

Compiladores

Trabalho Prático

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

2018–2019, 2^o semestre

1 Introdução

Um compilador pode ser encarado como sendo um *tradutor* da linguagem fonte (i.e. da linguagem a compilar) para uma linguagem destino. A linguagem destino pode ser próxima da linguagem fonte, ou muito distante (por exemplo, *assembly* ou linguagem máquina). Neste processo de *tradução* o compilador deve não só garantir a validade sintáctica do programa, como também a sua correcção semântica (i.e. uma utilização com significado das instruções da linguagem).

2 Objectivos

O trabalho a desenvolver deve envolver, pelo menos, duas linguagens: uma para um compilador (que, em princípio, será a principal linguagem do trabalho) e outra para ler informação estruturada (por exemplo, um ficheiro de configuração, ou uma linguagem de especificação complementar à linguagem principal).

O desenvolvimento do compilador deve envolver, tanto quanto possível, todas as fases de construção de linguagens de programação:

1. Concepção e definição de uma linguagem de programação (sintaxe e semântica).
2. Implementação em ANTLR4 da análise léxica e da análise sintáctica de um compilador para a linguagem;
3. Definição das regras semânticas a aplicar à linguagem, e sua implementação no contexto do ponto anterior.
4. Escrita de documento que descreva a linguagem (instruções existentes e o seu significado; exemplos de programas; etc.).

5. Escolha criteriosa de uma linguagem destino, onde se possa implementar a síntese (*backend*) do compilador.
6. Definição dos padrões de geração de código para as instruções da linguagem.
7. Concretização completa do compilador.

3 Temas

O tema para a linguagem de programação a desenvolver pode ser proposto pelos alunos (sendo que, neste caso, antes de ser aceite terá de passar pelo “crivo” do docente das práticas). Alternativamente, pode ser escolhido um tema da seguinte lista:

1. Linguagem para manipulação de figuras gráficas (desenho, composição, animação, ...).
2. Linguagem para manipulação de tabelas (adaptação do problema do bloco 3 para um compilador).
3. Linguagem para manipulação de imagens (processamento de imagem, zoom, *crop*, detecção de contornos, extracção de caras de imagens, ...). Gerar código em `OpenCV`, ou noutra biblioteca/linguagem que suporte minimamente as funcionalidades pretendidas.
4. Linguagem para definição e manipulação de robots¹.
5. Linguagem para análise dimensional (física). A especificação de dimensões e unidades (metros, segundos, nano, micro, ...) pode ser feita numa linguagem separada, sendo o compilador aplicável a uma linguagem de programação que se aproxime qb. de uma linguagem de uso genérico (ver secção A).
6. Linguagem para criptografia (ver secção B).

As escolhas a tomar no desenvolvimento das linguagens e respectivas gramáticas são livres (e sujeitas a avaliação). Sugere-se a implementação de algumas das seguintes operações:

- Definição de variáveis;
- Operações interactivas com o utilizador;
- Definição de expressões que definam uma *álgebra* sobre elementos da linguagem (números, figuras, tabelas, imagens, ...);
- Instruções iterativas;

¹Para uma melhor compreensão deste problema deve falar com o Prof. Artur Pereira.

- Expressões booleanas (predicados) e instruções condicionais;
- Funções.

Para além do documento que descreve as linguagens desenvolvidas, tem de fazer parte da entrega do trabalho um conjunto adequado de programas (funcionais) de exemplo das linguagens.

4 Grupos

O trabalho deve ser realizado por grupos de 4/5 elementos. Os grupos devem ser formados preferencialmente por elementos da mesma turma. Poderão ser consideradas excepções desde que previamente sancionadas pelos docentes envolvidos. Serão criados projectos na plataforma `code.ua` para suporte da atividade dos grupos, sendo o código colocado num repositório em `git`. Os projectos serão criados pelo docente durante as aulas práticas. Alerta-se desde já que as actualizações feitas ao repositório devem ser executadas por quem desenvolve o código, usando mensagens adequadas. Serão mal toleradas situações do tipo “tive que pedir ao meu colega para o fazer”.

A entrega do trabalho será feita recorrendo a estes repositórios.

5 Avaliação

A avaliação terá em consideração o trabalho desenvolvido pelo grupo. Serão realizadas reuniões com cada grupo com a presença de dois docentes onde se fará uma avaliação prática com o trabalho realizado (onde se espera que o grupo demonstre o trabalho feito e responda a eventuais dúvidas e questões).

No relatório escrito do projecto, é obrigatório incluir uma secção “Contribuições dos autores” onde se descrevem resumidamente as contribuições de cada elemento do grupo e se avalia a percentagem de trabalho de cada um. Esta auto-avaliação poderá afetar a ponderação da nota a atribuir a cada elemento.

Na avaliação vão pesar a qualidade da solução desenvolvida e os objectivos parcelares por ela cobertos. Fazem parte destes objectivos os seguintes pontos:

1. Concepção da linguagem. A simplicidade e expressividade da linguagem definida serão aspectos a valorizar.
2. Gramáticas desenvolvidas.
3. Análise semântica.
4. Gestão de erros.
5. Legibilidade do código e documentação.

6. Geração de código (será valorizado o uso de uma linguagem destino mais “baixo nível”).

O grau de ambição do trabalho desenvolvido, confrontado com os resultados obtidos, será também tido em conta.

