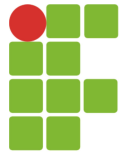
Introdução

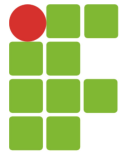
Compiladores

Prof. Daniel Saad Nogueira Nunes

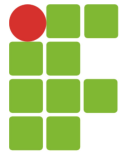
IFB – Instituto Federal de Brasília,
Campus Taguatinga



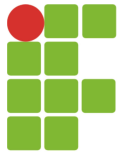
Sumário



Sumário

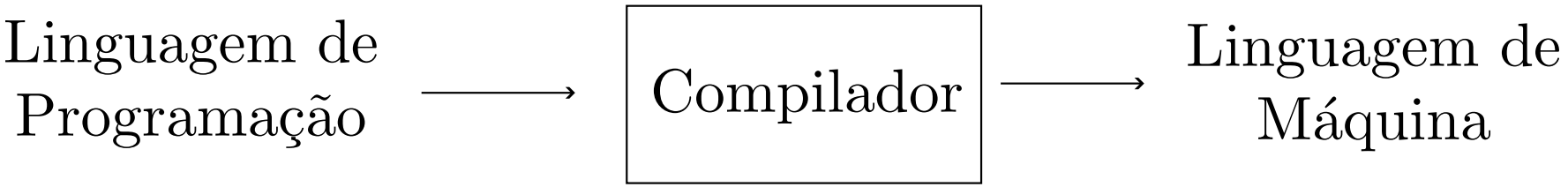


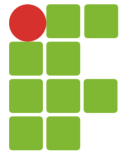
Sumário



Compiladores

- Compiladores tem como função traduzir um código com um nível de abstração mais alto (*eg.* linguagem de programação) em outro com um nível de abstração mais baixo (*eg.* linguagem de máquina).



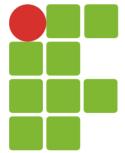


Compiladores

Tipos de códigos de máquina

Existem tipos diferentes de códigos de linguagem de máquina que compiladores podem produzir:

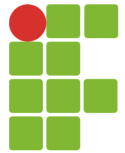
- Código de máquina puro.
- Código de máquina aumentado.
- Código de máquina virtual.



Compiladores: tipos de códigos de máquina

Código de máquina puro

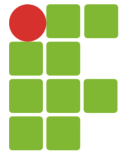
- Contém apenas instruções do conjunto de instruções da arquitetura (ISA).
- Executa diretamente no hardware, sem qualquer dependência.
- Não presume existência de mecanismos do S.O. ou de bibliotecas.
- Raro, pois muitos compiladores assumem algum grau de interface com as chamadas de sistema do S.O. e bibliotecas.
- Este tipo de código é mais utilizado em compiladores para implementação de sistemas operacionais ou aplicações embarcadas.



Compiladores: tipos de códigos de máquina

Código de máquina aumentado

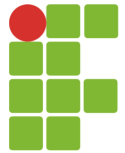
- Este tipo de código de máquina leva em consideração os mecanismos de S.O e bibliotecas.
- A execução deste tipo de código requer um S.O. específico e suporte de bibliotecas de sistema.



Compiladores: tipos de códigos de máquina

Código de máquina virtual

- O código de máquina virtual contém apenas instruções virtuais.
- Executável através de uma **máquina virtual**.
- Exemplo: bytecode Java, o qual pode ser executado através da máquina virtual java.
- Portável: pode ser executado em diferentes plataformas.

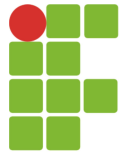


Compiladores

Formato de códigos gerados

Outra forma de enxergar compiladores é através do formato do código gerado por eles, os quais podem ser categorizados como:

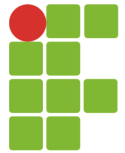
- Assembly (linguagem de montagem).
- Binário relocável.
- Binário absoluto.



Compiladores: formato de códigos gerados

Assembly

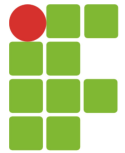
- O processo de produzir código assembly pode simplificar a tradução.
- Algumas decisões de geração de código de máquina, especialmente ligadas ao endereçamento de memória, são deixadas a cargo do montador.
- Fácil examinação, o que torna essa decisão fácil para fins didáticos ou de prototipagem.
- Útil para compilação cruzada,



Compiladores: formato de códigos gerados

Binário relocável

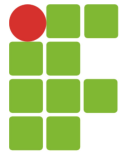
- Formato mais eficiente e permite o compilador ter mais controle sobre o processo de geração de código.
- Mais utilizado por compilador a nível de produção.
- Endereços são relativos ao início de cada módulo ou a alguma referência simbólica.
- Um processo de ligação é necessário para incorporar bibliotecas e outras subrotinas compiladas.
- O resultado do processo de ligação é o binário absoluto.



