Interpretação e Compilação de Linguagens de Programação

28 de Fevereiro de 2013

Esta unidade curricular pretende transmitir ao longo de um semestre as noções fundamentais sobre o desenho e implementação de linguagens de programação. Muitos dos conceitos estudados são essenciais, não só no desenvolvimento de linguagens de programação, mas também na resolução de problemas de forma computacional [Win06]. Pretende-se ao mesmo tempo aprofundar o conhecimento acerca do funcionamento de linguagens de programação já existentes através do conhecimento profundo dos seus mecanismos de construção.

> *If you don't understand interpreters, you can still write programs; you can even be a competent programmer. But you can't be a master.* (Hal Abelson, *in* prefácio de [FW08])

Durante esta unidade curricular serão apresentados os componentes fundamentais de construção de linguagens de programação, o seu modelo de execução, verificação e transformação, tendo como foco uma linguagem de alto nível de abstracção. Ao longo de um semestre serão desenvolvidos de forma incremental algoritmos interpretadores, que serão os veículos privilegiados para a aprendizagem da semântica de linguagens, dos sistemas de tipos e das técnicas de compilação. Nesta primeira versão das notas de apoio serão descritos brevemente os tópicos abordados em aula, algum código de apoio, e alguns apontadores para outros recursos de estudo e aprofundamento de conhecimentos.

Nesta primeira aula descreve-se a arquitetura geral dos interpretadores e compiladores e a sua utilidade no contexto de um sistema de software. Estes tópicos podem ser mais explorados na bibliografia recomendada (nomeadamente em [AP02, AVAU06]).

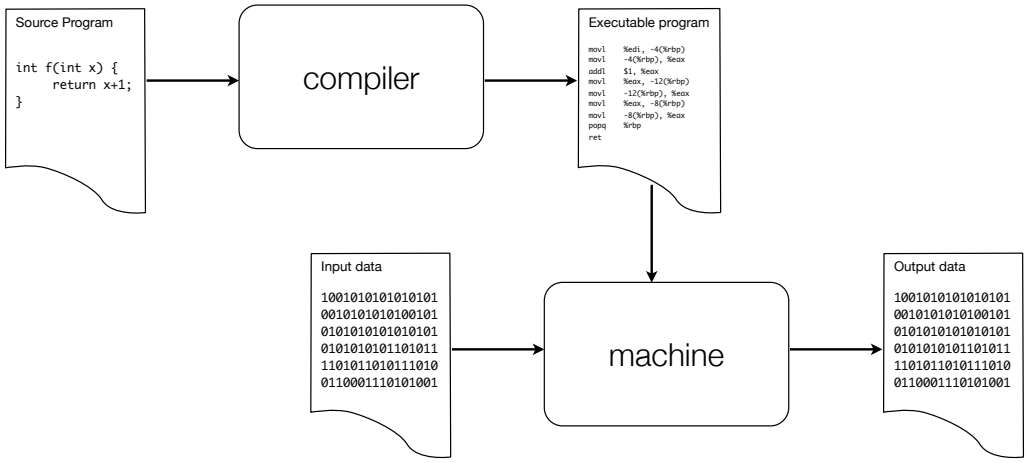

Figura 1: Compilador

1 Interpretadores / compiladores

Qual a diferença entre um compilador e um interpretador? o que é uma linguagem máquina? ou uma linguagem intermédia? o que é um *JIT compiler*? Cada uma destas ferramentas tem a sua utilidade específica e contexto de utilização adequado.

Os interpretadores e compiladores são essencialmente programas cujo comportamento (ou output) não está estabelecido à partida, o seu resultado é definido através de um programa dado como input, escrito numa linguagem de programação. São programas cujos dados são outros programas. Dentro desta categoria encontram-se ainda programas como ferramentas de verificação de programas, ambientes de desenvolvimento, etc.

Invariavelmente, os dados de entrada de um compilador e de um interpretador é um programa expresso numa linguagem de programação (linguagem fonte). O mais usual é um programa ser expresso por intermédio de um texto, podendo ser expresso de outras formas, usando linguagens visuais por exemplo. Embora a maioria dos conceitos sejam genéricos, no contexto desta unidade curricular focamos em linguagens expressas em texto.

