



UNIVERSIDADE DO MINHO  
Departamento de Informática

PROCESSAMENTO DE LINGUAGENS

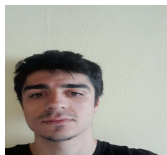
## Projeto Final

### Conversor Toml-JSON

Grupo 30  
Grupo "Casa do Povo"

#### Feito por:

Dinis Gonçalves Estrada (A97503)  
Emanuel Lopes Monteiro da Silva (A95114)



A97393



A95114

May 28, 2023  
Ano Letivo 2022/23

# 1 Resumo

Este trabalho realizado no âmbito da unidade curricular de Processamento de Linguagens consistiu na criação de um conversor de Toml para Json. Ao longo deste relatório está explicado detalhadamente todos os passos da resolução deste projeto, nomeadamente as escolhas feitas pelo grupo e o raciocínio na base das soluções propostas.

É importante referir que o desenvolvimento do projeto foi faseado, na medida em que, inicialmente começou-se por desenvolver uma gramática independente de contexto e, só depois o analisador léxico. Por fim, o último passo consistiu no desenvolvimento da gramática tradutora com o objetivo de criar um conversor entre estas duas linguagens.

# Conteúdo

<b>1</b>	<b>Resumo</b>	<b>1</b>
<b>2</b>	<b>Introdução</b>	<b>4</b>
2.1	Enquadramento e Contexto . . . . .	4
2.2	Problema e Objetivo . . . . .	4
2.3	Estrutura do Relatório . . . . .	4
<b>3</b>	<b>Análise e Especificação</b>	<b>5</b>
3.1	Descrição informal do problema . . . . .	5
3.2	Especificação dos requisitos . . . . .	5
<b>4</b>	<b>Conceção/Desenho da Resolução</b>	<b>6</b>
4.1	Gramática Independente de Contexto (GIC) . . . . .	6
4.2	Implementação . . . . .	8
4.2.1	Analisador Léxico . . . . .	8
4.2.2	Gramática Tradutora (GT) . . . . .	11
4.2.3	Compilador Yacc . . . . .	13
4.2.4	Compilação e Execução . . . . .	17
<b>5</b>	<b>Codificação e Testes</b>	<b>19</b>
5.1	Testes realizados e Resultados . . . . .	19
5.2	Problemas e Decisões . . . . .	22
<b>6</b>	<b>Conclusão</b>	<b>23</b>
<b>7</b>	<b>Apêndice</b>	<b>24</b>

## Lista de Figuras

1	Regra de produção referente a <b>Toml</b> . . . . .	6
2	Regra de produção referente às <b>Tables</b> . . . . .	6
3	Regra de produção referente a <b>Toml</b> . . . . .	6
4	Regra de produção referente a <b>Toml</b> . . . . .	6
5	Regra de produção referente a <b>Toml</b> . . . . .	6
6	Regra de produção referente a <b>Toml</b> . . . . .	7
7	Regra de produção referente aos dicionários . . . . .	7
8	Regra de produção referente aos possíveis valores . . . . .	7
9	Menu . . . . .	17
10	Lexer . . . . .	17
11	Conversor . . . . .	17
12	Interface . . . . .	18

## 2 Introdução

### 2.1 Enquadramento e Contexto

No âmbito da unidade curricular de Processamento de Linguagens, foi-nos proposto um projeto no qual teríamos de desenvolver um conversor de *toml* para *json*. Para desenvolver este conversor teríamos de construir uma ferramenta (Flex,Yacc) que possa fazer ligação entre estas duas linguagens.

### 2.2 Problema e Objetivo

Este trabalho prático teve como principais objetivos:

- aumentar a experiência em engenharia de linguagens e em programação generativa (gramatical), reforçando a capacidade de escrever gramáticas, quer independentes de contexto (GIC), quer tradutoras (GT);
- desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, a partir de uma gramática tradutora;
- desenvolver um compilador gerando código para um objetivo específico;
- utilizar geradores de compiladores baseados em gramáticas tradutoras, concretamente o Yacc, versão PLY do Python, completado pelo gerador de analisadores léxicos Lex, também versão PLY do Python.

### 2.3 Estrutura do Relatório

Este relatório possui 5 capítulos.

No capítulo 1, Introdução, é feito o enquadramento e contextualização do projeto, bem como uma breve descrição do problema em mãos. É feita também, uma referência às decisões tomadas no trabalho.

No capítulo 2, Análise e Especificação, é feita uma descrição informal do desafio em questão, além de serem especificados os requisitos necessários.

No capítulo 3, Implementação, é explicada, detalhadamente, a resolução do problema, nomeadamente o desenvolvimento da gramática, a parte relativa ao analisador léxico e a parte relativa ao analisador sintático.

No capítulo 4, Codificação e Testes, são apresentados testes realizados e os resultados obtidos.

No capítulo 5, Conclusão, é feita uma análise do projeto.

## 3 Análise e Especificação

### 3.1 Descrição informal do problema

Desenvolver um conversor de toml para json. Com recurso a uma gramática tradutora gerar código json a partir de toml.

### 3.2 Especificação dos requisitos

De forma a responder devidamente ao problema proposto foram definidos inicialmente alguns requisitos a cumprir ao longo da sua elaboração:

- Elaborar uma linguagem bem definida, ou seja, respeitar a documentação do *toml*
- Efetuar a análise léxica e sintática
- Elaborar as regras de tradução para toml
- Elaborar uma interface

## 4 Conceção/Desenho da Resolução

### 4.1 Gramática Independente de Contexto (GIC)

O primeiro passo do desenvolvimento deste projeto, consistiu na criação de uma gramática regular independente do contexto.

