Projeto de LFA

TableHandler

Universidade de Aveiro

Cristiano Vagos (65169), Paulo Gil (76361)

Projeto de LFA

Linguagens Formais e Autómatos

Universidade de Aveiro

Cristiano Vagos (65169), Paulo Gil (76361)
cristianovagos@ua.pt, paulogil@ua.pt

31-06-2017

Conteúdo

Capítulo 1

Introdução

No âmbito da disciplina de Linguagens Formais e Autómatos foi proposta a realização de um projeto de forma a consolidarmos e colocarmos em prática os conhecimentos adquiridos durante o semestre.

De entre as várias opções de projetos (propostas pelos professores ou incentivando a nossa criatividade) decidimos escolher uma proposta indicada pelos professores no enunciado do projeto: uma linguagem para manipulação de tabelas, cujo objetivo final é gerar código em Java.

Decidimos chamar à nossa linguagem "TableHandler", porque de facto é o que fazemos internamente: manipulamos tabelas. No entanto a nossa linguagem não se resume apenas à manipulação de tabelas, pelo que decidimos adicionar outras operações, disponíveis em todas as linguagens de programação.

Mais detalhes sobre a implementação e o funcionamento desta linguagem podem ser vistos mais à frente, neste relatório.

Capítulo 2

Descrição da Linguagem

A linguagem denomina-se de TableHandler, e destina-se não só à manipulação de operações com tabelas mas também às operações básicas de uma linguagem de programação, tais como operações aritméticas, declaração de variáveis, uma operação de impressão para a consola, e ainda conta com uma expressão de condição. As operações com tabelas fazem-se de uma forma um pouco semelhante a comandos SQL. O delimitador de instruções escolhido é o ponto e vírgula ';'. É usado para facilitar a separação de instruções.

2.1 Tipos de dados

TableHandler conta com os seguintes tipos de dados:

- int
- double
- string
- boolean
- table

2.2 Declaração

A declaração de variáveis faz-se declarando o tipo de dados seguido do nome da variável. A variável respeita a seguinte regra do lexer:

ID : [a–zA–Za–9_]+

2.3 Atribuição

A atribuição é feita apenas depois de a variável ter sido declarada. O valor da atribuição tem que respeitar o tipo de dados introduzido.

2.4 Comando print

O comando print é usado para retornar o output para a consola daquilo que está a acontecer no decorrer do programa. Podem se imprimir expressões aritméticas, expressões booleanas e variáveis (caso estejam declaradas).

```
print "Preciso de ter pelo menos 10 neste projeto";
print 2+3;
print varDummy;
```

2.5 Instrução if

A instrução if segue o padrão normal. É avaliada uma condição, caso a condição seja cumprida, segue-se uma lista (teóricamente infinita) de instruções *else if* e por último uma instrução *else*.

```
if(condition) {
    statementList;
} else if(otherCondition) {
    statementList;
} else {
    statementList;
}
```

2.6 Expressões de tabelas

As expressões para a manipulação são acompanhadas de um prefixo que as identifica.

`table` (COMMAND)

Comandos disponíveis para a manipulação de tabelas:

- **read FILE to VAR** - lê um ficheiro *csv* e inicializa a variável VAR
- **add row STRING to VAR** - insere uma linha no formato *csv* na tabela da variável VAR
- **add row STRING at INTEGER to VAR** - insere uma linha no *index* INTEGER da tabela da variável VAR

- **get value(ROW,COL) from VAR** - devolve o valor da linha pela coluna da tabela associada à variável VAR
- **insert into VAR value=EXPR at cell(ROW,COL)** - insere numa célula de uma tabela o valor de uma expressão
- **clear row at INTEGER from VAR** - limpa os dados de uma linha específica de uma tabela associada à variável VAR
- **remove row at INTEGER from VAR** - elimina uma linha específica numa tabela
- **add column STRING to VAR** - insere uma coluna no formato *csv* na tabela associada à variável VAR
- **add column STRING at INTEGER to VAR** - insere uma coluna no *index* INTEGER da tabela da variável VAR
- **remove column at INTEGER from VAR** - elimina uma coluna específica numa tabela
- **clear field(ROW,COL) from VAR** - limpa uma célula específica da tabela associada à variável VAR
- **get row size from VAR** - devolve o número de linhas de uma tabela
- **get column size from VAR** - devolve o número de colunas de uma tabela
- **get unique column from VAR at INTEGER** - devolve uma coluna específica, sem repetição de elementos, de uma tabela
- **get column from VAR at INTEGER header= boolean** - devolve uma coluna específica de uma tabela, com ou sem o cabeçalho
- **get row from VAR at INTEGER** - devolve uma linha específica de uma tabela
- **get index from VAR of value=STRING** - devolve o *index* de uma tabela correspondente a uma STRING
- **col-subtable from VAR start=INTEGER end=INTEGER** - devolve uma sub-tabela entre as colunas *start=INTEGER* e *end=INTEGER* de uma tabela associada à variável VAR
- **col-subtable from VAR start=INTEGER** - devolve uma sub-tabela entre a coluna *start=INTEGER* e o fim, de uma tabela associada à variável VAR
- **row-subtable from VAR start=INTEGER end=INTEGER** - devolve uma sub-tabela entre as linhas *start=INTEGER* e *end=INTEGER* de uma tabela associada à variável VAR