A distinção entre interpretadores e compiladores faz-se essencialmente a nível da forma de execução e interação que proporcionam. Existe depois uma miríade de alternativas e combinações sobre as verificações que são feitas sobre o código, e sobre a eficiência na execução, e na produtividade da atividade de programação.

Os compiladores, cuja interação é representada na Figura 1, transformam

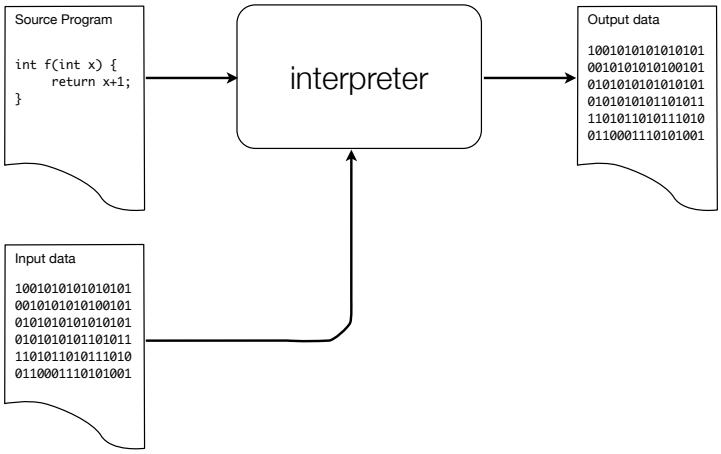

Figura 2: Interpretador

código de uma linguagem — tipicamente com um nível de abstração alto — para uma linguagem com um nível de abstração mais baixo. Esse programa alvo é executado ou interpretado por uma máquina cujas instruções são de uma granularidade mais baixa e adaptadas às suas estruturas internas. De uma forma simplificada representamos essa interação, nas Figuras 1 e 2, pela transformação de dados de entrada em dados de saída. Esta forma de interação pode sem perca de generalidade ser extendida para aplicações interativas mais ricas.

Os interpretadores (simples) processam diretamente o programa na linguagem fonte e os dados de entrada para produzir os seus resultados ou efeitos, Figura 2. Uma configuração mais realista e sofisticada como é o caso da máquina virtual Java (JVM) ou a máquina virtual da plataforma .NET combina compilação e interpretação. A sua arquitetura inclui a compilação para código numa linguagem intermédia que depois é interpretado, e nessa altura ainda pode ser compilado para código nativo (máquina) e executado diretamente no processador.

Comparando os dois modos de execução, podemos dizer que os compiladores produzem código cuja execução é mais eficiente do que a interpretação do mesmo código. Esta realidade deve-se ao que o processamento "offline" do código oferece como oportunidades de optimização e verificação que a interpretação de código tipicamente não permite. Por outro lado, aos interpretadores são associadas as características de flexibilidade no desenvolvimento, permitindo inclusivamente a execução de programas, cuja compilação para código nativo seria impossível, sem incorporar de alguma forma o algoritmo interpretador no resultado (e.g. Javascript).

