Compiladores/Projecto de Compiladores/Fases Desenvolvimento

*From Wiki**3*

< Compiladores | Projecto de Compiladores

AVISOS - Avaliação em Época Normal **[Expand]**

Material de Uso Obrigatório **[Expand]**

Contents

Introdução

O desenvolvimento do novo compilador deve ser incremental. O desenvolvimento dos novos processos de análise e geração deve ser horizontal.

O desenvolvimento do compilador, em especial na análise sintática e geração de código, pode evoluir da seguinte forma (partindo da sintaxe/semântica do compilador exemplo):

1. migrar os tipos de dados básicos (inteiros, etc.) para a linguagem a implementar;
2. migrar a instrução de impressão para a semântica a implementar;
3. migrar/criar expressões aritméticas e lógicas inteiras;
4. migrar/criar instruções condicionais e de ciclo;
5. implementar invocação de funções;
6. implementar funções;
7. outras partes da linguagem.

Embora as fases de análise genéricas (lexical, sintáctica, semântica) estejam indicadas sequencialmente, devem ser atacadas horizontalmente e em simultâneo para cada parte da linguagem. É natural que alguns tópicos da teoria não estejam cobertos em momentos do desenvolvimento. Nestes casos, devem ser criados esqueletos de código para definir a localização do material em falta.

Análise Lexical

A análise lexical da linguagem é realizada pela ferramenta **flex** (em modo C++), que deve operar (sem erros) sobre o ficheiro que contém a especificação lexical (**.l**). Durante a análise lexical, deverá ser possível remover comentários e espaços brancos, identificar literais (valores constantes), identificadores (nomes de variáveis e funções), palavras chave, etc. Notar que a análise lexical não garante que estes elementos se encontrem pela ordem correcta. As sequências de escape nas cadeias de caracteres deverão ser substituídas pelos respectivos caracteres (ver manual de referência da linguagem), o espaço necessário para o texto dos identificadores e cadeias de caracteres deverá ser reservado antes de devolvido.

A avaliação da análise lexical considera as expressões regulares utilizadas, bem como a robustez, clareza, simplicidade e extensibilidade da solução.

Análise Sintáctica

A análise sintáctica é realizada pela ferramenta byacc, que opera sobre o ficheiro que contém a especificação sintáctica (**.y**), sem gerar qualquer tipo de conflitos (*shift-reduce* ou *reduce-reduce*) nem avisos ou erros. Com a análise sintáctica, é possível garantir a correcta sequência dos símbolos, embora não se verifique se as variáveis utilizadas nas expressões estão

declaradas, nem se as operações suportam os tipos de dados utilizados, etc. Simultaneamente, são associadas acções que permitam construir uma árvore de análise sintáctica abstracta do programa a ser processado. Para tal, são utilizadas subclassess de **cdk::basic_node** (existentes na CDK, ou definidas por derivação, no contexto do novo compilador).

Eventuais erros em programas, encontrados durante a análise sintáctica, devem ser identificados e descritos ao utilizador (o processamento não deve ser interrompido, devendo o ficheiro deve ser processado até ao fim, procurando outros erros).

Análise Semântica e Geração de Código

A avaliação da AST considera a gramática do ponto de vista das verificações de coerência tratadas sintacticamente, das regras escolhidas e símbolos não terminais escolhidos, bem como a robustez, clareza, simplicidade e extensibilidade da solução proposta.

A análise semântica deverá garantir que um programa se encontra correctamente escrito e que pode ser executado.

A análise semântica identifica todos os erros estáticos (detectáveis no processo de compilação) e produzidas mensagens de erro descritivas. Caso surjam erros semânticos (estáticos), o compilador deverá terminar com um código de erro 2 (dois). Na sequência da análise semântica, e caso não surjam erros sintácticos ou semânticos (estáticos), é gerado o código final através da CDK (subclasses de **cdk::basic_postfix_emitter**).

Utilizando a árvore sintáctica abstracta (composta por instâncias de subclasses de **cdk::basic_node**), pode-se efectuar a análise semântica e a geração de código numa só passagem em profundidade sobre a árvore sintáctica gerada.

Embora a análise semântica possa decorrer em paralelo com a geração de código, não deve ser gerado código se houver erros em alguma das fases de análise. Recorda-se que a execução de código pelo processador realiza poucas verificações, podendo produzir resultados indesejados. **Aconselha-se nunca executar os programas gerados como utilizador previlegiado (e.g., em Linux, como "root" (uid=0) ou equivalente) ou num sistema sem protecção.**

A geração de código final é realizada através das subsclasses de **cdk::basic_postfix_emitter** fornecidas. Estas classes utilizam o processador como uma máquina de pilha para gerar código final. Por exemplo, uma soma de duas constantes inteiras pode ser efectuada por:

```
// exemplo para gerador ix86 (ferramenta yasm/nasm)
basic_postfix_emitter *gen = new postfix_ix86_emitter(compiler);
gen->INT(2);                  // coloca o primeiro operando na pilha
gen->INT(2);                  // coloca o segundo operando na pilha
gen->ADD();                   // realiza a soma e deixa o resultado na pilha
```

O operador retira os operandos da pilha e deixa lá o resultado da operação. A impressão de um inteiro no topo da pilha pode ser realizada através das seguintes instruções:

```
gen->CALL("printi");   // chama a rotina de impressão
gen->TRASH(4);         // remove o operando da pilha
```

A remoção do resultado da soma da pilha é necessária, depois da impressão, já que na convenção de chamada (C) é o chamador quem deve retirar os argumentos. A documentação da classe **cdk::basic_postfix_emitter** está disponível na CDK.

Execução de Programas

O resultado da "execução" das instruções Postfix é um ficheiro assembly, que deverá ser compilado com a ferramenta **yasm**, por forma a produzir um binário ELF. Assim, o seguinte comando permite gerar o ficheiro **file.o** a partir do ficheiro **file.asm** (assumindo que não são encontrados erros).

```
yasm -felf32 file.asm
```

A geração do executável é efecutada através da ligação do ficheiro objecto com a biblioteca de run-time da linguagem **librts.a** -- biblioteca RTS -- com o seguinte comando (podem ser incluídos outros ficheiros objecto ou bibliotecas, caso se utilize compilação separada).

```
ld -m elf_i386 -o executavel file.o -lrts
```

Informação Adicional e Material de Apoio

- Ver também Material de Apoio ao Desenvolvimento.

Categories: **Projecto de Compiladores** **Compiladores** **Ensino**