A gramática inicia com o símbolo não terminal *Toml* que deriva em dois símbolos não terminais designados por *Attributes* e *Tables*. Isto permite dividir o programa em dois grandes blocos: as declarações de variáveis que não pertencem a nenhuma *Table*, sempre no início do programa, e todo o tipo de *tables* restantes.

```
p0 Toml => Attributes Tables
```

Figure 1: Regra de produção referente a **Toml**

Em relação às *tables*, estas podem ser vazias ou então podem se representar de duas formas: como uma table normal ou um array de tables.

```
p1 Tables => €  
p2      | Tables Table  
p3      | Tables ARRAY_TABLES
```

Figure 2: Regra de produção referente às **Tables**

Quando uma table é reconhecida esta pode não ter atributos guardando armazenando apenas o nome da table. ou ter atributos e armazenar os atributos dentro no dicionário do nome da table.

```
p4 Table => TABLE  
p5      | TABLE Attributes
```

Figure 3: Regra de produção referente a **Toml**

Tal como a table, o array de table pode não ter argumentos e assim guardar apenas

```
p6 Array_Tables => ARRAY_TABLES  
p7             | ARRAY_TABLES Attributes
```

Figure 4: Regra de produção referente a **Toml**

Nesta gramática chamamos as variáveis de atributos podem ser vazio ou ter um ou vários atributos. Sendo que cada atributo pode

```
p8 Attributes => €  
p9          | Attribute Attributes  
p10 Attribute => NAME EQUALS Value
```

Figure 5: Regra de produção referente a **Toml**

Um array é então representado pelos parênteses retos no início e no final contendo valores(*array\_values*) no meio destas chavetas. Este *array\_values* pode então ser vazio ou possuir valores.

```
p11 Array => LEFT_BRACKET Array_Values RIGHT_BRACKET
p12 Array_Values => €
p13          | Value
p14          | Array_Values COMMA Value
```

Figure 6: Regra de produção referente a **Toml**

Um dicionário é então representado pelas chavetas no início e no final contendo valores(*dict\_values*) no meio destas chavetas. Este *dict\_values* pode então ser vazio ou possuir atributos.

```
p15 Dict => LEFT_CURLY Dict_Values RIGHT_CURLY
p16 Dict_Values => €
p17          | Attribute
p18          | Dict_Values COMMA Attribute
```

Figure 7: Regra de produção referente aos dicionários

Todos os valores possíveis podem ser vistos na figura abaixo. Para facilitar a implementação consideramos um array ou dicionário, que não seja de Tables, como um valor.

```
p19 Value => Array
p20          | Dict
p21          | OFFSET_DATE_TIME
p22          | LOCAL_DATE_TIME
p23          | LOCAL_DATE
p24          | LOCAL_TIME
p25          | DECIMAL
p26          | EXPONENT
p27          | OCTA
p28          | HEXA
p29          | BIN
p30          | INTEGER
p31          | BOOLEAN
p32          | STRING_BIG
p33          | STRING
```

Figure 8: Regra de produção referente aos possíveis valores



## 4.2 Implementação

### 4.2.1 Analisador Léxico

Como é possível ver pelo código abaixo, foram definidos todos os símbolos terminais possíveis. Para alguns destes símbolos houve a necessidade de definir uma expressão regular adequada para os identificar.

Para que o *lexer* conseguisse distinguir corretamente todos os símbolos foi necessário ter atenção à ordem em que estes eram definidos.

É importante referir que, para que seja possível haver uma conversão correta usamos a função *int()* para converter as strings para números e depois, caso se aplique, passar de número binário, hexadecimal, ou octal para inteiro, a função *float()* para decimal e verificamos se também pode ser um booleano. Depois desta conversão armazenamos o resultado final em *t.value*.

```
tokens = (
    'TABLE',
    'STRING',
    'STRING_BIG',
    'INTEGER',
    'DECIMAL',
    'EXPONENT',
    'LOCAL_DATE',
    'LOCAL_TIME',
    'LOCAL_DATE_TIME',
    'OFFSET_DATE_TIME',
    'LEFT_BRACKET',
    'RIGHT_BRACKET',
    'LEFT_CURLY',
    'RIGHT_CURLY',
    'EQUALS',
    'COMMA',
    'BOOLEAN',
    'ARRAY_TABLES',
    'OCTA',
    'HEXA',
    'BIN',
    'NAME'
)

# Expressões regulares para cada token
t_LEFT_BRACKET = r'\['
t_RIGHT_BRACKET = r'\]'
t_LEFT_CURLY = r'\{'
t_RIGHT_CURLY = r'\}'
t_EQUALS = r'\='
t_COMMA = r'\,'

# Ignora espaços em branco e tabulações
t_ignore = '\t\r\n'

def t_OFFSET_DATE_TIME(t):
    r'(\d{4}\-\d{2}\-\d{2}T\d{2}\:\d{2}\:\d{2})(Z)?(\.\d+)?(\-\d+:\d+)?'
    return t
```

```

def t_LOCAL_DATE_TIME(t):
    r'(\d{4}\-\d{2}\-\d{2}T\d{2}\:\d{2}\:\d{2})(\.\d+)?'
    return t

def t_LOCAL_DATE(t):
    r'\d{4}\-\d{2}\-\d{2}'
    return t

def t_LOCAL_TIME(t):
    r'(\d{2}\:\d{2}\:\d{2})(\.\d+)?'
    return t

def t_BOOLEAN(t):
    r'(true|false)'
    if str(t.value).lower() == "true": t.value = True
    else: t.value = False
    return t

def t_EXPONENT(t):
    r'((\+|\-)?\d+(\.\d+)?(e|E)(\+|\-)?\d+)'
    return t

def t_DECIMAL(t):
    r'(\+|\-)?(\d+\.\d+)'
    t.value = float(t.value)
    return t

def t_HEX(t):
    r'(0x)([0-9a-f]+)'
    t.value = int(t.value[2:], 16)
    return t

def t_OCTA(t):
    r'(0o)([0-8]+)'
    t.value = int(t.value[2:], 8)
    return t

def t_BIN(t):
    r'(0b)([0-1]+)'
    t.value = int(t.value[2:], 2)
    return t

def t_INTEGER(t):
    r'(\+|\-)?(\d+)'
    t.value = int(t.value)
    return t

def t_NAME(t):
    r'[\w\-\.\+]'
    return t

def t_TABLE(t):
    r'(<=\n)(\[([\w\-\.\+\\])(?!=)'

```



#### 4.2.2 Gramática Tradutora (GT)

Através do *Yacc* versão **PLY** do *python* foi criado uma gramática tradutora, a partir da qual foi possível desenvolver um processador de linguagens. Esta teve por base a gramática independente de contexto desenvolvida numa primeira fase dependendo dos resultados que eram necessários obter.