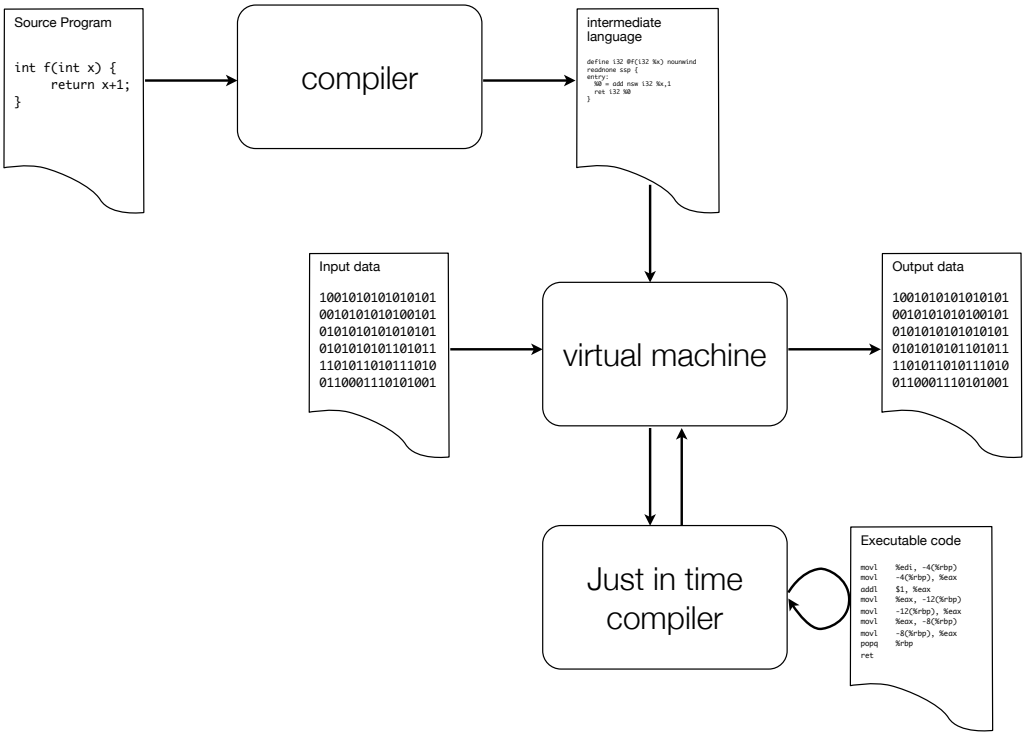

Figura 3: Interpretador com compilador JIT

A flexibilidade oferecida pela utilização de um interpretador (da linguagem intermédia) é aproveitada no caso das linguagens Java e C# para possibilitar a portabilidade de código. No caso de linguagens como OCaml, Haskell ou Scala, à flexibilidade dada pelo modo interpretação, acrescenta uma verificação de tipos completa e optimizações como a utilização de *tail recursion*.

Na sua estrutura interna, os compiladores e interpretadores partilham alguns componentes, nomeadamente no seu *front-end*. Na Figura 4, são descritos os blocos que tipicamente compõem o processo de transformação feito por um interpretador. Os dois primeiros blocos, análise lexicográfica e sintática, transformam o formato de entrada (texto) numa representação interna do programa (tipo de dados) que pode ser mais facilmente tratado por um algoritmo (esta representação será o tema da próxima aula). Essa representação, que pode ou não ser analisada e anotada, é depois interpretada.

A arquitetura interna de um compilador é composta por um *front-end* como aquele descrito para os interpretadores, e por mais duas camadas. Uma camada intermédia e uma camada de saída. A camada intermédia