- **row-subtable from VAR start=INTEGER** - devolve uma sub-tabela entre a linha *start=INTEGER* e o fim, de uma tabela associada à variável VAR
- **add VAR with VAR** - adiciona duas tabelas
- **subtract VAR with VAR** - subtrai duas tabelas
- **sort VAR** - ordena por ordem ascendente uma tabela associada à variável VAR
- **sort descendent VAR** - ordena por ordem descendente uma tabela associada à variável VAR
- **VAR equals VAR** - verifica se duas tabelas são iguais
- **export VAR to FILE** - exporta uma tabela associada a uma variável VAR para um ficheiro

  print VAR - imprime uma tabela
- **print first INTEGER lines of VAR** - imprime as primeiras INTEGER linhas de uma tabela
- **print last INTEGER lines of VAR** - imprime as últimas INTEGER linhas de uma tabela

2.7 Comentários

A linguagem suporta também comentários. Estes podem ser *single line* ou *multi-line*.

```
//Single line comment

/*Multi
 *line
 *comment
 */
```

Capítulo 3

Implementação e Arquitectura

Vamos agora proceder à apresentação da nossa implementação e arquitectura da nossa linguagem. Aqui reunimos a informação sobre a forma como pensámos e implementámos a linguagem e o papel que cada ficheiro tem na execução.

Aconselhamos (como sempre) a visualização do código para melhor compreensão da nossa abordagem ao problema.

3.1 Manipulação de Tabelas - ficheiro Table.java

Antes da criação da linguagem propriamente dita, o primeiro passo foi de facto criar uma classe em Java que nos permitisse manipular as tabelas. Nesta classe foram criadas várias funções que permitem criar, manipular e obter informação acerca de tabelas, a partir de uma estrutura de dados (ArrayList). Internamente consideramos uma tabela uma lista de listas (ArrayList de ArrayList). Tendo em conta a facilidade de compreensão e implementação da nossa estrutura de dados, achámos esta a melhor forma de o fazer.

A partir desta classe torna-se possível criar uma tabela através de um ficheiro CSV (Comma Separated Values), ficheiro esse que é analisado (parse) e inserido na estrutura de dados referida anteriormente. É a única forma de se poder criar uma tabela tendo em conta conteúdo vindo do exterior. É também possível criar uma tabela a partir de uma outra tabela já existente, derivada de uma operação anterior.

Esta classe está totalmente documentada, tendo toda a informação sobre as funções nela implementadas, o que é necessário para as funções serem executadas corretamente, o que devolvem, e usa programação defensiva de forma a poder lidar com possíveis erros (excepções e asserções).

As operações implementadas são as seguintes:

- Criação de Tabelas

  - A partir de um ficheiro CSV (Comma Separated Values)
  - A partir de uma outra Tabela já criada (objeto ou via estrutura de dados)
- Modificar, alterar, obter e limpar campos da tabela
- Adicionar, remover, obter e limpar linhas/colunas
- Obter número de linhas/colunas
- Obter coluna com valores únicos (não repetidos)
- Obter índice com base no cabeçalho
- Obter sub-tabelas entre linhas ou colunas
- Soma/subtração de tabelas
- Ordenação (ascendente/descendente) da tabela
- Comparação de tabelas
- Exportação para ficheiro CSV (Comma Separated Values)
- Impressão da tabela no terminal
- Impressão das primeiras/últimas 'n' linhas
- Obter valor máximo/mínimo, soma, média de uma linha ou coluna

Estas operações são fulcrais ao funcionamento em geral da nossa linguagem, após a tradução para Java. Este foi o primeiro objetivo, para que depois pudéssemos criar a linguagem sobre estas operações, e com a sintaxe correta.

3.2 Gramática - ficheiro TableHandler.g4

Depois da criação do ficheiro que cria e manipula as tabelas, é altura de tratar da criação e interpretação da nossa linguagem. Como dito acima na "Descrição da Linguagem", optámos por seguir uma abordagem "à la"SQL, e como tal, este ficheiro é o responsável pelo reconhecimento dos tokens, bem como a análise lexical e sintática da nossa linguagem. O funcionamento e interpretação da linguagem foi feito conforme descrito anteriormente no capítulo "Descrição da Linguagem".

