, yyleng - 4);
 ↪  printf("Token: P_BELIEF (%s)\n", yylval.s); return P_BELIEF; }
\"G_([^\"]*)\"     { yylval.s = strndup(yytext + 3, yyleng - 4);
 ↪  printf("Token: P_GOAL (%s)\n", yylval.s); return P_GOAL; }
\"[^\"]*\"               { yylval.s = strndup(yytext + 1, yyleng -
 ↪  2); printf("Token: STRING (%s)\n", yylval.s); return STRING; }
```

```
[ \t]+                  ;
\n                      ;
.                       { printf("Unexpected character: %c\n",
 ↪  yytext[0]); return yytext[0]; }


%%

```

No *lexer* estão contidas todas as definições de *tokens* necessárias para a análise do arquivo de entrada. Adicionalmente, tem-se comandos `printf()` ao lado das principais regras, a fim de dar um retorno ao usuário do programa sobre o processamento dos *tokens* durante a execução.

4. Arquivo Parser

No processo de análise sintática, temos as definições do arquivo `parser.y`:

```
%{
    #define YYDEBUG 1
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "conversor.h"

    int yylex(void);
%}

%union {
    Agent *a;
    char *s;
    AgentList *al;
    BeliefList *bl;
    BodyList *b;
    PlanList *pl;
    Plan *p;
    Knowledge *k;
}

%debug

%token <s> STRING P_ACTION P_BELIEF P_GOAL
%token AGENTCODE BELIEFS GOAL PLANS TRIGGER CTX BODY NAME

%type <a> agent
%type <al> agentList
%type <bl> beliefList beliefs
%type <s> goal name
```

```
%type <k> knowledge
%type <p> plan
%type <pl> plans planList
%type <b> bodyList

%start agents

%%

agents:
    '{' AGENTCODE ':' '{' agentList '}' '}' { eval($5); freeAgentList($5); }
    ;

agentList:
    STRING ':' agent { $$ = newAgentList($1, $3, NULL); }
    | STRING ':' agent ',' agentList { $$ = newAgentList($1, $3, $5); }
    ;

agent:
    '{' name ',' beliefs ',' goal ',' plans '}' { $$ = newAgent($2, $4, $6, $8); }
    ;
name:
    NAME ':' STRING { $$ = $3; }
    ;
beliefs:
    BELIEFS ':' '[' beliefList ']' { $$ = $4; }
    ;

beliefList:
    STRING { $$ = newBeliefList($1, NULL); }
    | STRING ',' beliefList { $$ = newBeliefList($1, $3); }
    ;

goal:
    GOAL ':' STRING { $$ = $3; }
    ;

plans:
    PLANS ':' '{' planList '}' { $$ = $4; }
    ;

planList:
    plan { $$ = newPlanList($1, NULL); }
```

```
    | plan ',' planList { $$ = newPlanList($1, $3); }
    ;

plan:
    STRING ':' '{' TRIGGER ':' knowledge ',' CTX ':' knowledge ','
    ↪ BODY ':' '[' bodyList ']' '}' { $$ = newPlan($1, $6, $10,
    ↪ $15); }
    ;
knowledge:
    P_BELIEF    { $$ = newKnowledge($1, 'B'); }
    | P_GOAL    { $$ = newKnowledge($1, 'G'); }
    ;
bodyList:
    P_ACTION { $$ = newBodyList($1, 'A', NULL); }
    | P_BELIEF { $$ = newBodyList($1, 'B', NULL); }
    | P_GOAL { $$ = newBodyList($1, 'G', NULL); }
    | P_ACTION ',' bodyList { $$ = newBodyList($1, 'A', $3); }
    | P_BELIEF ',' bodyList { $$ = newBodyList($1, 'B', $3); }
    | P_GOAL ',' bodyList { $$ = newBodyList($1, 'G', $3); }
    ;

