

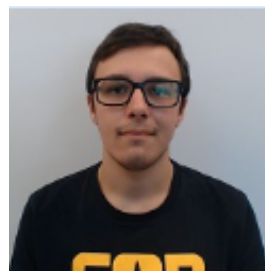


**Universidade do Minho**  
Escola de Engenharia

Processamento de Linguagens  
Conversor toml-json - Grupo 22  
2022/2023

Afonso Xavier Cardoso Marques a94940  
Ana Filipa da Cunha Rebelo a90234  
Tomás Cardoso Francisco a93193

Maio 2023



# Índice

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Análise e Especificação do Problema</b>	<b>3</b>
2.1	Descrição Informal . . . . .	3
2.2	Requisitos . . . . .	3
<b>3</b>	<b>Desenho da Solução</b>	<b>3</b>
3.1	Módulo main.py . . . . .	4
3.2	Módulo converter.py . . . . .	4
3.3	Módulo lexico.py . . . . .	5
3.4	Módulo criaDicionario.py . . . . .	7
<b>4</b>	<b>Testes</b>	<b>10</b>
4.1	Teste 1 . . . . .	10
4.2	Teste 2 . . . . .	14
4.3	Teste 3 . . . . .	16
<b>5</b>	<b>Conclusão</b>	<b>18</b>
<b>6</b>	<b>Anexos</b>	<b>19</b>
6.1	Anexo A . . . . .	19
6.2	Anexo B . . . . .	21

# 1 Introdução

O presente relatório enquadra-se na unidade curricular Processamento de Linguagens, na qual nos foi proposto o desenvolvimento de uma ferramenta capaz de converter um subconjunto de Toml para JSON. A linguagem de programação utilizada foi Python com recurso às ferramentas Lex e Yacc. Recorremos também à informação disponibilizada sobre a gramática do TOML no website oficial (<https://github.com/toml-lang/toml/blob/1.0.0/toml.md>).

Ao longo do relatório irá ser abordado em mais detalhe o problema desenvolvido, bem como todo o processo de implementação da gramática responsável pela conversão.

## 2 Análise e Especificação do Problema

### 2.1 Descrição Informal

Desenvolver um conversor que permita transformar um subconjunto Toml em JSON, criada pelo grupo.

### 2.2 Requisitos

Para atingir o objetivo deste projeto, foi necessário estabelecer os requisitos essenciais para sua realização. Isso incluiu a criação de uma sintaxe bem estruturada que permitisse a transição entre estados de forma fácil e intuitiva. Além disso, foi necessário realizar a análise léxica utilizando a ferramenta Lex e a análise sintática com o auxílio do Yacc. Por fim, era preciso desenvolver uma gramática de tradução que fosse independente do contexto.

## 3 Desenho da Solução

Chegamos à etapa de elaboração da solução, na qual será apresentada a proposta de solução desenvolvida pelo grupo de trabalho. Com vista a seguir um modo de desenvolvimento modular, a solução do problema passou por dividir o código a ser executado em módulos. Os módulos *lexico.py* e *criarDicionario.py* executam a análise lexical e sintática dos ficheiros TOML. Os módulos *main.py* e *converter.py* tratam das questões relacionados com a interface de utilizador e processo de criação dos ficheiros JSON.

Em suma, o processo de conversão de TOML para JSON decorre segundo quatro etapas distintas:

1. indicação do caminho do ficheiro que se quer converter;
2. ficheiro é processado pelo LEX;

3. ficheiro é processado pelo YACC;
4. o resultado do processamento do YACC é despejado para um ficheiro JSON.

A seguir iremos explicar em maior detalhe o que faz cada módulo do programa.

### 3.1 Módulo `main.py`

Este módulo corresponde à função principal (`main`) que inicia o programa. Ele lida com a interação do utilizador, recebe o caminho de um ficheiro TOML que será convertido para JSON e chama a função `toml_to_json` do módulo `converter.py` para realizar a conversão.

É imprimida uma interface de utilizador (UI) simples no terminal que solicita o caminho do ficheiro TOML que se deseja converter. É fornecida a opção de sair do programa escrevendo 'q' e é criada uma diretoria chamada `json_files` (caso ainda não exista) para armazenar os arquivos JSON criados durante a conversão. Após converter um ficheiro, o programa continua a solicitar novos caminhos de arquivos TOML até que o utilizador digite 'q' para sair.

```
----- TomlToJson v.1 -----  
  
Insira o caminho do ficheiro toml que quer converter...  
Insira 'q' para fechar programa  
  
-----  
  
--> toml_files(exemplo2.toml
```

Figure 1: Interface com o utilizador

### 3.2 Módulo `converter.py`

Este módulo serve apenas para criar o arquivo JSON final depois de ter sido obtido o resultado do parser. Começamos por lidar com o caminho do arquivo TOML que foi dado pelo utilizador, extraíndo o nome do ficheiro e definindo o nome do arquivo JSON resultante que de seguida é aberto em modo de escrita.

Realizamos a análise léxica do ficheiro TOML usando a função `analisar_ficheiro_toml` do módulo `lexico.py` e de seguida imprime-se os tokens detetados no terminal.