3.3 Compilador - ficheiro Compiler.java

Este ficheiro é o nosso compilador: um Listener responsável pelo registo das variáveis já utilizadas numa estrutura de dados (HashMap), e pela geração de código em Java, utilizando a ferramenta String Template, do mesmo criador do ANTLR4, Terrence Parr.

Se tudo correu bem, este ficheiro imprime no terminal o código final traduzido para Java, e cria um outro ficheiro, ao qual chamámos de "TableOutput.java", que é o resultado final da tradução. Este ficheiro passa a estar pronto a ser compilado e executado.

3.4 Classe Variable - ficheiro Variable.java

Esta classe foi criada para auxiliar o Compilador.

Como utilizamos na nossa linguagem vários tipos de dados, é necessário especificar estes dados (tipo da variável e nome) no HashMap em conjunto com o seu valor, daí a necessidade de haver uma classe extra que fosse capaz de guardar não só o nome da variável como também o seu tipo de dados, para a sua distinção e inserção no HashMap.

3.5 Tratamento de Erros

Para o tratamento de erros, decidimos utilizar o módulo realizado pelo professor Miguel Oliveira e Silva, abordado e discutido nas aulas (ficheiros ErrorHandling.java e ErrorHandlingListener.java).

Achámos por bem utilizar este módulo em detrimento do ConsoleErrorListener, o Listener de erros nativo do ANTLR4, de forma a podermos personalizar as nossas mensagens de erro, bem como podermos mostrar ao utilizador onde os erros estão, de forma a que possa fazer debug facilmente.

3.6 Análise Semântica - ficheiro THSemanticCheck.java

Este ficheiro é o responsável pela análise semântica verificando erros no código TableHandler, e em caso de detecção dos mesmos, a impressão do tipo de erro, e sua localização utilizando o módulo de Tratamento de Erros abordado acima, de forma a informar o utilizador.

3.7 Execução da linguagem - ficheiro TableHandlerRun.java

Este ficheiro é o ficheiro principal, que irá tratar das análises necessárias à execução do tradutor de código entre a linguagem origem - TableHandler, e a linguagem destino - Java.

Colocou-se um nome diferente no ficheiro de modo a que não seja possível a alteração deste código via ferramenta *antlr-main*.

3.8 Outros ficheiros

Para além dos ficheiros aqui mencionados e dos ficheiros criados pelo ANTLR4 decidimos incluir scripts em bash que facilitem a compilação e execução dos ficheiros disponibilizados.

- **build** - compilação dos ficheiros necessários à execução
- **run** - execução (pede qual o ficheiro na linguagem TableHandler para ser gerado)
- **clean** - limpar os ficheiros gerados

De forma a agrupar todos os templates usados por nós na geração do código Java, criámos um ficheiro "templates.stg". Também incluímos um ficheiro CSV "example.csv", e dois ficheiros de exemplo na linguagem TableHandler "exemplo.txt"e "exemplo1.txt"para teste das funcionalidades.

Lembramos que no fim da execução, se tudo correu bem é impresso no terminal o código traduzido para Java e gerado um ficheiro "TableOutput.java", que contém o código final, pronto a ser compilado e executado.

Capítulo 4

Conclusão

Acreditamos que os objetivos do projeto foram cumpridos. Este projeto sem dúvida que nos fez utilizar os conceitos abordados nas aulas, e colocar em prática o que foi leccionado de forma a podermos criar a nossa própria linguagem de programação e saber de facto como funciona um compilador, ainda que gere código numa linguagem com a qual estamos familiarizados.

Quando confrontados com a possibilidade de sermos criativos na criação da linguagem revelámos algumas dificuldades, por exemplo na soma e subtracção de tabelas, bem como noutras operações na qual se exigia alguma criatividade da nossa parte na implementação destas operações, pois não há uma forma "correta"de o fazer, no entanto estas dificuldades foram resolvidas rapidamente após discussão com os professores.

Outra dificuldade prendeu-se com o facto de não termos conseguido acabar o trabalho a tempo da primeira entrega. Não queríamos ser prejudicados pelo trabalho inicial, portanto decidimos entregar agora e ter algo decente para apresentar do que ter algo incompleto e que nos prejudique (como era o caso do trabalho inicial).

Tentámos implementar todos os conceitos abordados nas aulas. Listeners, Visitors, geração de código, criação da gramática, uso de *callbacks*, e um pouco de programação Java à mistura. Tudo se tornou mais fácil podendo usar esta linguagem de programação.

Concluíndo, este projeto permitiu-nos adquirir conhecimentos acerca da tecnologia aliada à compilação e análise léxica, sintática e semântica das linguagens de programação, aliado aos objetivos da unidade curricular.