%%

```

O parser está definido de maneira a processar o `JSON` da maneira correta, construindo uma estrutura de dados usando conceitos de listas e árvores. Ao término da análise sintática, temos um nó raiz do tipo `AgentList` que conterá todos os agentes definidos no `JSON`.

5. Funções Auxiliares

Nesta seção, iremos analisar as funções auxiliares que, juntamente ao *parser*, possibilita a criação da estrutura do agente. Primeiramente, a declaração do programa no arquivo `conversor.h`:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

extern int yylineno;
extern int yydebug;
void yyerror(char *s, ...);
extern FILE *yyin;
typedef struct Plan Plan;
typedef struct Agent Agent;
typedef struct BodyList BodyList;
typedef struct PlanList PlanList;
```

```
14 typedef struct Knowledge Knowledge;
15 typedef struct AgentList AgentList;
16 typedef struct BeliefList BeliefList;
17
18 struct BodyList
19 {
20     char *body;
21     char type;
22     struct BodyList *next;
23 };
24
25 struct BeliefList
26 {
27     char *b;
28     struct BeliefList *next;
29 };
30
31 struct Plan
32 {
33     char *nome;
34     Knowledge *trigger;
35     Knowledge *ctx;
36     BodyList *body;
37 };
38
39 struct PlanList
40 {
41     Plan *p;
42     PlanList *next;
43 };
44
45 struct Agent
46 {
47     char *nome;
48     BeliefList *beliefs;
49     char *goal;
50     PlanList *plans;
51 };
52
53 struct AgentList
54 {
55     char *id;
56     Agent *a;
57     AgentList *next;
58 };
59
```

```
60 struct Knowledge
61 {
62     char *knowledge;
63     int type; //'B' = belief, 'G' = goal
64 };
65 
66 void eval(AgentList *al);
67 void freePlan(Plan *plan);
68 void freeAgent(Agent *agent);
69 void freeBodyList(BodyList *body);
70 void freePlanList(PlanList *plans);
71 void freeAgentList(AgentList *agents);
72 void freeBeliefList(BeliefList *beliefs);
73 Knowledge *newKnowledge(char *name, int type);
74 void copyString(char **destino, char *conteudo);
75 PlanList *newPlanList(Plan *plan, PlanList *next);
76 BeliefList *newBeliefList(char *belief, BeliefList *next);
77 AgentList *newAgentList(char *id, Agent *a, AgentList *next);
78 BodyList *newBodyList(char *body, char type, BodyList *next);
79 Agent *newAgent(char *nome, BeliefList *beliefs, char *goal,
   ↪ PlanList *plans);
80 Plan *newPlan(char *nome, Knowledge *trigger, Knowledge *ctx,
   ↪ BodyList *body);
```

