Conclusões

Baseado no Capítulo 9 de Programming Language Processors in Java, de Watt & Brown

Atualizado em 29/03/2018

QUESTÕES FUNDAMENTAIS

1. Correção do código gerado
2. Desempenho do compilador:
    a. Notificação de erros;
    b. Eficiência.
3. As características de um projeto de compilador dependem diretamente das características de projeto da linguagem de programação-fonte.

O CICLO DE VIDA DE UMA LINGUAGEM DE PROGRAMAÇÃO

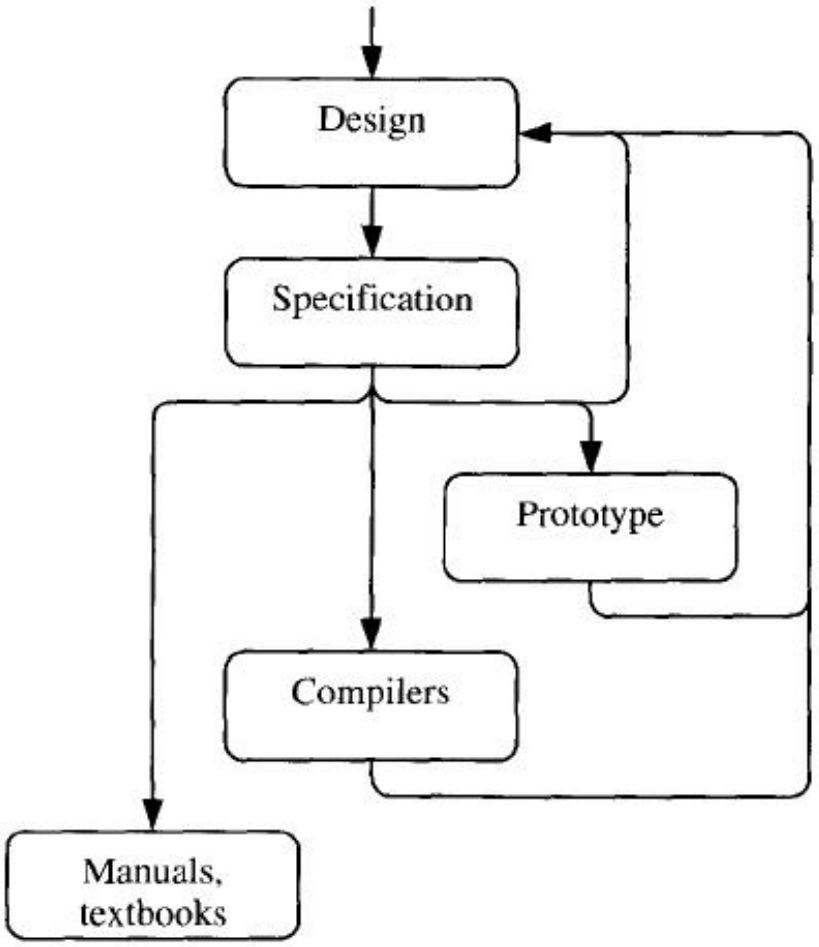

Figure 9.1 A programming language life cycle model.

O PROJETO DE UMA LINGUAGEM DE PROGRAMAÇÃO

- Inúmeros recursos e conceitos estão à disposição;
- A escolha deve levar em conta a área de aplicação a que se destina a linguagem;
- A complexidade da linguagem afeta diretamente a complexidade do compilador;
- Uma escolha criteriosa e consistente deve ser feita;
- Simplicidade e regularidade devem ser observadas;
- Outros pincípios são aplicáveis;
- O assunto não é trivial e existe muita literatura à respeito.