6 Execução do trabalho e prazos de entrega

As aulas práticas que decorrem até ao fim do semestre serão dedicadas ao trabalho prático. No entanto, é expectável que não sejam suficientes (principalmente aos grupos que pretendem uma melhor classificação). Espera-se por isso que uma parte do trabalho seja feita fora das aulas.

O prazo limite de entrega do trabalho será uma semana antes da data do exame da época de exames respectiva (i.e., 18 de junho na época normal e 4 de julho para recurso). Lembra-se que, como definido no guião da unidade curricular no início do semestre, o trabalho prático tem uma única entrega (ou para a época normal ou para recurso).

A Linguagem para análise dimensional

Pretende-se estender o sistema de tipos de uma linguagem de programação (tanto quanto baste, de uso geral) com a possibilidade de definir dimensões distintas (e interoperáveis) a expressões numéricas (e como tal, a variáveis e outras entidades com tipos da linguagem). Por exemplo, poder definir tipos de dados distância e tempo (expressáveis, por exemplo, com unidades metro [m] e segundo [s]) e poder definir um novo tipo de dados velocidade como sendo distância/tempo [m/s]. O sistema de tipos da linguagem não só deve permitir álgebra sobre dimensões existentes, com validar a respectiva correcção (atribuir e/ou somar distância com tempo não deve ser permitido, muito embora ambos sejam números).

Muito embora seja na área da física que as questões dimensionais sejam mais conhecidas, na realidade todos os programadores são confrontados com este tipo de problemas. Por exemplo, sempre que definimos uma variável inteira para percorrer índices de um array com um histograma de idades, não faz sentido que ela possa ser misturada com outra variável, também inteira, mas que represente o número de porcos de uma quinta.

Seria assim interessante haver soluções práticas para este tipo de problemas (o uso do sistema de tipos estáticos de linguagens de programação seria uma solução com grandes potencialidades).

Poder-se-ia ter uma linguagem de especificação (interpretada) para definir dimensões e unidades, que definiria novos tipos de dados (numéricos) que poderiam ser utilizados com segurança na linguagem genérica (compilada), evitando dessa forma, erros dimensionais nesse código.

Seria necessário aplicar uma álgebra dimensional (neste projecto, aplicável apenas a números inteiros e reais) com as seguintes regras:

- somas e subtrações apenas para a mesma dimensão (ex: $1s + 50ms$, para a dimensão tempo);
- multiplicações e divisões aplicáveis a expressões dimensionais a gerar outra dimensão (ex: $1m/10s$, a gerar a dimensão m/s);
- multiplicações e divisões por expressões adimensionais a manter a mesma dimensão (ex: $5s \cdot 100$).

(Para uma melhor compreensão deste problema falar com o Prof. Miguel Oliveira e Silva.)

B Linguagem para criptografia

As cifras são funções que transformam dados ditos em claro em algo ininteligível (criptograma), e vice-versa, usando para o efeito um algoritmo bem conhecido e um valor secreto, designado por chave. Os dados em claro, os criptogramas e as chaves são tratadas pelo algoritmo como conjuntos de bits com um comprimento fixo. Estes conjuntos de bits são processados pelo algoritmo usando operações que operam sobre blocos de bits de várias dimensões. Essas operações incluem, por exemplo, rotações, deslocamentos, permutações, concatenações, substituições, etc. Normalmente estas operações são muito fastidiosas de programar, tanto em linguagens de alto nível como em linguagem máquina, o que complica o desenvolvimento de novos algoritmos.

Pretende-se com este trabalho desenvolver uma linguagem para descrever a operação de um algoritmo de cifra (e decifra). A linguagem deverá suportar a definição de operandos com dimensão, de forma a permitir uma validação semântica e a balizar os tipos necessários na geração de código. Para obter uma lista de operadores a concretizar, pode-se ver como exemplo os algoritmos do DES, IDEA, AES, SHA-1, SHA-2, SHA-3 e A5.

Existem também os chamados modos de cifra, que são formas genéricas de aplicação de cifras a volumes de dados arbitrários. Estes modos também permitem definir novas cifras à custa de outras cifras. Neste sentido, a linguagem deverá permitir o desenvolvimento de uma cifra recorrendo a outra.

Outro aspeto interessante é a definição, através de uma linguagem, de padrões de teste de um algoritmo. Por exemplo, certas cifras produzem sequências de bits que se assemelham a sequências aleatórias (em termos estatísticos), outras possuem o chamado efeito de avalanche: a mudança de 1 bit causa uma alteração na saída que seguirá uma distribuição Gaussiana, centrada nos 50%. Outras ainda podem ter valores internos que terão desejavelmente uma distribuição normal.

Outro aspeto interessante é a definição, através de uma linguagem, do modelo de aplicação de uma cifra a dados estruturados, onde se pode definir que dados permanecem alterados, que dados devem ser cifrados e que envelope se dá aos dados cifrados para serem correctamente interpretados pelo receptor. Esse envelope tipicamente indica o criptograma e meta-informação relativa a sua geração (quais o(s) algoritmo(s) usado(s), o modo de cifra,

o alinhamento, etc.). Existem várias formas padrão de realizar este empacotamento (e.g. PKCS #7, PKCS #12), os quais poderão ser sumariamente concretizados com a linguagem a desenvolver (está fora de questão a realização completa destes padrões).

(Para uma melhor compreensão deste problema deve falar com o Prof. André Zúquete.)