A abordagem adotada no desenvolvimento da gramática foi um parser *bottom-up*, de forma a satisfazer a condição LALR(1).

Posto isto, o código seguinte retrata o que foi feito inicialmente no ficheiro *Yacc* antes de o grupo começar a desenvolver uma solução que consiga fazer a conversão.

```
def p_Toml(p):
    """
    Toml : Tables
        | Attributes
        | Attributes Tables
    """
    pass

def p_Tables_Table(p):
    """
    Tables : Table
        | Tables Table
    """
    pass

def p_Tables_Array_Table(p):
    """
    Tables : Array_Tables
        | Tables Array_Tables
    """
    pass

def p_Table(p):
    """
    Table : TABLE
        | TABLE Attributes
    """
    pass

def p_Array_Tables(p):
    """
    Array_Tables : ARRAY_TABLES
        | ARRAY_TABLES Attributes
    """
    pass

def p_Attributes(p):
    """
    Attributes : Attribute
        | Attribute Attributes
    """
    pass
```

```

def p_Attribute(p):
    '''
        Attribute : NAME EQUALS Value
                  | STRING EQUALS Value
    '''
    pass

def p_Array(p):
    '''
        Array : LEFT_BRACKET RIGHT_BRACKET
              | LEFT_BRACKET Array_Values RIGHT_BRACKET
    '''
    pass

def p_Array_Values(p):
    '''
        Array_Values : Value
                    | Array_Values COMMA Value
    '''
    pass

def p_Dict(p):
    '''
        Dict : LEFT_CURLY RIGHT_CURLY
             | LEFT_CURLY Dict_Values RIGHT_CURLY
    '''
    pass

def p_Dict_Values(p):
    '''
        Dict_Values : Attribute
                   | Dict_Values COMMA Attribute
    '''
    pass

def p_Value(p):
    '''
        Value : Array
              | Dict
              | OFFSET_DATE_TIME
              | LOCAL_DATE_TIME
              | LOCAL_DATE
              | LOCAL_TIME
              | DECIMAL
              | EXPONENT
              | OCTA
              | HEXA
              | BIN
              | INTEGER
              | BOOLEAN
              | STRING_BIG
              | STRING
    '''
    pass

```

```

# Erro de sintaxe
def p_error(p):
    print(p)
    if p:
        print(f"Erro de sintaxe na linha {p.lineno}: token inesperado {p.value}")
    else:
        print("Erro de sintaxe: fim de entrada inesperado")

```

#### 4.2.3 Compilador Yacc

Em consequência daquilo que foi mencionado no tópico anterior, foram depois completadas as definições reazadas para cada símbolo não terminal da gramática, de forma a obter um compilador capaz de fazer a conversão.

Assim sendo o resultado final foi o seguinte:

```

def p_Toml(p):
    '''
    Toml : Tables
        | Attributes
        | Attributes Tables
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[1].update(p[2])
        p[0] = p[1]

def p_Tables_Table(p):
    '''
    Tables : Table
        | Tables Table
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        lista = list(p[2].keys())

        nova_lista = lista[0].split('.')
        temp = p[1]
        if len(nova_lista) > 1 :

            for x in nova_lista[:-1]:
                if x not in temp:
                    temp[x] = {}
                temp = temp[x]
            temp[nova_lista[-1]] = p[2][lista[0]]
        p[0] = p[1]
    else:
        p[1].update(p[2])
        p[0] = p[1]

def p_Tables_Array_Table(p):

```

```

'''
Tables : Array-Tables
        | Tables Array-Tables
'''
if len(p) == 2:
    p[0] = p[1]
else:
    lista = list(p[2].keys())

    nova_lista = lista[0].split('.')
    temp = p[1]
    if len(nova_lista) > 1 :

        for x in nova_lista[:-1]:
            if x not in temp:
                temp[x] = {}
            temp = temp[x]
        if nova_lista[-1] in temp.keys():
            temp[nova_lista[-1]].append(p[2][lista[0]][0])
        else:
            temp[nova_lista[-1]] = p[2][lista[0]]
        p[0] = p[1]
    else:
        if nova_lista[-1] in temp.keys():
            temp[nova_lista[-1]].append(p[2][lista[0]][0])
        else:
            temp[nova_lista[-1]] = p[2][lista[0]]
        p[0] = p[1]

def p_Table(p):
'''
Table : TABLE
        | TABLE Attributes
'''
if len(p) == 2:
    p[0] = {p[1] : {}}
else:
    p[0] = {p[1] : p[2]}

def p_Array-Tables(p):
'''
Array-Tables : ARRAY-TABLES
               | ARRAY-TABLES Attributes
'''
if len(p) == 2:
    p[0] = {p[1]:[{}]}
else:
    p[0] = {p[1]:[p[2]]}

def p_Attributes(p):
'''
Attributes : Attribute
             | Attribute Attributes
'''