A ESPECIFICAÇÃO DE UMA LINGUAGEM DE PROGRAMAÇÃO

- O projeto precisa ser comunicado para aquele que irá implementar a linguagem;
- Deve-se descrever, de forma completa e não-ambígua, toda a sintaxe e toda a semântica;
- O uso de gramáticas livres de contexto (BNF, EBNF etc) contribui de forma decisiva para uma maior concisão, simplicidade, uniformidade e elegância da sintaxe livre de contexto;
- FORTRAN e COBOL tiveram problemas com a sintaxe descrita de maneira informal;
- Dependências de contexto e semântica são descritas de maneira informal;
- No entanto, uma boa formalização da sintaxe livre de contexto contribui para uma melhor especificação daquelas outras duas componentes;

PROTÓTIPOS

- Implementação de baixa qualidade de uma linguagem de programação;
- Baixa qualidade refere-se ao desempenho e não à correção;
- Tem objetivo experimental e educacional (treinamento);
- Pode ser importante para fazer ajustes no projeto da linguagem;
- Outra forma é traduzir para uma segunda linguagem de programação de alto-nível;
- Protótipos não são adequados para uso por longos períodos nem por grandes grupos de usuários;
- Compiladores com qualidade industrial (alta qualidade) devem ser construídos e usados depois dos experimentos feitos com os protótipos;
- Eles devem ser otimizados em relação ao tempo de execução do próprio compilador e também do código gerado;
- Da mesma forma, eles devem ser otimizados em relação à quantidade de memória requerida para a compilação e a execução do código gerado;
- Mensagens de erro devem ser precisas e informativas.

NOTIFICAÇÃO DE ERROS

- Deve identificar com precisão o ponto do programa-fonte onde o erro foi cometido;
- Deve informar o usuário com mensagens claras sobre a natureza do erro cometido e possíveis formas de corrigir o mesmo;
- Deve evitar que outras mensagens de erro sejam produzidas como decorrência de erros anteriores cometidos;
- Envolve técnicas para a recuperação de erros;
- Mecanismos de depuração são úteis e precisam ser previstos pelo compilador.

Erros detectados em tempo de compilação:

- Sintaxe livre de contexto (símbolos errados, em excesso ou faltantes);
- Violação das regras de escopo;
- Violação das regras de tipos

Erros detectados em tempo de execução:

- Overflow (underflow)
- Divisão por zero
- Expressões indexadoras inválidas
- Etc

- Se o compilador não puder verificar em tempo de tradução, ele precisa instrumentar o código para que a verificação ocorra em tempo de execução;
- Recursos do hardware podem ser úteis para detectar erros em tempo de execução;

EFICIÊNCIA

Normalmente as eficiências em tempo de compilação e em tempo de execução são inversamente proporcionais;

- Eficiência em tempo de compilação;
- Eficiência em tempo de execução.

Eficiência em tempo de compilação:

- Compiladores mais eficientes geram código para máquina abstratas; máquinas reais oneram os compiladores com seus detalhes e idiossincracias;
- Análise sintática, verificação de tipos e geração de código são O(n);
- Verificação de regras de escopo é geralmente o ponto crítico;
- Conforme a técnica usada, pode variar de $O(n^2)$ (para busca linear) até quase O(n), (para casos especiais de hash) passando por O(n log n) (para árvores binárias balanceadas);
- Algumas técnicas de otimização de código podem ser $O(n^2)$;
- De uma forma geral, um compilador que usa árvore binária para a tabela de símbolos executa em O(n log n).

Eficiência em tempo de execução:

- Depende do modelo usado (pilha ou registradores);
- Depende de otimizações (independentes e/ou dependentes de máquina);
- Técnicas de otimização mais usadas:
  - Constant folding;
  - Common subexpression elimination;
  - Code movement.
- Impactam diretamente a eficiência em tempo de compilação;
- Normalmente são opcionais e selecionadas sob comando do programador.

GERADORES DE COMPILADORES:

- Adequados para analisadores léxicos e sintaxe livre de contexto;
- Incorporam código para outras fases da compilação;
  - UNIX lex & yacc (LR em C);
  - JavaCC e JavaLex (LL em Java);
  - Flex, Bison etc
  - http://javacc.org/
  - http://dinosaur.compilertools.net/

QUESTÕES FUNDAMENTAIS

- Conceitos, projeto e especificação de linguagens de programação;
- Modelos para ambientes de execução;
- Técnicas básicas para a implementação de processadores de linguagens;
- Técnicas para identificação, notificação e recuperação de erros;
- Métodos para otimização de código.