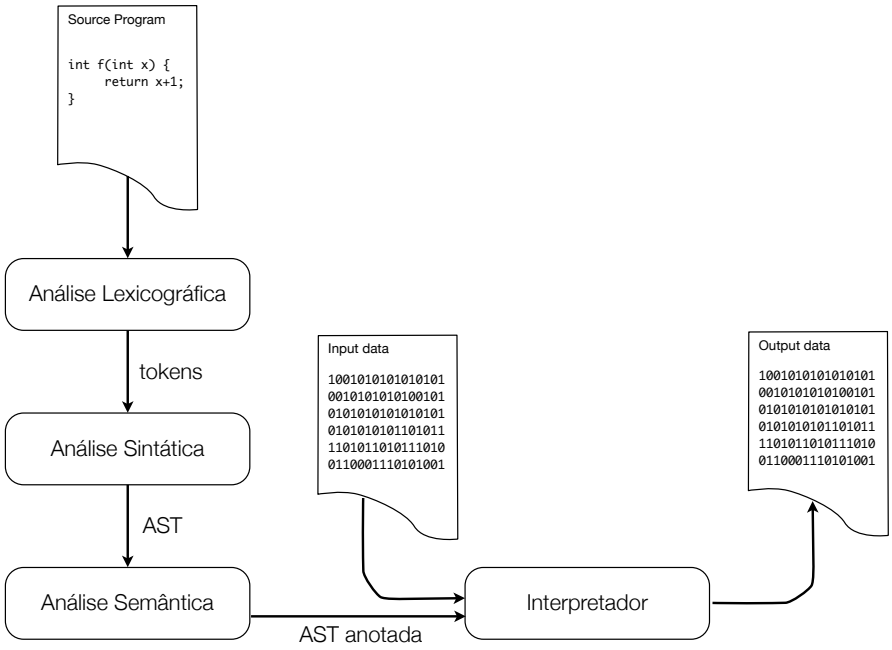

Figura 4: Arquitetura de um interpretador

de um compilador típico, faz o tratamento de uma linguagem intermédia. Esta divisão entre *front-end* e *back-end*, com um ponto comum numa camada intermédia, permite com uma única ferramenta, criar uma família de compiladores. Compilando e integrando um conjunto de linguagens, e podendo produzir código de qualquer destas linguagens para um conjunto de processadores.

A Figura 5 mostra os principais blocos que compõem um compilador. Os componentes do *front-end* de um compilador são basicamente os mesmos do que os de um interpretador. A diferença começa na geração de código numa linguagem intermédia que permite normalmente a realização de operações de análise e optimização de código, e que é agnóstica quanto ao processador para o qual se vai gerar código nativo. O *back-end* do processador pode então gerar código específico para diferentes processadores, tirando partido das particularidades de cada um.

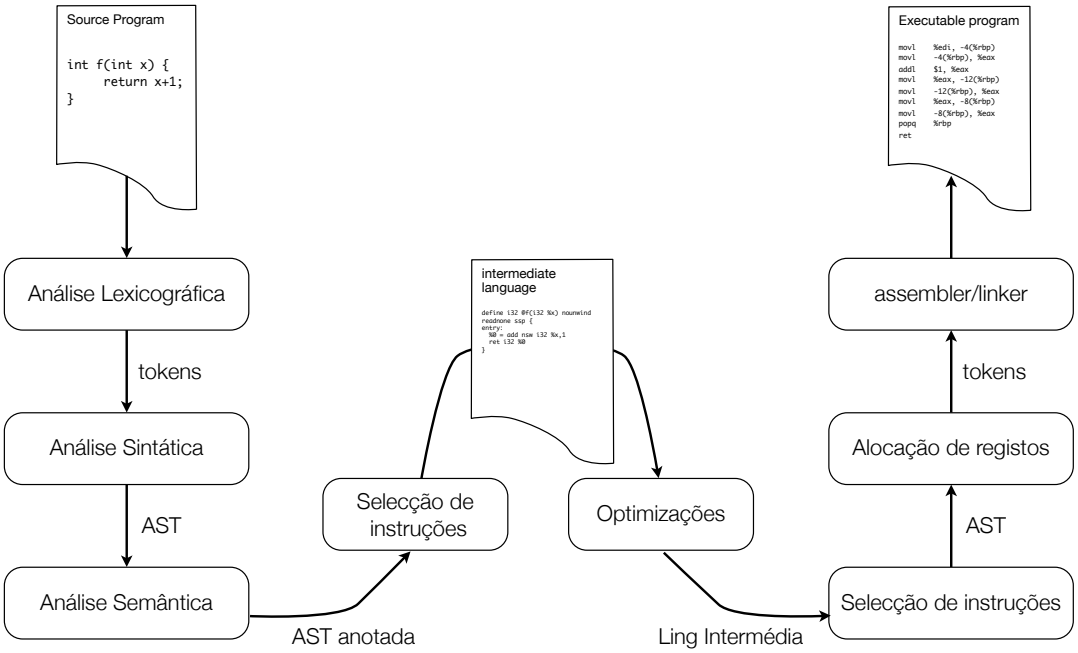

Figura 5: Arquitetura de um compilador (adaptado de [Pfe])

Referências

[AP02] A.W. Appel and J. Palsberg. *Modern Compiler Implementation in Java*. Cambridge University Press, 2002.

[AVAU06] Ravi Sethi Alfred V. Aho, Monica S. Lam and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Pearson Education, Inc, 2006.

[FW08] D.P. Friedman and M. Wand. *Essentials of programming languages*. Essentials of Programming Languages. MIT Press, 2008.

[Pfe] Frank Pfenning. Lecture notes: compiler design, overview.

[Win06] Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, March 2006.