Com isso, tem-se o arquivo com as definições das funções e a função `main()`, `conversor.c`:

```
1  #include "conversor.h"
2  
3  void copyString(char **destino, char *conteudo)
4  {
5      *destino = malloc(strlen(conteudo) + 1);
6      if (!*destino)
7      {
8          yyerror("Sem espaco para string");
9          exit(0);
10     }
11     strcpy(*destino, conteudo);
12 }
13 BodyList *newBodyList(char *body, char type, BodyList *next)
14 {
15     BodyList *b = malloc(sizeof(BodyList));
16     if (!b)
17     {
18         yyerror("Sem espaco");
19         exit(0);
20     }
21     copyString(&b->body, body);
```

```
22      b->next = next;
23      b->type = type;
24      return b;
25  }
26
27  Plan *newPlan(char *nome, Knowledge *trigger, Knowledge *ctx,
    ↪   BodyList *body)
28  {
29      Plan *p = malloc(sizeof(Plan));
30      if (!p)
31      {
32          yyerror("Sem espaco");
33          exit(0);
34      }
35      copyString(&p->nome, nome);
36
37      p->trigger = malloc(sizeof(Knowledge));
38      if (!p->trigger)
39      {
40          yyerror("Sem espaco para trigger");
41          exit(0);
42      }
43      copyString(&p->trigger->knowledge, trigger->knowledge);
44      p->trigger->type = trigger->type;
45
46      p->ctx = malloc(sizeof(Knowledge));
47      if (!p->ctx)
48      {
49          yyerror("Sem espaco para contexto");
50          exit(0);
51      }
52      copyString(&p->ctx->knowledge, ctx->knowledge);
53      p->ctx->type = ctx->type;
54
55      p->body = body;
56      return p;
57  }
58
59  PlanList *newPlanList(Plan *p, PlanList *next)
60  {
61      PlanList *pl = malloc(sizeof(PlanList));
62      if (!pl)
63      {
64          yyerror("Sem espaco");
65          exit(0);
66      }
```

```
67      pl->p = p;
68      pl->next = next;
69      return pl;
70  }
71
72  BeliefList *newBeliefList(char *belief, BeliefList *next)
73  {
74      BeliefList *b = malloc(sizeof(BeliefList));
75      if (!b)
76      {
77          yyerror("Sem espaco");
78          exit(0);
79      }
80      copyString(&b->b, belief);
81      b->next = next;
82      return b;
83  }
84
85  Agent *newAgent(char *nome, BeliefList *beliefs, char *goal,
    ↪  PlanList *plans)
86  {
87      Agent *a = malloc(sizeof(Agent));
88      if (!a)
89      {
90          yyerror("Sem espaco");
91          exit(0);
92      }
93      copyString(&a->nome, nome);
94      a->beliefs = beliefs;
95      copyString(&a->goal, goal);
96      a->plans = plans;
97      return a;
98  }
99
100 AgentList *newAgentList(char *id, Agent *a, AgentList *next)
101 {
102     AgentList *al = malloc(sizeof(AgentList));
103     if (!al)
104     {
105         yyerror("Sem espaco");
106         exit(0);
107     }
108     al->id = strdup(id);
109     al->a = a;
110     al->next = next;
111     return al;
```

```
112 }
113 Knowledge *newKnowledge(char *name, int type)
114 {
115     Knowledge *k = malloc(sizeof(Knowledge));
116     if (!k)
117     {
118         yyerror("Sem espaco");
119         exit(0);
120     }
121     copyString(&k->knowledge, name);
122     k->type = type;
123 
124     return k;
125 }
126 
127 void eval(AgentList *al)
128 {
129 
130     // Abrindo o arquivo de saída
131     FILE *file = fopen("agente.py", "w");
132     if (!file)
133     {
134         fprintf(stderr, "Erro ao criar o arquivo de saída.\n");
135         return;
136     }
137 
138     // Adiciona o cabeçalho do MASPY
139     fprintf(file, "from maspy import *\n\n");
140 
141     // Iterar sobre a lista de agentes
142     AgentList *currentAgent = al;
143     while (currentAgent)
144     {
145         Agent *a = currentAgent->a;
146 
147         // Classe do agente
148         fprintf(file, "class %s(Agent):\n", currentAgent->id);
149         fprintf(file, "    def __init__(self, agt_name):\n");
150         fprintf(file, "        super().__init__(agt_name)\n");
151         // Adiciona as crenças
152         BeliefList *currentBelief = a->beliefs;
153         while (currentBelief)
154         {
155             fprintf(file, "        self.add(Belief(\"%s\"))\n",
                ↪  currentBelief->b);
156             currentBelief = currentBelief->next;
```

```
157         }
158
159         // Adiciona as metas
160         if (a->goal && a->goal[0] != '\0')
161             fprintf(file, "        self.add(Goal(\"%s\"))\n",
                ↪  a->goal);
162
163         // Adiciona os planos
164         PlanList *currentPlan = a->plans;
165         while (currentPlan)
166         {
167             Plan *p = currentPlan->p;
168             char trigger[10];
169             char ctx[10];
170             if (p->trigger->type == 'B')
171             {
172                 strcpy(trigger, "Belief");
173             }
174             else
175             {
176                 strcpy(trigger, "Goal");
177             }
178             if (p->ctx->type == 'B')
179             {
180                 strcpy(ctx, "Belief");
181             }