```

```

    if len(p) == 2:
        p[0] = p[1]
    else:
        p[1].update(p[2])
        p[0] = p[1]

def p_Attribute(p):
    Attribute : NAME EQUALS Value
               | STRING EQUALS Value
    Attribute : NAME EQUALS Value
               | STRING EQUALS Value
    p[0] = {p[1] : p[3]}

def p_Array(p):
    Array : LEFT_BRACKET RIGHT_BRACKET
           | LEFT_BRACKET Array-Values RIGHT_BRACKET
    Array : LEFT_BRACKET RIGHT_BRACKET
           | LEFT_BRACKET Array-Values RIGHT_BRACKET
    if len(p) == 3:
        p[0] = []
    else:
        p[0] = p[2]

def p_Array_Values(p):
    Array-Values : Value
                  | Array-Values COMMA Value
    Array-Values : Value
                  | Array-Values COMMA Value
    if len(p) == 2:
        p[0] = [p[1]]
    else:
        p[1].append(p[3])
        p[0] = p[1]

def p_Dict(p):
    Dict : LEFT_CURLY RIGHT_CURLY
          | LEFT_CURLY Dict-Values RIGHT_CURLY
    Dict : LEFT_CURLY RIGHT_CURLY
          | LEFT_CURLY Dict-Values RIGHT_CURLY
    if len(p) == 3:
        p[0] = {}
    else:
        p[0] = p[2]

def p_Dict_Values(p):
    Dict-Values : Attribute
                  | Dict-Values COMMA Attribute
    Dict-Values : Attribute
                  | Dict-Values COMMA Attribute
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[1].update(p[3])
        p[0] = p[1]

```



```

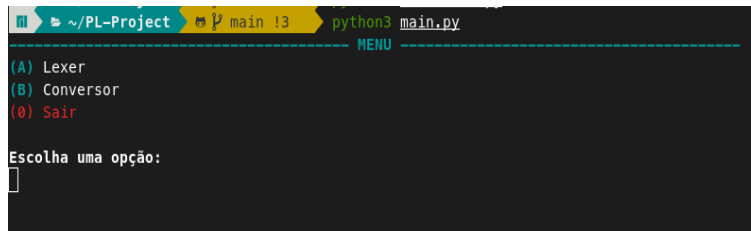
def p_Value(p):
    , , ,
    Value : Array
           | Dict
           | OFFSET_DATE_TIME
           | LOCAL_DATE_TIME
           | LOCAL_DATE
           | LOCAL_TIME
           | DECIMAL
           | EXPONENT
           | OCTA
           | HEXA
           | BIN
           | INTEGER
           | BOOLEAN
           | STRING_BIG
           | STRING
    , , ,
    p[0] = p[1]

# Erro de sintaxe
def p_error(p):
    print(p)
    if p:
        print(f"Erro de sintaxe na linha {p.lineno}: token inesperado {p.value}")
    else:
        print("Erro de sintaxe: fim de entrada inesperado")

```

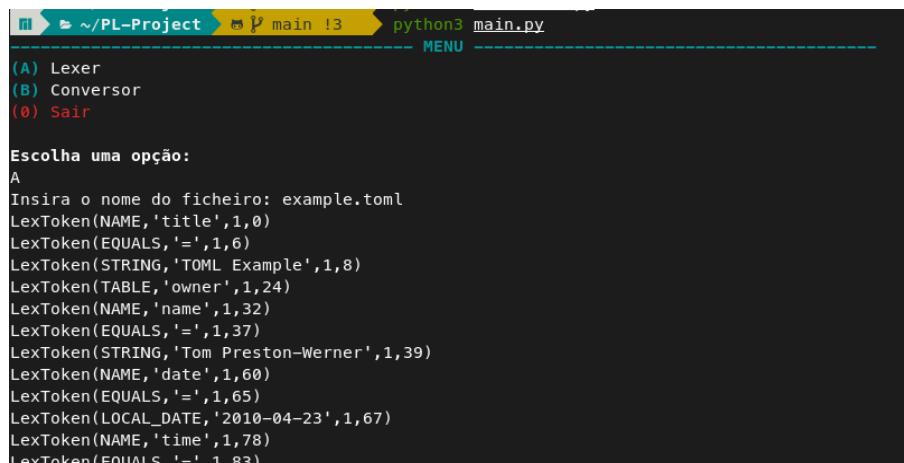
#### 4.2.4 Compilação e Execução

Para compilar e executar o programa, foi realizado um menu onde o utilizador poderá escolher se pretende consultar o output do *lexer* de um ficheiro, ou então pode escolher se quer fazer a conversão entre as duas linguagens. Como tarefa adicional criamos um ficheiro adicional com uma interface, utilizando a biblioteca *tkinter* do *python* de forma a imitar um conversor online.



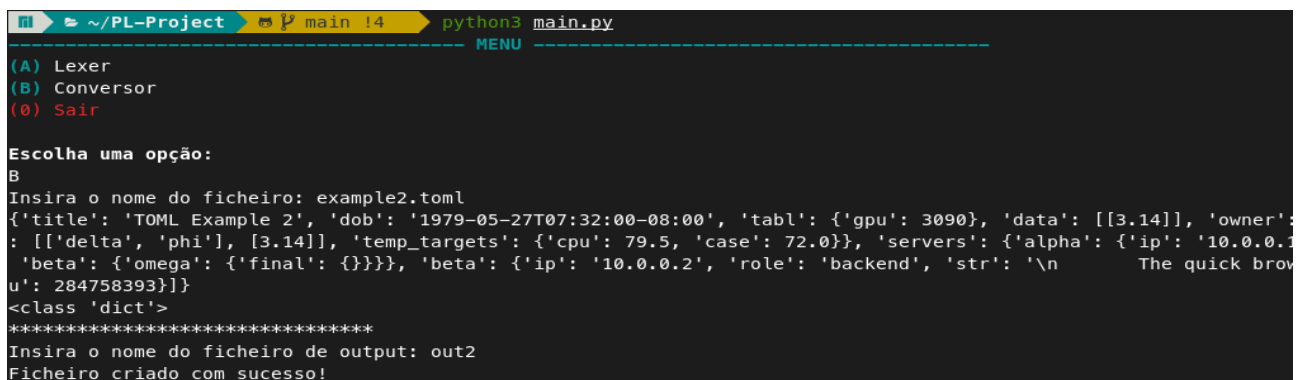
```
~/PL-Project main 13 python3 main.py
----- MENU -----
(A) Lexer
(B) Conversor
(0) Sair
Escolha uma opção:
█
```