A seguir chama-se a função `cria_dict` do módulo `criaDicionario.py` para criar o dicionário final com base no arquivo TOML; é nesta fase que enviamos o

ficheiro para o processamento por YACC, que devolve o dicionário final mencionado. Verificamos se de facto o dicionário criado possui elementos, e caso seja verdade, imprime uma mensagem de sucesso indicando que a conversão ocorreu sem problemas. O dicionário é então despejado no ficheiro JSON logo de seguida, usando uma formatação adequada.

Em resumo, o módulo trata de ler um arquivo TOML, realiza a análise léxica para obter os tokens, cria um dicionário com base nos tokens detetados e na gramática do módulo *criaDicionario.py*, em seguida escreve esse dicionário em um arquivo JSON final. O módulo lida com a criação do arquivo JSON e a organização correta dos dados convertidos a partir do formato TOML.

### 3.3 Módulo *lexico.py*

O analisador lexical construído permite definir o conjunto de palavras consideradas válidas para a linguagem definida pelo TOML, segundo a documentação consultada para a realização deste trabalho.

Assim sendo, a variável *tokens* define o conjunto de estruturas da linguagem válidas para o formato TOML – cujo código pode ser consultado no Anexo A. Parte dos tokens foi definida como uma expressão regular e outra sob a forma de funções para quando era necessário entrar ou sair de estados. Adicionalmente, foram ainda definidos os caracteres a ignorar (escolhemos apenas ignorar os comentários que não são de maneira nenhuma essenciais) e a ação a ser executada aquando da deteção de erros – sendo impressa uma mensagem com o erro. Algumas das expressões regulares usadas para definir tokens foram, por exemplo, *t.KEY = r'[A-Za-z\_]+'*, que representa uma sequência de letras, dígitos, underscores e hifens. Os tokens mais complexos, como números com underscore (e.g., *'t\_INDIANNUMBER'*) ou chaves aninhadas (e.g., *'t\_DOTTEDKEY'*), também são especificados.

Os estados definidos garantem apenas o reconhecimento de que estamos dentro de um dicionário ou de uma lista de dicionários. Ambos são inclusivos, o que significa que o lexer pode continuar a procurar padrões definidos em estados inclusivos mesmo quando está dentro de um estado principal. A definição destes estados mais os tokens definidos encontra-se discriminada no excerto de código abaixo apresentado, incorporado no módulo *lexico.py*.

```
1 states = (  
2     ('openlist', 'inclusive'),  
3     ('opendict', 'inclusive'),  
4 )  
5  
6  
7 tokens = (  
8     'DICT',  
9     'LISTNAME',  
10    'KEY',  
11    'INTEGER',  
12    'STRING',  
13    'BINARY',
```

```

14     'FLOAT' ,
15     'DATE' ,
16     'EQUALS' ,
17     'COMMENT' ,
18     'BOOLEAN' ,
19     'LBRACKET' ,
20     'RBRACKET' ,
21     'COMMA' ,
22     'DICTNAME' ,
23     'HOURS' ,
24     'DOTTEDKEY' ,
25     'MULTILINESTRING' ,
26     'INDIANNUMBER' ,
27     'SIGNAL' ,
28     'OCTAL' ,
29     'HEXADECIMAL' ,
30     'LCHAVETA' ,
31     'RCHAVETA' ,
32     'OPENLIST' ,
33     'CLOSELIST'
34 )

```

Tendo definido os tokens, segue-se a função *analisar\_ficheiro\_toml* que abre o ficheiro especificado pelo nome fornecido em modo de leitura ('r') e usando a codificação UTF-8. É lido todo o conteúdo do arquivo e armazena na variável *dados*. Estando o lexer criado, processamos os dados, fornecendo-os como argumento usando *lexer.input(dados)*.

Inicia-se um ciclo onde a cada iteração, chama-se *lexer.token()* para obter o próximo token detetado. Verifica-se se o token retornado não é nulo e, caso seja, significa que todos os tokens do arquivo foram processados e o ciclo é interrompido. Caso contrário, o token é impresso usando *print(tok)*.

Esta função basicamente lê um ficheiro TOML, aplica a análise léxica nele usando o lexer que configuramos e imprime os tokens resultantes. Cada token corresponde a uma parte reconhecida do arquivo de acordo com as regras definidas pelo lexer.

O excerto de código abaixo apresentado mostra a definição da função em causa que permite determinar os tokens detetados no ficheiro:

```

1
2     def analisar_ficheiro_toml(filename):
3         with open(filename, 'r', encoding='UTF-8') as file:
4             dados = file.read()
5             lexer.input(dados)
6
7             while True:
8                 tok = lexer.token()
9                 if not tok:
10                     break
11                 print(tok)