182             else
183             {
184                 strcpy(ctx, "Goal");
185             }
186             fprintf(file, "\n    @pl(gain,%s(\"%s\"),
                ↪  %s(\"%s\"))\n", trigger, p->trigger->knowledge,
                ↪  ctx, p->ctx->knowledge);
187             fprintf(file, "    def %s_(self, src):\n",
                ↪  p->trigger->knowledge);
188             BodyList *currentBody = p->body;
189
190             while (currentBody)
191             {
192                 if (currentBody->type == 'A')
193                     fprintf(file, "        self.print(\"%s\")\n",
                        ↪  currentBody->body);
194                 else if (currentBody->type == 'B')
195                     fprintf(file, "
                        ↪  self.add(Belief(\"%s\"))\n",
                        ↪  currentBody->body);
```

```
196                 else if (currentBody->type == 'G')
197                     fprintf(file, "
                        ↪  self.add(Goal(\"%s\"))\n",
                        ↪  currentBody->body);
198                 else
199                 {
200                     fprintf(stderr, "Erro: tipo de body
                        ↪  incompativel.\n");
201                     fclose(file);
202                 }
203                 currentBody = currentBody->next;
204             }
205             currentPlan = currentPlan->next;
206         }
207 
208         // Linha em branco entre classes de agentes
209         fprintf(file, "\n");
210         currentAgent = currentAgent->next;
211     }
212 
213     // Adiciona código para iniciar o sistema
214     currentAgent = al;
215     while (currentAgent)
216     {
217         Agent *a = currentAgent->a;
218         fprintf(file, "%s(\"%s\")\n", currentAgent->id, a->nome);
219         currentAgent = currentAgent->next;
220     }
221     fprintf(file, "Admin().start_system()\n");
222 
223     fclose(file);
224     printf("Arquivo Python compatível com MASPY gerado com sucesso:
        ↪  output_maspy.py\n");
225 }
226 
227 void freeBodyList(BodyList *body)
228 {
229     while (body)
230     {
231         BodyList *temp = body;
232         body = body->next;
233         free(temp->body);
234         free(temp);
235     }
236 }
237 
```

```
238 void freeBeliefList(BeliefList *beliefs)
239 {
240     while (beliefs)
241     {
242         BeliefList *temp = beliefs;
243         beliefs = beliefs->next;
244         free(temp->b);
245         free(temp);
246     }
247 }
248
249 void freePlan(Plan *plan)
250 {
251     if (plan)
252     {
253         free(plan->nome);
254         free(plan->trigger);
255         free(plan->ctx);
256         freeBodyList(plan->body);
257         free(plan);
258     }
259 }
260
261 void freePlanList(PlanList *plans)
262 {
263     while (plans)
264     {
265         PlanList *temp = plans;
266         plans = plans->next;
267         freePlan(temp->p);
268         free(temp);
269     }
270 }
271
272 void freeAgent(Agent *agent)
273 {
274     if (agent)
275     {
276         free(agent->nome);
277         freeBeliefList(agent->beliefs);
278         free(agent->goal);
279         freePlanList(agent->plans);
280         free(agent);
281     }
282 }
283
```

```c
void freeAgentList(AgentList *agents)
{
    while (agents)
    {
        AgentList *temp = agents;
        agents = agents->next;
        free(temp->id);
        freeAgent(temp->a);
        free(temp);
    }
}

// Declaração de funções do Bison e Flex
extern int yyparse();
extern void yyrestart(FILE *input_file);
extern FILE *yyin;

void yyerror(char *s, ...)
{
    va_list ap;
    va_start(ap, s);

    fprintf(stderr, "Erro na linha %d: ", yylineno);
    vfprintf(stderr, s, ap);
    fprintf(stderr, "\n");
}

int main(int argc, char *argv[])
{
    // Verifica se o arquivo de entrada foi passado como argumento
    if (argc != 2)
    {
        fprintf(stderr, "Uso: %s <arquivo.json>\n", argv[0]);
        return 1;
    }

    // Abre o arquivo JSON para leitura
    FILE *input_file = fopen(argv[1], "r");
    if (!input_file)
    {
        fprintf(stderr, "Erro ao abrir o arquivo %s\n", argv[1]);
        return 1;
    }

    // Reinicia o lexer para processar o novo arquivo
    yyrestart(input_file);
```

```
330     yydebug = 0;
331
332     // Chama o parser
333     printf("Parsing do arquivo JSON em andamento...\n");
334     if (yyparse() == 0)
335     {
336         printf("Parsing concluído com sucesso.\n");
337     }
338     else
339     {
340         fprintf(stderr, "Erro durante o parsing do arquivo
            ↪  JSON.\n");
341         fclose(input_file);
342         return 1;
343     }
344
345     // Fecha o arquivo de entrada
346     fclose(input_file);
347
348     // Chama a função eval para processar os dados e gerar o
        ↪  código Python
349     printf("Gerando código Python a partir dos dados...\n");
350
351     printf("Processo concluído. Verifique o arquivo de saída.\n");
352     return 0;
353 }
354
```