Figure 9: Menu



```
~/PL-Project main 13 python3 main.py
----- MENU -----
(A) Lexer
(B) Conversor
(0) Sair
Escolha uma opção:
A
Insira o nome do ficheiro: example.toml
LexToken(NAME, 'title', 1, 0)
LexToken(EQUALS, '=', 1, 6)
LexToken(STRING, 'TOML Example', 1, 8)
LexToken(TABLE, 'owner', 1, 24)
LexToken(NAME, 'name', 1, 32)
LexToken(EQUALS, '=', 1, 37)
LexToken(STRING, 'Tom Preston-Werner', 1, 39)
LexToken(NAME, 'date', 1, 60)
LexToken(EQUALS, '=', 1, 65)
LexToken(LOCAL_DATE, '2010-04-23', 1, 67)
LexToken(NAME, 'time', 1, 78)
LexToken(EQUALS, '=', 1, 83)
```

Figure 10: Lexer



```
~/PL-Project main 14 python3 main.py
----- MENU -----
(A) Lexer
(B) Conversor
(0) Sair
Escolha uma opção:
B
Insira o nome do ficheiro: example2.toml
{'title': 'TOML Example 2', 'dob': '1979-05-27T07:32:00-08:00', 'tabl': {'gpu': 3090}, 'data': [[3.14]], 'owner':
: [['delta', 'phi'], [3.14]], 'temp_targets': {'cpu': 79.5, 'case': 72.0}, 'servers': {'alpha': {'ip': '10.0.0.1
'beta': {'omega': {'final': {}}}}}, 'beta': {'ip': '10.0.0.2', 'role': 'backend', 'str': '\n        The quick brov
u': 284758393}}]
<class 'dict'>
*****
Insira o nome do ficheiro de output: out2
Ficheiro criado com sucesso!
```

Figure 11: Conversor

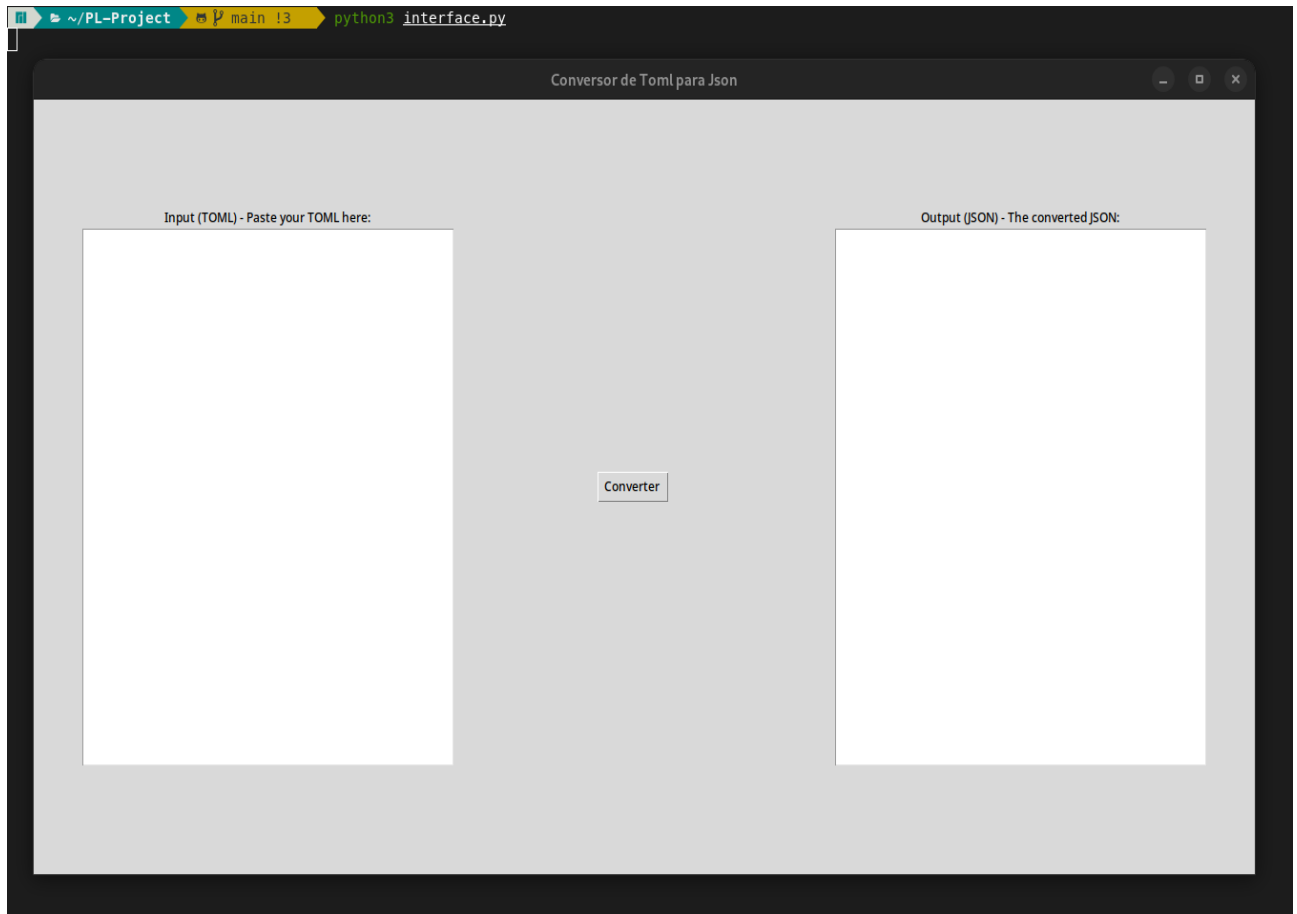


Figure 12: Interface

## 5 Codificação e Testes

### 5.1 Testes realizados e Resultados

Ao longo do desenvolvimento deste projeto foram realizados vários testes que estão presentes em arquivos separados. Nos dois exemplos abaixo podemos ver em primeiro lugar o exemplo apresentado no enunciado e depois um exemplo completo, com todas as funcionalidades apresentadas na documentação do *toml*, juntamente com o output de cada um.