```

### 3.4 Módulo `criaDicionario.py`

O analisador sintático encontra-se descrito no módulo *criaDicionario.py*. O grande trabalho executado nesta componente centrou-se na definição da gramática de conversão, que define o conjunto de frases válidas para a linguagem. As produções da gramática de tradução encontram-se apresentadas abaixo.

```
1
2     toml : blocks
3
4     blocks : blocks block
5             | block
6
7     block : DICT LISTNAME RBRACKET content
8            | DICT LISTNAME RBRACKET
9
10    block : DICT DICTNAME RBRACKET content
11           | DICT DICTNAME RBRACKET
12
13    block : keyvaluepair
14
15    block : OPENLIST LISTNAME CLOSELIST content
16           | OPENLIST LISTNAME CLOSELIST
17
18    content : content keyvaluepair
19            | keyvaluepair
20
21    keyvaluepair : KEY EQUALS value
22
23    keyvaluepair : DOTTEDKEY EQUALS value
24
25    value : DATE
26           | HOURS
27           | list
28           | inlinetable
29
30    inlinetable : LCHAVETA elems1 RCHAVETA
31
32    elems1 : elems1 COMMA elem
33            | elem
34
35    elem : KEY EQUALS value
36
37    value : INTEGER
38
39    value : INDIANNUMBER
40
41    value : SIGNAL
42
43    value : OCTAL
44
45    value : HEXADECIMAL
46
47    value : BINARY
48
49    value : FLOAT
50
```

```

51     value : STRING
52
53     value : MULTILINESTRING
54
55     value : BOOLEAN
56
57     list  : LBRACKET elems2 RBRACKET
58
59     elems2 : elems2 COMMA value
60           | value

```

O processamento das funções descritas neste módulo é realizado na função *cria\_dict* que chama o parser criado previamente e o aplica aos dados lidos do ficheiro TOML passado como argumento. É este parser que permite localizar e tratar devidamente as restantes funções definidas no módulo que servem para criar uma estrutura de dicionário aninhado (dicionário que contem outros dicionários lá dentro) que vai ser despejada no ficheiro JSON.

Uma dessas funções é *p\_blocks* que é uma regra de produção do parser que define como construir um bloco no dicionário durante o processo de análise sintática. A função é chamada quando ocorre uma correspondência com a regra blocks.

A regra blocks tem duas formas possíveis:

- blocks : blocks block: significa que há um bloco existente (p[1]) seguido por um novo bloco (p[2]).
- blocks : block: significa que só há um bloco (p[1]) sem um bloco existente anteriormente.

```

1
2     def p_blocks(p):
3         """
4         blocks : blocks block
5                 | block
6         """
7         if len(p) == 2:
8             p[0] = p[1]
9
10        else:
11            new_list = list(p[2].keys())
12            list_temp = new_list[0].split('.')
13
14            if len(list_temp) > 1:
15                temp_dict = p[1]
16
17                for key in list_temp[:-1]:
18
19                    if key not in temp_dict:
20                        temp_dict[key] = {}
21                    temp_dict = temp_dict[key]
22
23            if type(temp_dict) == list:
24                temp_dict.append({list_temp[-1]: p[2][new_list
[0]]})

```



```

25         else:
26             temp_dict[list_temp[-1]] = p[2][new_list[0]]
27             print(p[0], " ", p[1])
28             p[0] = p[1]
29         else:
30             key = list(p[2].keys())[0]
31             if key in p[1]:
32                 for elem in p[2][key]:
33                     p[1][key].append(elem)
34             else:
35                 p[1].update(p[2])
36
37         p[0] = p[1]

```

A primeira verificação é feita para determinar qual forma da regra foi correspondida com base no tamanho da lista `p`. Se o tamanho for igual a 2, isso significa que só há um bloco, caso contrário, há um bloco existente seguido por um novo bloco.

Se houver apenas um bloco (`len(p) == 2`), a função simplesmente atribui esse bloco ao resultado (`p[0] = p[1]`), pois não há bloco existente para ser combinado.

Se houver um bloco existente e um novo bloco (`len(p) > 2`), a função realiza as seguintes etapas:

1. Obtém a lista de chaves do novo bloco (`p[2]`) usando `list(p[2].keys())` e coloca-as em `new_list`;
2. Divide a primeira chave da lista em uma lista separada por ponto (.) usando `split('.')` e a coloca-a em `list_temp`;
3. Verifica se a lista `list_temp` tem mais de um elemento, o que indica que a chave tem uma estrutura aninhada;
4. Se a chave tiver uma estrutura aninhada:
  - (a) Cria uma variável `temp_dict` para armazenar o dicionário atual.
  - (b) Percorre a lista `list_temp` excluindo o último elemento (usando `[:-1]`) e verifica se cada chave está presente em `temp_dict`. Se não estiver, cria um novo dicionário vazio para essa chave.
  - (c) Navega pelo dicionário aninhado até chegar ao último nível, atualizando `temp_dict` a cada iteração.
  - (d) Verifica se o tipo de `temp_dict` é uma lista. Se for, adiciona um novo dicionário contendo a última chave e o valor correspondente ao final da lista. Caso contrário, adiciona a última chave e o valor diretamente ao dicionário.
  - (e) Imprime `p[0]` (que é `None`) e `p[1]` (bloco existente) para fins de depuração.
  - (f) Atribui o bloco existente (`p[1]`) como resultado (`p[0] = p[1]`).
5. Se a chave não tiver uma estrutura aninhada:

- (a) Obtém a primeira chave do novo bloco usando `list(p[2].keys())[0]` e a armazena em `key`.
- (b) Verifica se a chave já está presente no bloco existente (`p[1]`). Se estiver, itera sobre os elementos do novo bloco com a chave correspondente e adiciona-os ao bloco existente. Caso contrário, une o novo bloco com o bloco existente.
- (c) Atribui o bloco existente atualizado como resultado (`p[0] = p[1]`).