Como citado anteriormente, o conversor trabalha construindo uma estrutura de dados em árvores e listas, adequando as definições do agente de maneira congruente com auxílio do parser, que chama as funções auxiliares criando nós com o TAD correspondente.

Ao atingir o nó raiz, a função `eval()` é chamada, realizando o processamento do agente. Primeiro, cria-se o arquivo de saída, o qual conterá o código-fonte gerado pelo agente, insere-se o comando de importação do framework `MASPY` e então inicia-se o processamento. De maneira bem semelhante a um algoritmo de percorrer listas, o agente é lido nó-a-nó e suas definições são inseridas no código-fonte através da função `fprintf()` do C.

Finalmente, os agentes são instanciados e o comando de inicialização do MAS é chamado e o conversor exibe uma mensagem de sucesso.

6. Testes

Foram feitos três testes efetivos e um para demonstrar a detecção do erro no arquivo de entrada:

6.1. Teste 1

```
{
    "agentCode": {
        "ag1": {
            "name": "bob",
            "beliefs": [
                "estaChovendo",
                "naotenhoGuardaChuva",
                "tenhoProva"
            ],
            "goal": "comprarGuardaChuva",
            "plans": {
                "p1": {
                    "trigger": "G_comprarGuardaChuva",
                    "ctx": "B_estaChovendo",
                    "body": [
                        "A_sair para comprar guardachuva",
                        "A_procurar loja",
                        "A_comprar guardachuva",
                        "G_estudar"
                    ]
                },
                "p2": {
                    "trigger": "G_estudar",
                    "ctx": "B_tenhoProva",
                    "body": [
                        "A_convidar alice",
                        "A_estudar na biblioteca"
                    ]
                }
            }
        },
        "ag2": {
            "name": "alice",
            "beliefs": [
                "estaChovendo",
                "tenhoGuardaChuva",
                "tenhoProva"
            ],
            "goal": "",
            "plans": {
                "p1": {
                    "trigger": "B_tenhoGuardaChuva",
                    "ctx": "B_estaChovendo",
                    "body": [
                        "A_posso estudar",
```

```
                        "A_recebi convite de bob",
                        "G_estudarNaBiblioteca"
                    ]
                },
                "p2": {
                    "trigger": "G_estudarNaBiblioteca",
                    "ctx": "B_tenhoProva",
                    "body": [
                        "A_escolher caminho",
                        "A_usar guardachuva",
                        "A_ir para biblioteca",
                        "A_estudar!"
                    ]
                }
            }
        }
    }
}
```