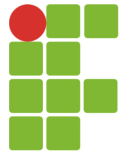
Compiladores: formato de códigos gerados

Binário absoluto

- Pode ser executado diretamente sem nenhum processo adicional, como o de ligação.
- Habilidade de realizar interface com outro código fica mais restrita.

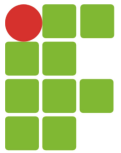


Sumário

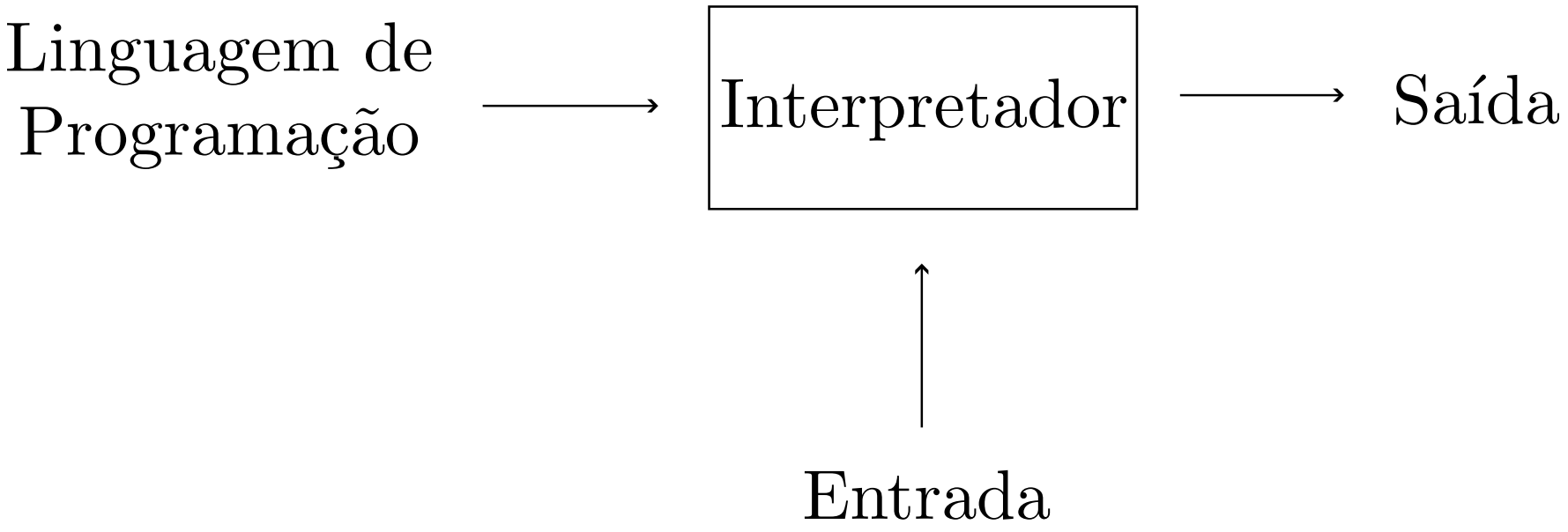


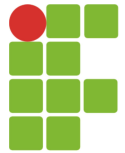
Interpretadores

- Interpretadores tem alguns componentes em comum com os compiladores, tais como as análises sintática e semântica.
- Contudo eles não executam o processo de tradução como os compiladores.
- Os comandos do código são interpretados sob demanda.



Interpretadores

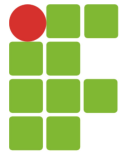




Interpretadores

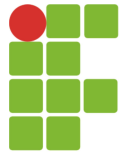
Interpretadores geralmente fornecem algumas características não disponíveis a compiladores, tais como:

- Programas podem ser modificados enquanto executados. Possibilita uma depuração iterativa.
- Linguagens em que tipos são construídos dinamicamente (*eg.* Lisp e Scheme) são mais problemáticas de serem compiladas, visto que o programa deve ser reexaminado continuamente enquanto executa.
- Possuem uma independência de hardware, visto que nenhum código de máquina é gerado. Todas as operações estão contidas no interpretador.