- Exemplo do enunciado

```
title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
date = 2010-04-23
time = 21:30:00

[database]
server = "192.168.1.1"
ports = [
    8001,
    8001,
    8002
]
connection_max = 5000
enabled = true

[servers]
[servers.alpha]
ip = "10.0.0.1"
dc = "eqdc10"

[servers.beta]
ip = "10.0.0.2"
dc = "eqdc10"

# Line breaks are OK when inside arrays
hosts = [
    "alpha",
    "omega"
]
```

- Output:

```
{
  "title": "TOML Example",
  "owner": {
    "name": "Tom Preston-Werner",
    "date": "2010-04-23",
    "time": "21:30:00"
  },
  "database": {
    "server": "192.168.1.1",
    "ports": [
      8001,
```

```

        8001,
        8002
    ],
    "connection_max": 5000,
    "enabled": true
},
"servers": {
    "alpha": {
        "ip": "10.0.0.1",
        "dc": "eqdc10"
    },
    "beta": {
        "ip": "10.0.0.2",
        "dc": "eqdc10",
        "hosts": [
            "alpha",
            "omega"
        ]
    }
}
}
}

```

- Exemplo criado pelo grupo

```

# This is a TOML document

title = "TOML Example 2"
dob = 1979-05-27T07:32:00-08:00
tabl = { gpu = 3090 }
data = [ [ 3.14 ] ]

[owner]
name = "Tom Preston-Werner"

[database]
enabled = true
ports = [ 8000, 8001, 8002 ]
data = [ ["delta", "phi"], [3.14] ]
temp_targets = { cpu = 79.5, case = 72.0 }

[servers]

[servers.alpha]
ip = "10.0.0.1"
role = "frontend"
hex = 0xdeadbeef
oct = 0o01234567
bin = 0b11010110
flt = 1e06
fl = -2E-2

[servers.beta]
ip = "10.0.0.2"
role = "backend"
str = ""

```

```

The quick brown
fox jumps over
the lazy dog
"""

```

```

[[products]] # empty table within the array
[[products]]
name = "Nail"
sku = 284758393

```

```

[servers.alpha.beta.omega.final]

```

- Output:

```

{
  "title": "TOML Example 2",
  "dob": "1979-05-27T07:32:00-08:00",
  "tbl": {
    "gpu": 3090
  },
  "data": [
    [
      3.14
    ]
  ],
  "owner": {
    "name": "Tom Preston-Werner"
  },
  "database": {
    "enabled": true,
    "ports": [
      8000,
      8001,
      8002
    ],
    "data": [
      [
        "delta",
        "phi"
      ],
      [
        3.14
      ]
    ]
  },
  "temp_targets": {
    "cpu": 79.5,
    "case": 72.0
  }
},
"servers": {
  "alpha": {
    "ip": "10.0.0.1",
    "role": "frontend",
    "hex": 3735928559,
    "oct": 342391,

```

```

        "bin": 214,
        "flt": "1e06",
        "fl": "-2E-2",
        "beta": {
            "omega": {
                "final": {}
            }
        },
        "beta": {
            "ip": "10.0.0.2",
            "role": "backend",
            "str": "\n
the lazy dog\n
        "
        The quick brown \n
        fox jumps over \n
    },
    "products": [
        {},
        {
            "name": "Nail",
            "sku": 284758393
        }
    ]
}

```

## 5.2 Problemas e Decisões

Numa fase inicial começamos por criar uma string com o output mas rapidamente o grupo decidiu alterar a estrutura para um dicionário de forma a ser mais fácil de fazer verificações e manusear dado o contexto do problema.

## 6 Conclusão

A resolução deste projeto permitiu ao grupo consolidar toda a matéria lecionada ao longo das aulas teóricas e práticas relativa às gramáticas independentes do contexto e tradutoras. Além disto, permitiu aprofundar ainda mais o nosso conhecimento em Python.

Consideramos que realizamos o trabalho com sucesso, na medida em que respondemos a todos os requisitos pedidos no enunciado, bem como tivemos a liberdade de acrescentar alguma *features* , obtendo assim resultados bastante plausíveis.



## 7 Apêndice

Ficheiro lexer.py

```
import ply.lex as lex

# Lista de tokens
tokens = (
    'TABLE',
    'STRING',
    'STRING_BIG',
    'INTEGER',
    'DECIMAL',
    'EXPONENT',
    'LOCAL_DATE',
    'LOCAL_TIME',
    'LOCAL_DATE_TIME',
    'OFFSET_DATE_TIME',
    'LEFT_BRACKET',
    'RIGHT_BRACKET',
    'LEFT_CURLY',
    'RIGHT_CURLY',
    'EQUALS',
    'COMMA',
    'BOOLEAN',
    'ARRAY_TABLES',
    'OCTA',
    'HEXA',
    'BIN',
    'NAME'
)

# Expressões regulares para cada token
t_LEFT_BRACKET = r'\['
t_RIGHT_BRACKET = r'\]'
t_LEFT_CURLY = r'\{'
t_RIGHT_CURLY = r'\}'
t_EQUALS = r'\='
t_COMMA = r','

# Ignora espaços em branco e tabulações
t_ignore = '\t\r\n'

def t_OFFSET_DATE_TIME(t):
    r'(\d{4}\-\d{2}\-\d{2}T\d{2}:\d{2}:\d{2})(Z)?(\.\d+)?(\-\d+:\d+)?'
    return t

def t_LOCAL_DATE_TIME(t):
    r'(\d{4}\-\d{2}\-\d{2}T\d{2}:\d{2}:\d{2})(\.\d+)?'
    return t

def t_LOCAL_DATE(t):
    r'\d{4}\-\d{2}\-\d{2}'
    return t
```

```

def t_LOCAL_TIME(t):
    r'(\d{2}\:\d{2}\:\d{2})(\.\d+)?'
    return t

def t_BOOLEAN(t):
    r'(true|false)'
    if str(t.value).lower() == "true": t.value = True
    else: t.value = False
    return t

def t_EXPONENT(t):
    r'((\+|\-)?\d+(\.\d+)?(e|E)(\+|\-)?\d+)'
    return t

def t_DECIMAL(t):
    r'(\+|\-)?(\d+\.\d+)'
    t.value = float(t.value)
    return t

def t_HEX(t):
    r'(0x)([0-9a-f]+)'
    t.value = int(t.value[2:], 16)
    return t

def t_OCTA(t):
    r'(0o)([0-8]+)'
    t.value = int(t.value[2:], 8)
    return t

def t_BIN(t):
    r'(0b)([0-1]+)'
    t.value = int(t.value[2:], 2)
    return t

def t_INTEGER(t):
    r'(\+|\-)?(\d+)'
    t.value = int(t.value)
    return t

def t_NAME(t):
    r'[\w\-\.\_]+'
    return t

def t_TABLE(t):
    r'(?<=\n)(\[([ \w\-\_]+\.\?)+\])(?!=)'
    t.value = t.value[1:-1]
    return t

def t_ARRAY_TABLES(t):
    r'(?<=\n)(\[([ \w\-\_]+\.\?)+\])(?!=)'
    t.value = t.value[2:-2]
    return t