Este exemplo é fornecido pelo professor na descrição do projeto, e gerá a seguinte saída:

```
from maspy import *

class ag1(Agent):
    def __init__(self, agt_name):
        super().__init__(agt_name)
        self.add(Belief("estaChovendo"))
        self.add(Belief("naotenhoGuardaChuva"))
        self.add(Belief("tenhoProva"))
        self.add(Goal("comprarGuardaChuva"))

    @pl(gain,Goal("comprarGuardaChuva"), Belief("estaChovendo"))
    def comprarGuardaChuva_(self, src):
        self.print("sair para comprar guardachuva")
        self.print("procurar loja")
        self.print("comprar guardachuva")
        self.add(Goal("estudar"))

    @pl(gain,Goal("estudar"), Belief("tenhoProva"))
    def estudar_(self, src):
        self.print("convidar alice")
        self.print("estudar na biblioteca")

class ag2(Agent):
    def __init__(self, agt_name):
        super().__init__(agt_name)
```

```
        self.add(Belief("estaChovendo"))
        self.add(Belief("tenhoGuardaChuva"))
        self.add(Belief("tenhoProva"))

    @pl(gain,Belief("tenhoGuardaChuva"), Belief("estaChovendo"))
    def tenhoGuardaChuva_(self, src):
        self.print("posso estudar")
        self.print("recebi convite de bob")
        self.add(Goal("estudarNaBiblioteca"))

    @pl(gain,Goal("estudarNaBiblioteca"), Belief("tenhoProva"))
    def estudarNaBiblioteca_(self, src):
        self.print("escolher caminho")
        self.print("usar guardachuva")
        self.print("ir para biblioteca")
        self.print("estudar!")

ag1("bob")
ag2("alice")
Admin().start_system()

```

6.2. Teste 2

Similar ao anterior, porém agora somente com um agente:

```
{
    "agentCode": {
        "ag1": {
            "name": "carla",
            "beliefs": [
                "estaEnsolarado",
                "tenhoOculosEscuros",
                "precisoFazerCompras"
            ],
            "goal": "comprarMantimentos",
            "plans": {
                "p1": {
                    "trigger": "G_comprarMantimentos",
                    "ctx": "B_precisoFazerCompras",
                    "body": [
                        "A_fazer lista de compras",
                        "A_ir ao supermercado",
                        "A_comprar mantimentos",
                        "B_comprasFeitas",
                        "G_voltarParaCasa"
                    ]
                },
```

```
                "p2": {
                    "trigger": "G_voltarParaCasa",
                    "ctx": "B_comprasFeitas",
                    "body": [
                        "A_pegar transporte",
                        "A_guardar mantimentos"
                    ]
                }
            }
        }
    }
}

```

Saída:

```
from maspy import *

class ag1(Agent):
    def __init__(self, agt_name):
        super().__init__(agt_name)
        self.add(Belief("estaEnsolarado"))
        self.add(Belief("tenhoOculosEscuros"))
        self.add(Belief("precisoFazerCompras"))
        self.add(Goal("comprarMantimentos"))

    @pl(gain,Goal("comprarMantimentos"),
     ↪  Belief("precisoFazerCompras"))
    def comprarMantimentos_(self, src):
        self.print("fazer lista de compras")
        self.print("ir ao supermercado")
        self.print("comprar mantimentos")
        self.add(Belief("comprasFeitas"))
        self.add(Goal("voltarParaCasa"))

    @pl(gain,Goal("voltarParaCasa"), Belief("comprasFeitas"))
    def voltarParaCasa_(self, src):
        self.print("pegar transporte")
        self.print("guardar mantimentos")

ag1("carla")
Admin().start_system()