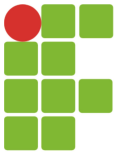


Interpretadores

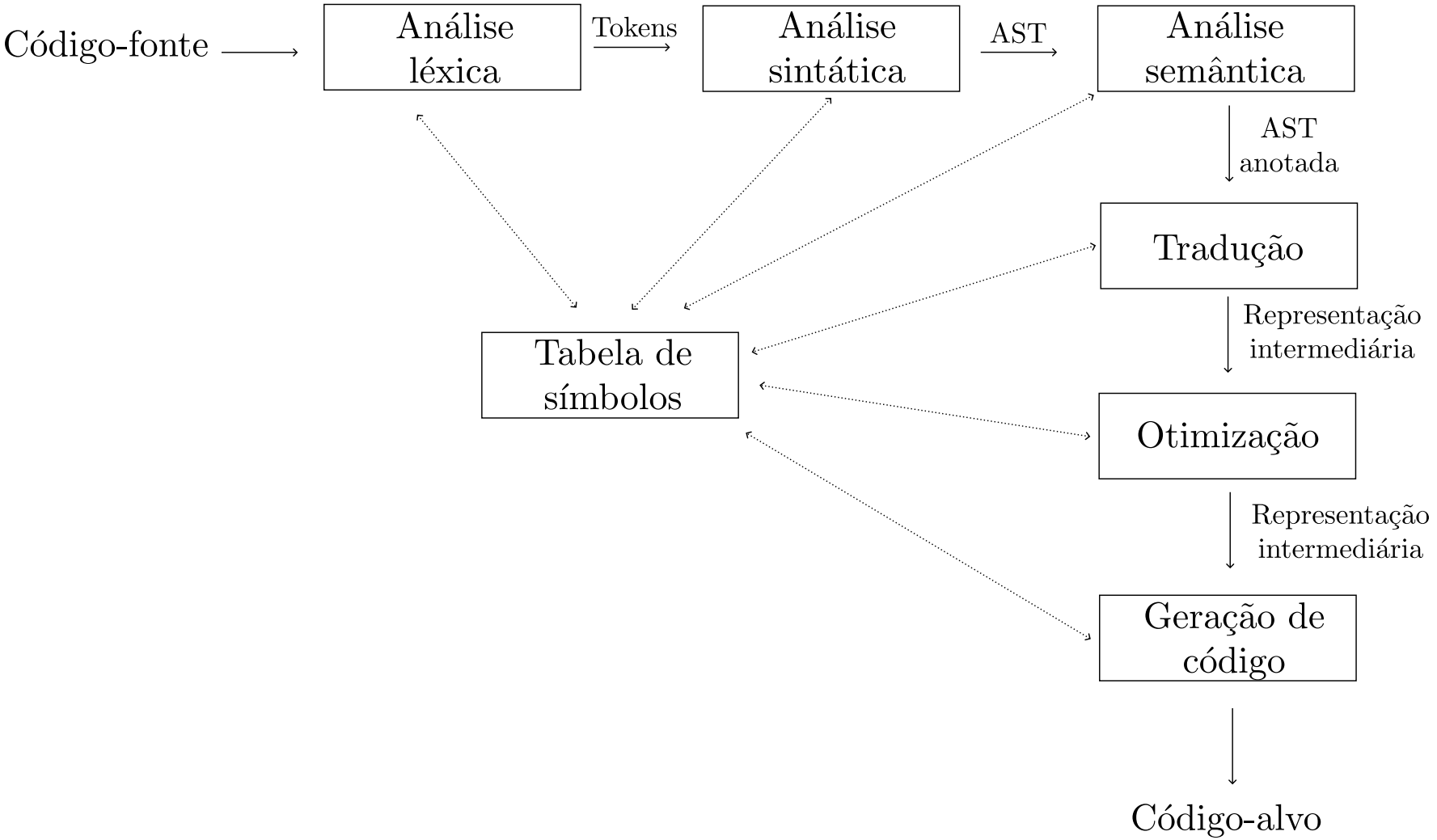
- Esta flexibilidade traz consigo uma penalidade de tempo.
- A interpretação pode ser uma ou duas ordens de magnitude mais lentas do que a execução de um código compilado.