```



```

        p[1].update(p[2])
        p[0] = p[1]

def p_Tables_Table(p):
    '''
    Tables : Table
            | Tables Table
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        lista = list(p[2].keys())

        nova_lista = lista[0].split('.')
        temp = p[1]
        if len(nova_lista) > 1 :

            for x in nova_lista[: -1]:
                if x not in temp:
                    temp[x] = {}
                temp = temp[x]
            temp[nova_lista[-1]] = p[2][lista[0]]
            p[0] = p[1]
        else:
            p[1].update(p[2])
            p[0] = p[1]

def p_Tables_Array_Table(p):
    '''
    Tables : Array_Tables
            | Tables Array_Tables
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        lista = list(p[2].keys())

        nova_lista = lista[0].split('.')
        temp = p[1]
        if len(nova_lista) > 1 :

            for x in nova_lista[: -1]:
                if x not in temp:
                    temp[x] = {}
                temp = temp[x]
            if nova_lista[-1] in temp.keys():
                temp[nova_lista[-1]].append(p[2][lista[0]][0])
            else:
                temp[nova_lista[-1]] = p[2][lista[0]]
            p[0] = p[1]
        else:
            if nova_lista[-1] in temp.keys():
                temp[nova_lista[-1]].append(p[2][lista[0]][0])
            else:

```

```

        temp[nova_lista[-1]] = p[2][lista[0]]
        p[0] = p[1]

def p_Table(p):
    '''
    Table : TABLE
           | TABLE Attributes
    '''
    if len(p) == 2:
        p[0] = {p[1] : {}}
    else:
        p[0] = {p[1] : p[2]}

def p_Array_Tables(p):
    '''
    Array_Tables : ARRAY_TABLES
                  | ARRAY_TABLES Attributes
    '''
    if len(p) == 2:
        p[0] = {p[1]:[{}]}
    else:
        p[0] = {p[1]:[p[2]]}

def p_Attributes(p):
    '''
    Attributes : Attribute
                | Attribute Attributes
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[1].update(p[2])
        p[0] = p[1]

def p_Attribute(p):
    '''
    Attribute : NAME EQUALS Value
               | STRING EQUALS Value
    '''
    p[0] = {p[1] : p[3]}

def p_Array(p):
    '''
    Array : LEFT_BRACKET RIGHT_BRACKET
           | LEFT_BRACKET Array_Values RIGHT_BRACKET
    '''
    if len(p) == 3:
        p[0] = []
    else:
        p[0] = p[2]

def p_Array_Values(p):
    '''
    Array_Values : Value
    '''

```

```

        , , , | Array-Values COMMA Value
    if len(p) == 2:
        p[0] = [p[1]]
    else:
        p[1].append(p[3])
        p[0] = p[1]

def p_Dict(p):
    , , ,
    Dict : LEFT_CURLY RIGHT_CURLY
        | LEFT_CURLY Dict-Values RIGHT_CURLY
    , , ,
    if len(p) == 3:
        p[0] = {}
    else:
        p[0] = p[2]

def p_Dict-Values(p):
    , , ,
    Dict-Values : Attribute
        | Dict-Values COMMA Attribute
    , , ,
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[1].update(p[3])
        p[0] = p[1]

def p_Value(p):
    , , ,
    Value : Array
        | Dict
        | OFFSET-DATE-TIME
        | LOCAL-DATE-TIME
        | LOCAL-DATE
        | LOCAL-TIME
        | DECIMAL
        | EXPONENT
        | OCTA
        | HEXA
        | BIN
        | INTEGER
        | BOOLEAN
        | STRING_BIG
        | STRING
    , , ,
    p[0] = p[1]

# Erro de sintaxe
def p_error(p):
    print(p)
    if p:
        print(f"Erro de sintaxe na linha {p.lineno}: token inesperado {p.value}")

```

```

        else:
            print("Erro_de_sintaxe:_fim_de_entrada_inesperado")

# Build parser
parser = yacc.yacc(debug=True)

# Ler o arquivo de entrada
def converter():
    file = input("Insira_o_nome_do_ficheiro:_")
    with open(file, 'r') as f:
        data = f.read()
        out = parser.parse(data)
        print(out)
        print((type(out)))
        print("*****")
        output = input("Insira_o_nome_do_ficheiro_de_output:_")
        out = "../out/" + output
        with open(out, 'w') as o:
            json.dump(out, o, indent=4)
            print("Ficheiro_criado_com_sucesso!")

Ficheiro main.py

from lexer import tokenizer
from yacc import converter

def print_menu():
    print("\033[36m\033[1m_____MENU_____ \033[0m")
    print("\033[1m\033[36m(A)\033[0m_Lexer")
    print("\033[1m\033[36m(B)\033[0m_Conversor")
    print("\033[91m(0)_Sair\033[0m")

while(True):
    print_menu()
    print("\n\033[1mEscolha_uma_opcao:_\033[0m")
    op = input()
    op=op.upper()
    if (op=='A'):
        tokenizer()
    elif (op=='B'):
        converter()
    elif (op=='0'):
        break
    else: print("\033[91m\033[1mOpcao_invalida.\033[0m")