Em resumo, a função `p_blocks(p)` combina os blocos existentes com os novos blocos no dicionário durante o processo de análise sintática. Ela lida com a lógica de união de blocos e acomoda a estrutura aninhada de chaves nos dicionários.

Para os outros símbolos não terminais que surgem nesta função foram criadas funções específicas para representar as suas produções. O resto do corpo da execução do analisador sintático encontra-se apresentado no Anexo B a fim de demonstrar a totalidade do raciocínio adotado.

Relativamente à tradução da gramática, esta ocorre, naturalmente, na eventualidade de esta estar presente e de se terem importado os tokens do módulo *lexico.py*. A gramática é uma componente obrigatória do Yacc, pelo que a sua ausência resulta num erro de execução do programa.

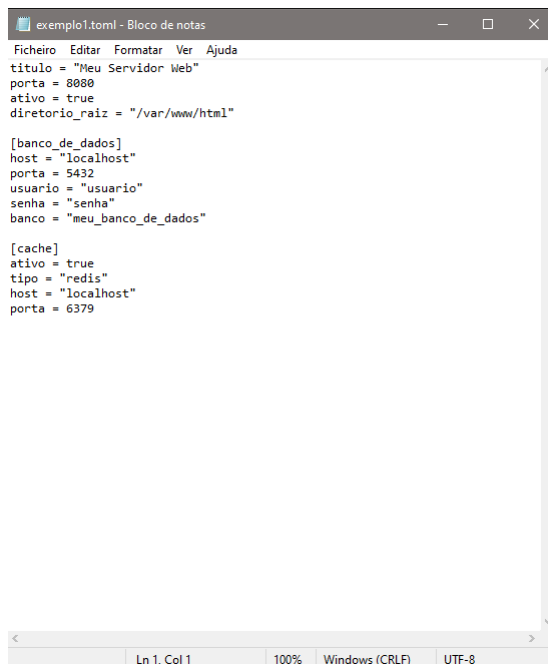
## 4 Testes

A presente secção procura apresentar o conjunto de testes efetuados para a validação da execução correta do programa desenvolvido. A seguir, são apresentados os ficheiros de teste criados, assim como o resultado obtido, e esperado, aquando da sua conversão.

### 4.1 Teste 1

Apresentado na seguinte figura, temos o ficheiro `exemplo1.toml`, que foi criado de origem para testar as funcionalidades mais básicas do nosso programa.

Tendo indicado ao programa que queríamos converter este ficheiro o output é o seguinte:



```
Ficheiro  Editar  Formatar  Ver  Ajuda
titulo = "Meu Servidor Web"
porta = 8080
ativo = true
diretorio_raiz = "/var/www/html"

[banco_de_dados]
host = "localhost"
porta = 5432
usuario = "usuario"
senha = "senha"
banco = "meu_banco_de_dados"

[cache]
ativo = true
tipo = "redis"
host = "localhost"
porta = 6379
```