```

6.3. Teste 3

Explorando um pouco mais o poder de MAS's, um teste é realizado com cinco agentes:

```
{
  "agentCode": {
      "ag1": {
          "name": "bob",
          "beliefs": [
              "estaChovendo",
              "naotenhoGuardaChuva",
              "tenhoProva"
          ],
          "goal": "comprarGuardaChuva",
          "plans": {
              "p1": {
                  "trigger": "G_comprarGuardaChuva",
                  "ctx": "B_estaChovendo",
                  "body": [
                      "A_sair para comprar guardachuva",
                      "A_procurar loja",
                      "A_comprar guardachuva",
                      "G_estudar"
                  ]
              },
              "p2": {
                  "trigger": "G_estudar",
                  "ctx": "B_tenhoProva",
                  "body": [
                      "A_estudar sozinho"
                  ]
              }
          }
      },
      "ag2": {
          "name": "alice",
          "beliefs": [
              "estaChovendo",
              "tenhoGuardaChuva",
              "tenhoProva"
          ],
          "goal": "",
          "plans": {
              "p1": {
                  "trigger": "B_tenhoGuardaChuva",
                  "ctx": "B_estaChovendo",
                  "body": [
                      "A_preparar materiais de estudo",
                      "G_estudarNaBiblioteca"
                  ]
```

```
            },
            "p2": {
                "trigger": "G_estudarNaBiblioteca",
                "ctx": "B_tenhoProva",
                "body": [
                    "A_escolher caminho",
                    "A_usar guardachuva",
                    "A_ir para biblioteca",
                    "A_estudar!"
                ]
            }
        }
    },
    "ag3": {
        "name": "carlos",
        "beliefs": [
            "estaFrio",
            "tenhoCasaco",
            "precisoFazerCompras"
        ],
        "goal": "irAoMercado",
        "plans": {
            "p1": {
                "trigger": "G_irAoMercado",
                "ctx": "B_precisoFazerCompras",
                "body": [
                    "A_colocar casaco",
                    "A_sair para o mercado",
                    "A_comprar mantimentos"
                ]
            }
        }
    },
    "ag4": {
        "name": "diana",
        "beliefs": [
            "tenhoTrabalho",
            "gostoDeCafe",
            "cafeteriaAberta"
        ],
        "goal": "tomarCafe",
        "plans": {
            "p1": {
                "trigger": "G_tomarCafe",
                "ctx": "B_cafeteriaAberta",
                "body": [
```

```
                        "A_pegar carteira",
                        "A_ir para cafeteria",
                        "A_comprar cafe",
                        "A_relaxar antes do trabalho"
                    ]
                }
            }
        },
        "ag5": {
            "name": "eduardo",
            "beliefs": [
                "tenhoSono",
                "precisoEstudar",
                "tenhoCafe"
            ],
            "goal": "acordar",
            "plans": {
                "p1": {
                    "trigger": "G_acordar",
                    "ctx": "B_tenhoSono",
                    "body": [
                        "A_beber cafe",
                        "A_lavar rosto",
                        "A_estudar"
                    ]
                }
            }
        }
    }
}
```

Saída:

```
from maspy import *

class ag1(Agent):
    def __init__(self, agt_name):
        super().__init__(agt_name)
        self.add(Belief("estaChovendo"))
        self.add(Belief("naotenhoGuardaChuva"))
        self.add(Belief("tenhoProva"))
        self.add(Goal("comprarGuardaChuva"))

    @pl(gain,Goal("comprarGuardaChuva"), Belief("estaChovendo"))
    def comprarGuardaChuva_(self, src):
        self.print("sair para comprar guardachuva")
        self.print("procurar loja")
        self.print("comprar guardachuva")
```

```
16          self.add(Goal("estudar"))
17
18      @pl(gain,Goal("estudar"), Belief("tenhoProva"))
19      def estudar_(self, src):
20          self.print("estudar sozinho")
21
22  class ag2(Agent):
23      def __init__(self, agt_name):
24          super().__init__(agt_name)
25          self.add(Belief("estaChovendo"))
26          self.add(Belief("tenhoGuardaChuva"))
27          self.add(Belief("tenhoProva"))
28
29      @pl(gain,Belief("tenhoGuardaChuva"), Belief("estaChovendo"))
30      def tenhoGuardaChuva_(self, src):
31          self.print("preparar materiais de estudo")
32          self.add(Goal("estudarNaBiblioteca"))
33
34      @pl(gain,Goal("estudarNaBiblioteca"), Belief("tenhoProva"))
35      def estudarNaBiblioteca_(self, src):
36          self.print("escolher caminho")
37          self.print("usar guardachuva")
38          self.print("ir para biblioteca")
39          self.print("estudar!")
40
41  class ag3(Agent):
42      def __init__(self, agt_name):
43          super().__init__(agt_name)
44          self.add(Belief("estaFrio"))
45          self.add(Belief("tenhoCasaco"))
46          self.add(Belief("precisoFazerCompras"))
47          self.add(Goal("irAoMercado"))
48
49      @pl(gain,Goal("irAoMercado"), Belief("precisoFazerCompras"))
50      def irAoMercado_(self, src):
51          self.print("colocar casaco")
52          self.print("sair para o mercado")
53          self.print("comprar mantimentos")
54
55  class ag4(Agent):
56      def __init__(self, agt_name):
57          super().__init__(agt_name)
58          self.add(Belief("tenhoTrabalho"))
59          self.add(Belief("gostoDeCafe"))
60          self.add(Belief("cafeteriaAberta"))
61          self.add(Goal("tomarCafe"))
```

```
62
63      @pl(gain,Goal("tomarCafe"), Belief("cafeteriaAberta"))
64      def tomarCafe_(self, src):
65          self.print("pegar carteira")
66          self.print("ir para cafeteria")
67          self.print("comprar cafe")
68          self.print("relaxar antes do trabalho")
69
70  class ag5(Agent):
71      def __init__(self, agt_name):
72          super().__init__(agt_name)
73          self.add(Belief("tenhoSono"))
74          self.add(Belief("precisoEstudar"))
75          self.add(Belief("tenhoCafe"))
76          self.add(Goal("acordar"))
77
78      @pl(gain,Goal("acordar"), Belief("tenhoSono"))
79      def acordar_(self, src):
80          self.print("beber cafe")
81          self.print("lavar rosto")
82          self.print("estudar")
83
84  ag1("bob")
85  ag2("alice")
86  ag3("carlos")
87  ag4("diana")
88  ag5("eduardo")
89  Admin().start_system()
90
```