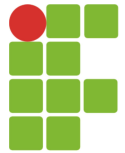


Sumário

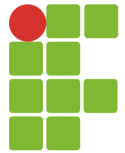


Componentes de um compiladores



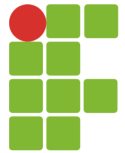


Sumário



Analisador léxico

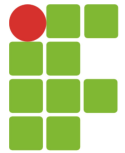
- O analisador léxico (scanner) lê o código-fonte agrupando caracteres como tokens.
- Reconhece identificadores, tipos, palavras reservadas, delimitadores, entre outros.
- Os tokens são representados usualmente como inteiros e utilizados na fase de análise sintática. Quando necessária em outras etapas, a string propriamente dita também é utilizada.



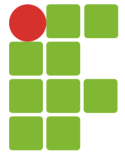
Analisador léxico

Em suma, o analisador léxico:

- Representa o programa de maneira compacta: uma sequência de tokens.
- Elimina informações desnecessárias (*eg.* comentários).
- Preenche informações preliminares na tabela de símbolos, tais como identificadores ou rótulos.
- Processa diretivas de compilação, como inclusão de outros códigos-fonte.



Sumário



Analisador sintático

- O analisador sintático (parser) segue a especificação de uma gramática livre de contexto.
- Ele lê os tokens e os agrupa em uma estrutura de árvore de acordo com as regras da **sintaxe**.
- Ele verifica se as regras são obedecidas e, em caso negativo, informa erros de sintaxe.
- Produz uma árvore sintática abstrata (AST em inglês), que representa a estrutura sintática do programa.

AST

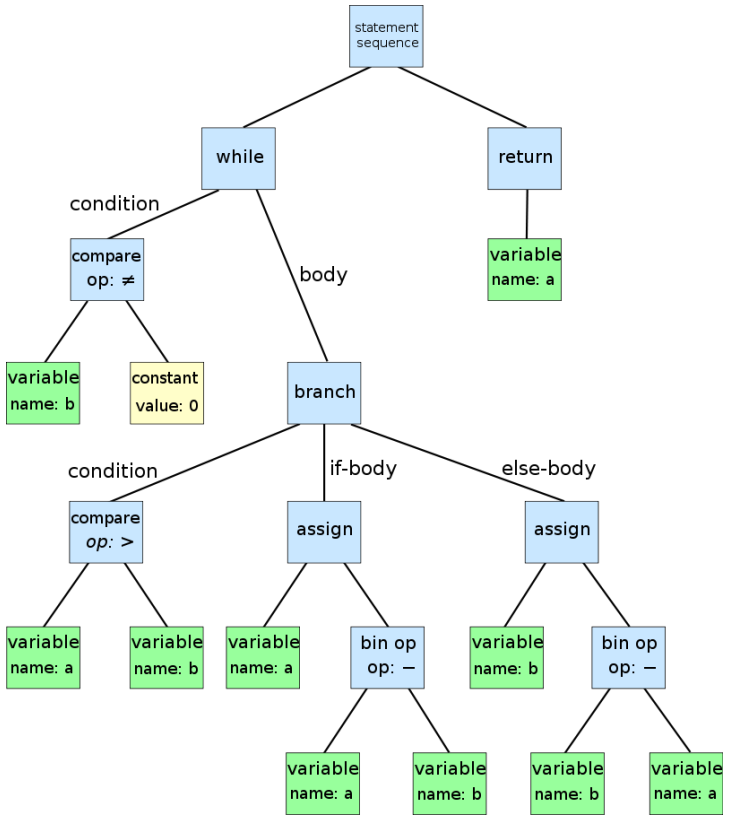
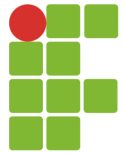
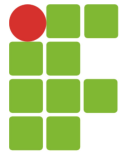