Figure 2: Ficheiro TOML de exemplo 1

A conversão deste ficheiro de teste foi um sucesso, sendo o resultado obtido demonstrado na figura seguinte:

```
----- TomlToJson v.1 -----
Insira o caminho do ficheiro toml que quer converter...
Insira 'q' para fechar programa

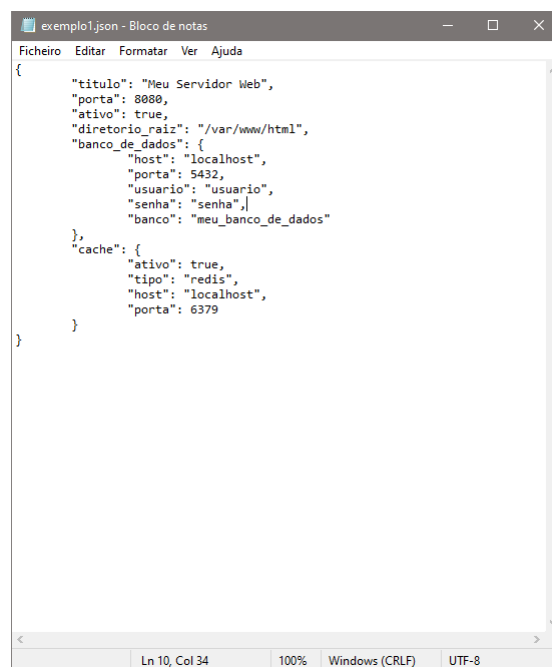
----> toml /Users/wesley/toml

LexToken(KEY, 'titulo', 1, 0)
LexToken(EQUALS, '=', 1, 7)
LexToken(STRING, "Meu Servidor Web", 1, 9)
LexToken(KEY, 'porta', 2, 28)
LexToken(EQUALS, '=', 2, 34)
LexToken(INTEGER, '8080', 2, 36)
LexToken(KEY, 'ativo', 3, 41)
LexToken(EQUALS, '=', 3, 47)
LexToken(BOOLEAN, 'true', 3, 49)
LexToken(KEY, 'diretorio_raiz', 4, 54)
LexToken(EQUALS, '=', 4, 69)
LexToken(STRING, "/var/www/html", 4, 71)
LexToken(DICT, '\n\n', 4, 86)
LexToken(DICTNAME, 'banco_de_dados', 4, 89)
LexToken(RBRACKET, '}', 4, 103)
LexToken(KEY, 'host', 5, 105)
LexToken(EQUALS, '=', 5, 110)
LexToken(STRING, "localhost", 5, 112)
LexToken(KEY, 'porta', 6, 124)
LexToken(EQUALS, '=', 6, 130)
LexToken(INTEGER, '5432', 6, 132)
LexToken(KEY, 'usuario', 7, 137)
LexToken(EQUALS, '=', 7, 145)
LexToken(STRING, "usuario", 7, 147)
LexToken(KEY, 'senha', 8, 157)
LexToken(EQUALS, '=', 8, 163)
LexToken(STRING, "senha", 8, 165)
LexToken(KEY, 'banco', 9, 171)
LexToken(EQUALS, '=', 9, 179)
LexToken(STRING, "meu_banco_de_dados", 9, 181)
LexToken(DICT, '\n\n', 9, 201)
LexToken(DICTNAME, 'cache', 9, 204)
LexToken(RBRACKET, '}', 9, 209)
LexToken(KEY, 'ativo', 10, 211)
LexToken(EQUALS, '=', 10, 217)
LexToken(BOOLEAN, 'true', 10, 219)
LexToken(KEY, 'tipo', 11, 224)
LexToken(EQUALS, '=', 11, 229)
LexToken(STRING, "redis", 11, 231)
LexToken(KEY, 'host', 12, 239)
LexToken(EQUALS, '=', 12, 244)
LexToken(STRING, "localhost", 12, 246)
LexToken(KEY, 'porta', 13, 258)
LexToken(EQUALS, '=', 13, 264)
LexToken(INTEGER, '6379', 13, 266)

Ficheiro convertido com sucesso (ver pasta json_files)

----> |
```

Figure 3: Output para o ficheiro 1

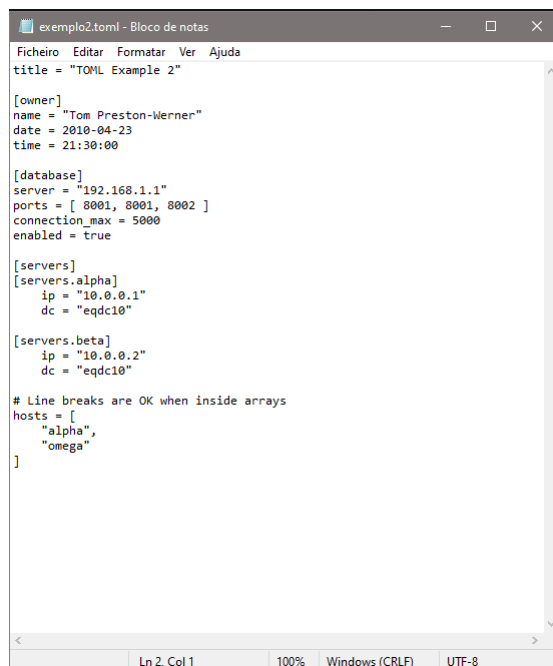


```
{
  "titulo": "Meu Servidor Web",
  "porta": 8080,
  "ativo": true,
  "diretorio_raiz": "/var/www/html",
  "banco_de_dados": {
    "host": "localhost",
    "porta": 5432,
    "usuario": "usuario",
    "senha": "senha",
    "banco": "meu_banco_de_dados"
  },
  "cache": {
    "ativo": true,
    "tipo": "redis",
    "host": "localhost",
    "porta": 6379
  }
}
```

Figure 4: Ficheiro JSON de exemplo 1

## 4.2 Teste 2

No segundo teste decidimos usar o excerto de TOML disponibilizado no enunciado do trabalho prático. Na seguinte figura, conseguimos ver que este ficheiro já apresenta maior complexidade, tendo sido introduzidas chaves com subchaves (ou dotted keys segundo o site oficial do TOML) que resultam num dicionário que é dividido no número de subchaves que existem.



```
exemplo2.toml - Bloco de notas
Ficheiro  Editar  Formatar  Ver  Ajuda
title = "TOML Example 2"

[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"
]
```

Figure 5: Ficheiro TOML de exemplo 2

A conversão resultante deste ficheiro foi novamente um sucesso, sendo o resultado obtido demonstrado na figura seguinte:

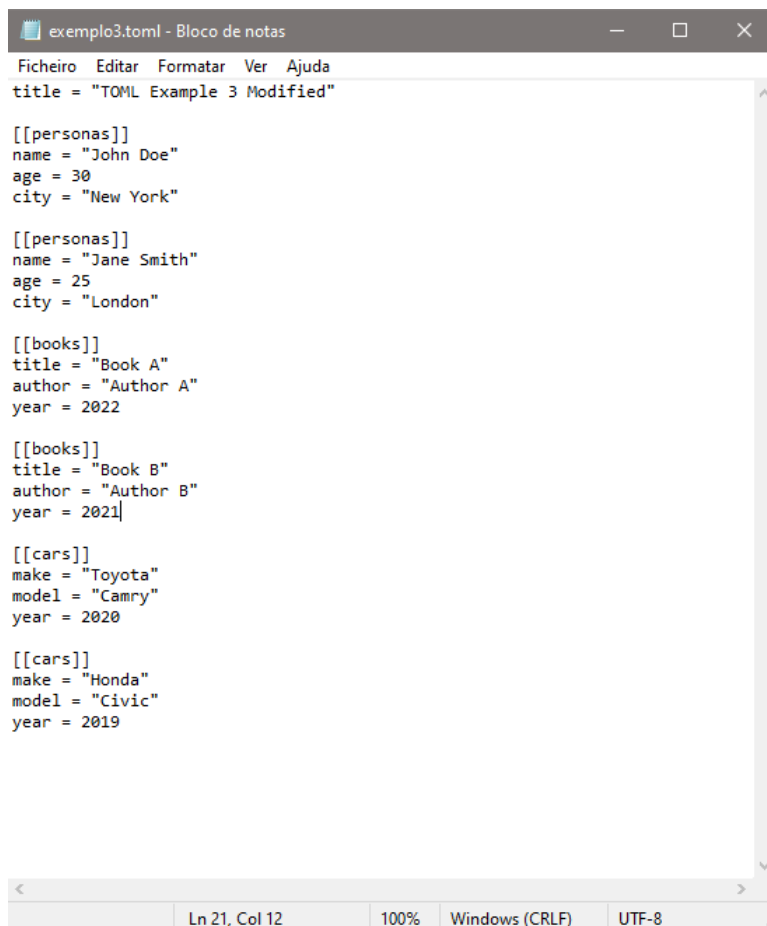


```
{
  "title": "TOML Example 2",
  "owner": {
    "name": "Tom Preston-Werner",
    "date": "2010-04-23",
    "time": "21:30:00"
  },
  "database": {
    "server": "192.168.1.1",
    "ports": [
      8001,
      8001,
      8002
    ],
    "connection_max": 5000,
    "enabled": true
  },
  "servers": {
    "alpha": {
      "ip": "10.0.0.1",
      "dc": "eqdc10"
    },
    "beta": {
      "ip": "10.0.0.2",
      "dc": "eqdc10",
      "hosts": [
        "alpha",
        "omega"
      ]
    }
  }
}
```

Figure 6: Ficheiro JSON de exemplo 2

### 4.3 Teste 3

No terceiro teste decidimos aumentar o nível de complexidade introduzindo listas com dicionários lá dentro. Criado de raiz, o ficheiro `exemplo3.toml` contém três listas de dicionários que serve para testar a forma como o programa lida com este tipo de estrutura.



```
exemplo3.toml - Bloco de notas
Ficheiro Editar Formatar Ver Ajuda
title = "TOML Example 3 Modified"

[[personas]]
name = "John Doe"
age = 30
city = "New York"

[[personas]]
name = "Jane Smith"
age = 25
city = "London"

[[books]]
title = "Book A"
author = "Author A"
year = 2022

[[books]]
title = "Book B"
author = "Author B"
year = 2021

[[cars]]
make = "Toyota"
model = "Camry"
year = 2020

[[cars]]
make = "Honda"
model = "Civic"
year = 2019
```

Ln 21, Col 12 100% Windows (CRLF) UTF-8

Figure 7: Ficheiro TOML de exemplo 3

Tendo convertido este ficheiro, o resultado obtido é demonstrado na figura seguinte:



```
{
  "title": "TOML Example 3 Modified",
  "personas": [
    {
      "name": "John Doe",
      "age": 30,
      "city": "New York"
    },
    {
      "name": "Jane Smith",
      "age": 25,
      "city": "London"
    }
  ],
  "books": [
    {
      "title": "Book A",
      "author": "Author A",
      "year": 2022
    },
    {
      "title": "Book B",
      "author": "Author B",
      "year": 2021
    }
  ],
  "cars": [
    {
      "make": "Toyota",
      "model": "Camry",
      "year": 2020
    },
    {
      "make": "Honda",
      "model": "Civic",
      "year": 2019
    }
  ]
}
```

Ln 17, Col 43    100%    Windows (CRLF)    UTF-8

Figure 8: Ficheiro JSON de exemplo 3

## 5 Conclusão

Dado por concluído o trabalho prático, consideramos importante realizar uma análise crítica, e ainda, realçar os pontos positivos e negativos do trabalho realizado.

No que diz respeito a pontos positivos, consideramos que o programa cumpre com os requisitos, sendo capaz de fazer uma conversão correta de toml-json e usando uma gramática que não apresenta qualquer tipo de conflito, sendo simples e de fácil entendimento.