Para fins demonstrativos, eis a saída do último teste ao executar o código Python:

```
1  # Admin #> Starting MASPY Program
2  # Admin #> Starting Agents
3  Agent:bob> sair para comprar guardachuva
4  Agent:bob> procurar loja
5  Agent:bob> comprar guardachuva
6  Agent:alice> preparar materiais de estudo
7  Agent:bob> estudar sozinho
8  Agent:diana> pegar carteira
9  Agent:diana> ir para cafeteria
10 Agent:carlos> colocar casaco
11 Agent:eduardo> beber cafe
12 Agent:carlos> sair para o mercado
13 Agent:eduardo> lavar rosto
14 Agent:alice> escolher caminho
15 Agent:diana> comprar cafe
16 Agent:carlos> comprar mantimentos
```

```
Agent:diana> relaxar antes do trabalho
Agent:alice> usar guardachuva
Agent:alice> ir para biblioteca
Agent:eduardo> estudar
Agent:alice> estudar!
^C# Admin #> [Closing System]
# Admin #> Still running agent(s):
bob | alice | carlos | diana | eduardo |
# Admin #> Ending MASPY Program
```

6.4. Teste com falha

Para exemplificar um erro, foi removida a definição de `goal` do agente do Teste 2:

```
...
        "ag1": {
            "name": "carla",
            "beliefs": [
                "estaEnsolarado",
                "tenhoOculosEscuros",
                "precisoFazerCompras"
            ],
            "plans": {
                "p1": {
...
```

A mensagem exibida pelo programa será:

```
gab@Vivobook:~/dev/Faculdade/2024_2/Compiladores/T2$ ./conversor in4.json
Parsing do arquivo JSON em andamento...
Token: STRING (ag1)
Token: STRING (carla)
Token: STRING (estaEnsolarado)
Token: STRING (tenhoOculosEscuros)
Token: STRING (precisoFazerCompras)
Erro na linha 10: syntax error
Erro durante o parsing do arquivo JSON.
```

Figure 12. Erro exibido pelo conversor

7. Conclusão

Com o desenvolvimento desse trabalho, foi possível aprimorar o entendimento dos processos de compilação, bem como aperfeiçoar o conhecimento de sistemas multi-agentes. Certamente ferramentas como Flex e Bison são potentes aliadas no processo de análise léxica e sintática na Computação.

References

CMU. EBNF: A Notation to Describe Syntax, Carnegie Mellon School of Computer Science. `https://www.cs.cmu.edu/~pattis/misc/ebnf2.pdf`. Acesso em: 07/02/2025.

Maspy. A Python Framework for Multi-Agent Programming with Machine Learning. https://github.com/laca-is/MASPY. Acesso em: 07/02/2025.

Railroad. Railroad Diagram Generator. https://rr.red-dove.com/ui. Acesso em: 07/02/2025.