Compiler und Interpreter

Es gibt verschiedene Möglichkeiten, Quellcode in ein lauffähiges Programm zu übersetzen. Die beiden grundlegenden Varianten sind Compiler und Interpreter.

Compiler (genauer Ahead-of-Time-Compiler)

Ein Compiler übersetzt Quellcode in Maschinencode, der auf einer bestimmten Plattform läuft. Mit Plattform ist hier die Kombination aus Computer und Betriebssystem gemeint, zum Beispiel ein Intel-PC mit Windows.

Ein einmal kompiliertes Programm läuft relativ schnell und kann immer wieder gestartet werden. Der Compiler hat die Möglichkeit, beim Übersetzen den Code zu optimieren. Dabei übersetzt er zum Beispiel Programmteile anders, als sie der Programmierer geschrieben hat, oder er legt einzelne Variablen nicht im Arbeitsspeicher an, sondern direkt in CPU-Registern.

Beispiele für Compiler-Sprachen:

C++, C, Pascal, Delphi, Rust

Interpreter

Interpreter führen den Quellcode Befehl für Befehl auf der CPU aus. Das bedeutet, dass jeder Anwender den Quellcode haben muss und dass die Programme in der Regel langsamer laufen als mit einem Compiler. Dafür ist das Übertragen auf eine andere Plattform meistens sehr einfach.

Beispiele für Interpreter-Sprachen:

PHP, Python, JavaScript, Basic, Perl, Ruby

Just-in-Time-Compiler (JIT-Compiler)

Ein JIT-Compiler kompiliert benötigte Programmteile zur Laufzeit. Er behält die meisten Vorteile eines Interpreters bei, hat aber trotzdem die Möglichkeit, den Code zumindest eingeschränkt zu optimieren. Im Gegensatz zu einem gewöhnlichen Compiler nutzt er dafür auch die zur Laufzeit vorhandenen Daten und nicht nur den Quellcode.

Manchmal werden klassische Interpreter-Sprachen von einem Just-in-Time-Compiler übersetzt.

Beispiele für den Einsatz von JIT-Kompilierung:

PHP-Webserver, Bei vielen Interpretersprachen

Konzepte mit Zwischencode

Von Zwischencode spricht man, wenn eine Programmiersprache zunächst in eine andere Sprache und von dort aus in lauffähigen Maschinencode übersetzt wird.

Neben den folgenden, als Beispiele dargestellten Umgebungen verwenden auch die .NET-Sprachen von Microsoft Zwischencode. Zu ihnen gehören unter anderem C# und Visual Basic.

Java

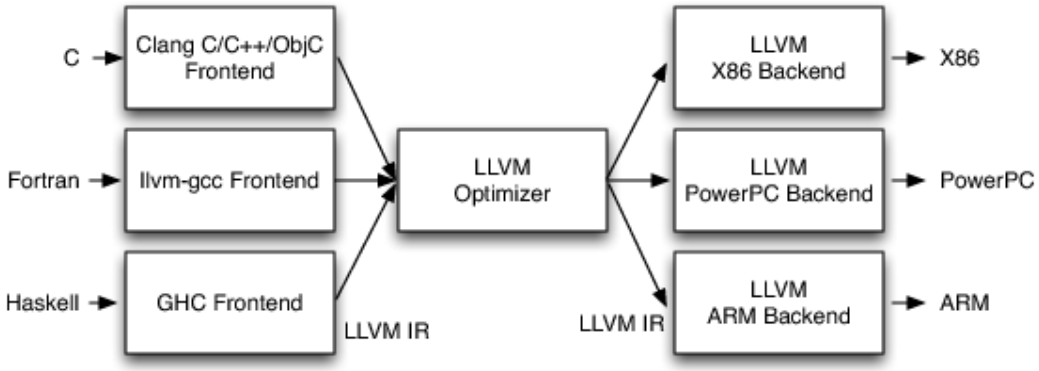
Bei Java von Oracle übersetzt ein Compiler den Quellcode in sogenannten Bytecode, der in einer Laufzeitumgebung läuft, der sogenannten *Java Virtual Machine* (JVM). Das Programmpaket mit der JVM zum Installieren für Anwender heißt *Java Runtime Environment* (JRE).

Bytecode läuft auf jeder Plattform, für die es eine JRE gibt, also auf allen gängigen PC-Betriebssystemen. Viele Programme verlangen die relativ alte JRE 1.8, die zu Java 8 gehört.

Inzwischen ermuntert Oracle die Entwickler dazu, sich nicht auf eine vorhandene JRE zu verlassen, sondern mit jedem Programm die passende Laufzeitumgebung mitzuliefern.

LLVM

Ein anderes Beispiel für die Verwendung von Zwischencode ist das Projekt LLVM. Eine der Einsatzmöglichkeiten besteht darin, C++ und ähnliche Sprachen zunächst in die Zwischensprache LLVM-IR und dann in plattformabhängigen Maschinencode zu kompilieren[1].

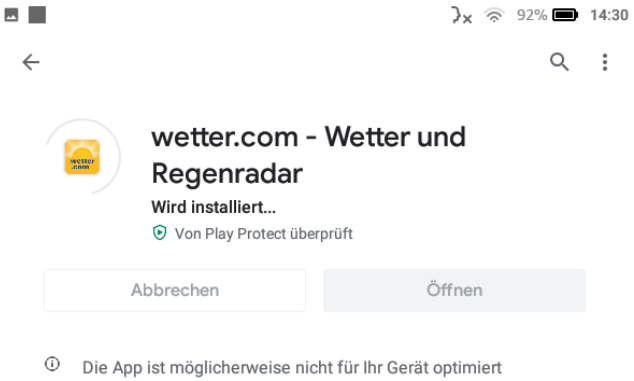


Was muss getan werden, um mit LLVM eine neue Programmiersprache unterstützen zu können?

Es genügt, ein Frontend für eine neue Sprache zu entwickeln.

Java bei Android

Android-Apps sind meistens in Java geschrieben. Die Appstores liefern sie als Bytecode aus. Google hatte bis Android 4 eine eigene JVM verwendet, die ein Just-in-Time-Compiler war. Seit Android 5 dagegen wird der Bytecode beim Installieren auf dem Gerät in Maschinencode kompiliert.



Überlegen Sie, wie sich die benötigte Zeit für die Installation und die Geschwindigkeit zur Laufzeit beim Wechsel von Android 4 nach 5 verändert haben!

Die Installation ist länger, weil kompiliert werden muss. Dafür starten die Programme schneller.

1 Bild: http://www.aosabook.org/images/llvm/LLVMCompiler1.png (Stand 13.12.2015)