```

```

Ficheiro interface.py

import tkinter as tk
from tkinter import ttk
import json
from lexer import tokens
import ply.yacc as yacc

```

*# Define as regras gramaticais para a conversão de TOML para JSON*

```
def p_Toml(p):
    '''
    Toml : Tables
          | Attributes
          | Attributes Tables
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[1].update(p[2])
        p[0] = p[1]

def p_Tables_Table(p):
    '''
    Tables : Table
            | Tables Table
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        lista = list(p[2].keys())

        nova_lista = lista[0].split('.')
        temp = p[1]
        if len(nova_lista) > 1 :

            for x in nova_lista[:-1]:
                if x not in temp:
                    temp[x] = {}
                temp = temp[x]
            temp[nova_lista[-1]] = p[2][lista[0]]
        p[0] = p[1]
    else:
        p[1].update(p[2])
        p[0] = p[1]

def p_Tables_Array_Table(p):
    '''
    Tables : Array_Tables
            | Tables Array_Tables
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        lista = list(p[2].keys())

        nova_lista = lista[0].split('.')
        temp = p[1]
        if len(nova_lista) > 1 :

            for x in nova_lista[:-1]:
                if x not in temp:
                    temp[x] = {}
```



```

        temp = temp[x]
        if nova_lista[-1] in temp.keys():
            temp[nova_lista[-1]].append(p[2][lista[0]][0])
        else:
            temp[nova_lista[-1]] = p[2][lista[0]]
        p[0] = p[1]
    else:
        if nova_lista[-1] in temp.keys():
            temp[nova_lista[-1]].append(p[2][lista[0]][0])
        else:
            temp[nova_lista[-1]] = p[2][lista[0]]
        p[0] = p[1]

def p_Table(p):
    '''
    Table : TABLE
           | TABLE Attributes
    '''
    if len(p) == 2:
        p[0] = {p[1] : {}}
    else:
        p[0] = {p[1] : p[2]}

def p_Array_Tables(p):
    '''
    Array_Tables : ARRAY_TABLES
                  | ARRAY_TABLES Attributes
    '''
    if len(p) == 2:
        p[0] = {p[1]:[{}]}
    else:
        p[0] = {p[1]:[p[2]]}

def p_Attributes(p):
    '''
    Attributes : Attribute
                | Attribute Attributes
    '''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[1].update(p[2])
        p[0] = p[1]

def p_Attribute(p):
    '''
    Attribute : NAME EQUALS Value
               | STRING EQUALS Value
    '''
    p[0] = {p[1] : p[3]}

def p_Array(p):
    '''
    Array : LEFT_BRACKET RIGHT_BRACKET
    '''

```

```

    , , , | LEFT_BRACKET Array_Values RIGHT_BRACKET
if len(p) == 3:
    p[0] = []
else:
    p[0] = p[2]

def p_Array_Values(p):
    , , ,
    Array_Values : Value
    , , , | Array_Values COMMA Value
    , , ,
    if len(p) == 2:
    p[0] = [p[1]]
    else:
    p[1].append(p[3])
    p[0] = p[1]

def p_Dict(p):
    , , ,
    Dict : LEFT_CURLY RIGHT_CURLY
    , , , | LEFT_CURLY Dict_Values RIGHT_CURLY
    , , ,
    if len(p) == 3:
    p[0] = {}
    else:
    p[0] = p[2]

def p_Dict_Values(p):
    , , ,
    Dict_Values : Attribute
    , , , | Dict_Values COMMA Attribute
    , , ,
    if len(p) == 2:
    p[0] = p[1]
    else:
    p[1].update(p[3])
    p[0] = p[1]

def p_Value(p):
    , , ,
    Value : OFFSET_DATE_TIME
    | LOCAL_DATE_TIME
    | LOCAL_DATE
    | LOCAL_TIME
    | STRING_BIG
    | STRING
    | Array
    | EXPONENT
    | OCTA
    | HEXA
    | BIN
    | DECIMAL
    | INTEGER

```

```

        | BOOLEAN
        | Dict
    , , ,
    p[0] = p[1]

# Erro de sintaxe
def p_error(p):
    print(p)
    if p:
        print(f"Erro de sintaxe na linha {p.lineno}: token inesperado {p.value}")
    else:
        print("Erro de sintaxe: fim de entrada inesperado")

# Build parser
parser = yacc.yacc(debug=True)

def aux():
    original_code = original_entry.get("1.0", tk.END)
    converted_code = parser.parse(original_code)
    print(converted_code)
    print((type(converted_code)))
    print("*****")
    if converted_code == None:
        converted_entry.delete("1.0", tk.END)
        converted_entry.insert(tk.END, "Conversão não suportada (Input inválido).")
    else:
        out = json.dumps(converted_code, indent=4)
        converted_entry.delete("1.0", tk.END)
        converted_entry.insert(tk.END, out)

# Cria a janela principal
window = tk.Tk()
window.title("Conversor de Toml para Json")
window.geometry("1280x720")

# Cria o dos widgets
original_frame = ttk.Frame(window)
original_frame.pack(side=tk.LEFT, padx=50, pady=100)

original_label = ttk.Label(original_frame, text="Input (TOML) -- Paste your TOML here:")
original_label.pack()

original_entry = tk.Text(original_frame, height=80, width=55)
original_entry.pack()

convert_button = ttk.Button(window, text="Converter", command=aux)
convert_button.pack(side=tk.LEFT, padx=100, pady=10)

converted_frame = ttk.Frame(window)
converted_frame.pack(side=tk.RIGHT, padx=50, pady=100)

converted_label = ttk.Label(converted_frame, text="Output (JSON) -- The converted JSON:")

```

```
converted_label.pack()

converted_entry = tk.Text(converted_frame, height=80, width=55)
converted_entry.pack()

# Inicia o loop do tkinter
window.mainloop()
```