Figura: Fonte:
`https://en.wikipedia.org/wiki/Abstract_syntax_tree`

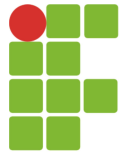


AST

```
def mdc(a,b):
    while b != 0:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

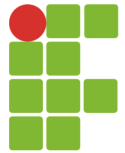


Sumário

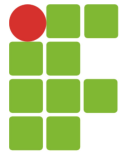


Analisador semântico

- Verifica se a construção de cada nó da AST faz sentido do ponto de vista semântico.
- Verificação de tipos, se as variáveis foram declaradas, etc. . .
- A AST vai sendo anotada (decorated AST) conforme o processo avança.
- Se um erro semântico é identificado, um erro é reportado.

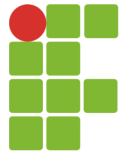


Sumário



Tradução

- Se um nó da AST está correto semanticamente, ele pode ser traduzido em uma linguagem intermediária (IR em inglês).
- O tradutor é amplamente guiado pela semântica da linguagem fonte, pouco da natureza da máquina alvo precisa ser utilizado.
- Em compiladores mais simples, que não lidam com otimização de código, o tradutor pode gerar o código alvo diretamente, sem traduzir para uma linguagem intermediária.
- O GCC primeiramente gera uma IR e, em seguida, traduz o código em IR para a linguagem alvo.
- Essa estratégia permite uma separação clara das dependências das da fonte e do alvo.



Sumário

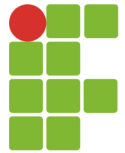
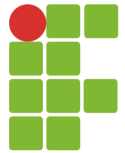
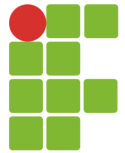


Tabela de símbolos

- Permite a associação entre os identificadores e é compartilhada com as várias fases de uma compilação.
- Toda vez que um identificador é declarado ou usado, uma tabela de símbolos providencia acesso à informação coletada.
- Muito utilizadas no procedimento de verificação de tipos.

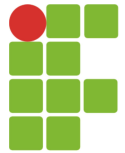


Sumário

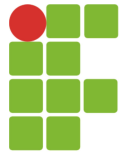


Otimizador

- O otimizador analisa o código IR e o transforma em um código semanticamente equivalente, mas otimizado.
- Um compilador com um bom otimizador pode aumentar significantemente o desempenho de execução.
- O otimizador também pode atuar após o código alvo ter sido gerado.

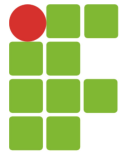


Sumário

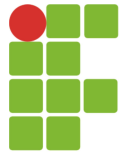


Gerador de código

- O gerador de código mapeia o código IR produzido nas fases anteriores para o código alvo de máquina.
- Requer informações detalhadas sobre a máquina alvo e incluir otimizações específicas para a máquina, tais como alocação de registradores.

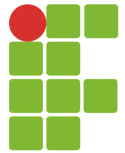


Sumário

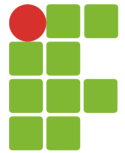


Variantes

- Existem algumas variantes de compiladores utilizadas em nichos específicos, tais como:
  - Compiladores de depuração.
  - Compiladores otimizadores.
  - Compiladores redirecionáveis.
  - IDEs.

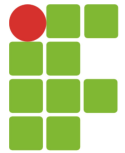


Sumário

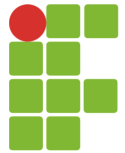


Compiladores de depuração

- Especialmente projetados para ajudar na depuração de programas em desenvolvimento
- Examina cuidadosamente os programas e detalha erros de programação.
- Incluem códigos de verificação em tempo de execução.
- Podem ser utilizados durante o desenvolvimento e depois trocados por um compiladora nível de produção para maximizar o desempenho.

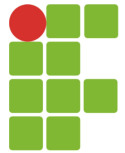


Sumário

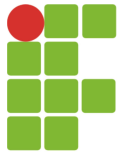


Compiladores otimizadores

- São compiladores projetados para produzir código alvo eficiente em troca de uma complexidade maior e aumento do tempo de compilação.
- Essa complexidade se dá por conta das varias transformações que conflitam entre si.
- Por exemplo, manter variáveis frequentemente utilizadas em registradores pode reduzir o tempo de acesso a elas, mas, aumentar o tempo das chamadas de funções, visto que os registradores precisam ser salvos entre chamadas de funções.



Sumário



Compiladores redirecionáveis

- Compiladores redirecionáveis permitem que o código de máquina gerado possa ser modificado, a depender da arquitetura.
- Idealmente nesses compiladores, os componentes independentes do código de máquina não precisam ser reescritos.
- Em compensação, é mais complicado gerar código tão eficiente quanto um compilador especializado, dada a dificuldade de explorar casos especiais e idiossincráticos relacionados à máquina objetivo.

o