Porém, o grupo considera que é possível que a gramática não tenha em conta todos os casos possíveis da sintaxe toml-json e que possivelmente podia ser simplificada em alguns aspetos.

Em conclusão, o grupo está satisfeito com o trabalho desenvolvido, dado que este cumpre com os requisitos pedidos. Além disso, a realização deste trabalho ajudou-nos a consolidar a matéria lecionada relativamente à criação e manipulação de gramáticas tradutoras e independentes de contexto.

## 6 Anexos

### 6.1 Anexo A

```
1
2     from ply import lex
3
4     states = (
5         ('openlist', 'inclusive'),
6         ('opendict', 'inclusive'),
7     )
8
9     tokens = (
10         'DICT',
11         'LISTNAME',
12         'KEY',
13         'INTEGER',
14         'STRING',
15         'BINARY',
16         'FLOAT',
17         'DATE',
18         'EQUALS',
19         'COMMENT',
20         'BOOLEAN',
21         'LBRACKET',
22         'RBRACKET',
23         'COMMA',
24         'DICTNAME',
25         'HOURS',
26         'DOTTEDKEY',
27         'MULTILINESTRING',
28         'INDIANNUMBER',
29         'SIGNAL',
30         'OCTAL',
31         'HEXADECIMAL',
32         'LCHAVETA',
33         'RCHAVETA',
34         'OPENLIST',
35         'CLOSELIST',
36     )
37
38
39
40 t_KEY = r'[A-Za-z_\-]+'
41 t_DOTTEDKEY = r'[A-Za-z_\-]+(\.[A-Za-z_\-]+)+'
42
43 t_INTEGER = r'\d+'
44 t_INDIANNUMBER = r'\d+(-\d+)+'
45 t_SIGNAL = r'[+-]\d+'
46 t_BINARY = r'0b[01]+'
47 t_OCTAL = r'0o[0-7]+'
48 t_HEXADECIMAL = r'0x[0-9a-fA-F]+'
49
50 t_STRING = r'\".*?\"'
51 t_FLOAT = r'\d+.\d+'
52
53 t_DATE = r'\d{4}\-\d{2}\-\d{2}'
```

```

54 t_HOURS = r'\d{2}\:\d{2}\:\d{2}'
55
56 t_EQUALS = r'\='
57 t_ignore.COMMENT = r'\#.*'
58 t_BOOLEAN = r'(?i)(true|false)'
59
60 t_LBRACKET = r'\['
61 t_RBRACKET = r'\]'
62 t_COMMA = r','
63 t_LCHAVETA = r'\{'
64 t_RCHAVETA = r'\}'
65
66
67 def t_OPENLIST(t):
68     r'\n+\[['
69     t.lexer.begin('openlist')
70     return t
71
72 def t_openlist_LISTNAME(t):
73     r'[A-Za-z_\.\d-9]+'
74     return t
75
76 def t_openlist_DICTNAME(t):
77     r'\w+(\.\w+)*'
78     return t
79
80 def t_openlist_CLOSELIST(t):
81     r'\]\]'
82     t.lexer.begin('INITIAL')
83     return t
84
85
86 def t_DICT(t):
87     r'\n+\s*[['
88     t.lexer.begin('opendict')
89     return t
90
91 def t_opendict_DICTNAME(t):
92     r'\w+(\.\w+)*'
93     return t
94
95 def t_opendict_RBRACKET(t):
96     r'\]'
97     t.lexer.begin('INITIAL')
98     return t
99
100
101 def t_newline(t):
102     r'\n+'
103     t.lexer.lineno += len(t.value)
104
105 def t_MULTILINESTRING(t):
106     r"""["']\n\s*["']"""
107     return t
108
109 t_ignore = ' \t'
110

```

```

111 def t_error(t):
112     print(f"Carácter ilegal {t.value[0]}")
113     t.lexer.skip(1)
114
115
116 lexer = lex.lex()
117
118
119 def analisar_ficheiro_toml(filename):
120     with open(filename, 'r', encoding='UTF-8') as file:
121         dados = file.read()
122         lexer.input(dados)
123
124         while True:
125             tok = lexer.token()
126             if not tok:
127                 break
128             print(tok)

```

## 6.2 Anexo B

```

1
2 import ply.yacc as yacc
3
4 from lexico import tokens
5
6
7 def p_toml(p):
8     """
9     toml : blocks
10     """
11     p[0] = p[1]
12
13
14 def p_blocks(p):
15     """
16     blocks : blocks block
17             | block
18     """
19     if len(p) == 2:
20         p[0] = p[1]
21
22     else:
23         new_list = list(p[2].keys())
24         list_temp = new_list[0].split('.')
25
26         if len(list_temp) > 1:
27             temp_dict = p[1]
28
29             for key in list_temp[:-1]:
30
31                 if key not in temp_dict:
32                     temp_dict[key] = {}
33                 temp_dict = temp_dict[key]
34
35         if type(temp_dict) == list:

```

```

36         temp_dict.append({list_temp[-1]: p[2][new_list
[0]]})
37     else:
38         temp_dict[list_temp[-1]] = p[2][new_list[0]]
39         print(p[0], " ", p[1])
40         p[0] = p[1]
41     else:
42         key = list(p[2].keys())[0]
43         if key in p[1]:
44             for elem in p[2][key]:
45                 p[1][key].append(elem)
46         else:
47             p[1].update(p[2])
48
49     p[0] = p[1]
50
51
52 def p_block(p):
53     """
54     block : DICT LISTNAME RBRACKET content
55           | DICT LISTNAME RBRACKET
56     """
57     if len(p) == 5:
58         p[0] = {p[2]: p[4]}
59     else:
60         p[0] = {p[2]: {}}
61
62
63 def p_block_dictname(p):
64     """
65     block : DICT DICTNAME RBRACKET content
66           | DICT DICTNAME RBRACKET
67     """
68     if len(p) == 5:
69         p[0] = {p[2]: p[4]}
70     else:
71         p[0] = {p[2]: {}}
72
73
74 def p_block_keyvaluepairs(p):
75     """
76     block : keyvaluepair
77     """
78     p[0] = p[1]
79
80
81 def p_block_openlist_listname(p):
82     """
83     block : OPENLIST LISTNAME CLOSELIST content
84           | OPENLIST LISTNAME CLOSELIST
85     """
86     if len(p) == 5:
87         p[0] = {p[2]: [p[4]]}
88     else:
89         p[0] = {p[2]: []}
90
91

```

```

92
93 def p_content(p):
94     """
95     content : content keyvaluepair
96             | keyvaluepair
97     """
98     if len(p) == 2:
99         p[0] = p[1]
100     else:
101         p[1].update(p[2])
102         p[0] = p[1]
103
104
105 def p_keyvaluepair_KEY(p):
106     """
107     keyvaluepair : KEY EQUALS value
108     """
109     p[0] = {p[1]: p[3]}
110
111
112 def p_keyValuePair_DOTTEDKEY(p):
113     """
114     keyvaluepair : DOTTEDKEY EQUALS value
115     """
116     list = p[1].split(".")
117     dict = p[3]
118
119     for key in reversed(list):
120         dict = {key: dict}
121
122     p[0] = dict
123
124
125 def p_value(p):
126     """
127     value : DATE
128           | HOURS
129           | list
130           | inlinetable
131     """
132     p[0] = p[1]
133
134
135 def p_inlinetable(p):
136     """
137     inlinetable : LCHAVETA elems1 RCHAVETA
138     """
139     p[0] = p[2]
140
141
142 def p_inlinetable_KEYS(p):
143     """
144     elems1 : elems1 COMMA elem
145            | elem
146     """
147     if len(p) == 4:
148         p[1].update(p[3])

```

```

149         p[0] = p[1]
150
151     else:
152         p[0] = p[1]
153
154
155     def p_inlinetable_KEY(p):
156         """
157         elem : KEY EQUALS value
158         """
159         p[0] = {p[1]: p[3]}
160
161
162     def p_INTEGER(p):
163         """
164         value : INTEGER
165         """
166         p[0] = int(p[1])
167
168
169     def p_INDIANNUMBER(p):
170         """
171         value : INDIANNUMBER
172         """
173         lista = p[1].split("-")
174         string = ""
175
176         for elem in lista:
177             string += elem
178
179         p[0] = int(string)
180
181
182     def p_SIGNAL(p):
183         """
184         value : SIGNAL
185         """
186         p[0] = int(p[1])
187
188
189     def p_OCTAL(p):
190         """
191         value : OCTAL
192         """
193         p[0] = int(p[1], 8)
194
195
196     def p_HEXADECIMAL(p):
197         """
198         value : HEXADECIMAL
199         """
200         p[0] = int(p[1], 16)
201
202
203     def p_BINARY(p):
204         """
205         value : BINARY

```



```

206     """
207     p[0] = int(p[1], 2)
208
209
210 def p_FLOAT(p):
211     """
212     value : FLOAT
213     """
214     p[0] = float(p[1])
215
216
217 def p_STRING(p):
218     """
219     value : STRING
220     """
221
222     p[0] = p[1].split("\")[1]
223
224
225 def p_MULTILINESTRING(p):
226     """
227     value : MULTILINESTRING
228     """
229     #p[0] = p[1].split("\n\n")[1]
230     p[0] = p[1]
231
232
233 def p_BOOLEAN(p):
234     """
235     value : BOOLEAN
236     """
237     p[0] = bool(p[1])
238
239
240 def p_list(p):
241     """
242     list : LBRACKET elems2 RBRACKET
243     """
244
245     p[0] = p[2]
246
247
248 def p_elems2(p):
249     """
250     elems2 : elems2 COMMA value
251             | value
252     """
253     if len(p) == 2:
254         p[0] = [p[1]]
255
256     else:
257         p[1].append(p[3])
258         p[0] = p[1]
259
260
261 def p_error(p):
262     print(f"Erro de sintaxe na linha {p.lineno}: Token inválido")

```

```
263 {p.value}")
264
265 parser = yacc.yacc()
266
267
268 def cria_dict(toml_file_path):
269     with open(toml_file_path, 'r', encoding='UTF-8') as
toml_file:
270         data = toml_file.read()
271         parser.parse(data)
272
273     return parser.parse(data)
```