Ostfalia
Hochschule für angewandte Wissenschaften

Realisierung eines Compilers für eine prozedurale Sprache

Pascal Ernst

EINLEITUNG

Der Compiler ist ein essenzieller Bestandteil der Informatik und eines der zentralen Werkzeuge von Programmierern. Ein großer Teil moderner Software ist durch ihn generiert worden. Um die Funktionsweise des Compilers näher kennenzulernen, wird ein simpler Compiler in C++ geschrieben. Er kompiliert eine neue C- und Pascal-ähnliche Programmiersprache, Simple Language (Slang), in Maschinencode für die RISC-V-Architektur. Dabei wird das Verständnis für den Aufbau und die Funktionsweise eines Compilers geschaffen und praxisbezogen umgesetzt.

PLANUNG

Der *Simple Language to Instruction Compiler in C++* (SLICC) soll in Eingabe und Ausgabe flexibel sein. Durch die Modularisierung des Projekts in Frontend, Backend und eine Zwischensprache (als Schnittstelle) können für neue Programmiersprachen Frontends und für neue Prozessorarchitekturen Backends geschrieben werden, ohne andere Module anpassen zu müssen. Diese Aufteilung macht den Compiler zu einem Zweipass-Compiler: Er liest und übersetzt den Programmcode in zwei Schritten.

IMPLEMENTATION

Die Trennung von Frontend und Backend wird durch Submodule von *CMake* geregelt, wodurch jedes Modul eine eigene Bibliothek ist, die separat eingebunden werden kann. Für das Slang-Frontend, das den Slang-Programmcode einliest, werden *Flex* als Lexergenerator und *GNU Bison* als Parsergenerator verwendet. Das Frontend parst den Programmcode zu *Drei-Adressen-Code* (eine vereinfachte Repräsentation der Operationen) und erstellt eine Symboltabelle. Im RISC-Backend wird mittels eines handgeschriebenen Code-Generators aus der Zwischensprache RISC-V-Maschinencode generiert.

ERGEBNIS

- Durch Parsergeneratoren erstellte Bottom-up-Parser können mehr Grammatiken parsen als Top-down-Parser, ohne einen wesentlich höheren Implementierungsaufwand zu verursachen.
- Die Unabhängigkeit der einzelnen Module voneinander hängt von der Zwischensprache ab, die sehr einfach oder komplex gestaltet sein kann, und von den Modulen, die die Zwischensprache möglichst "neutral" generieren sollten.
- Anstatt einzelne Backends zu optimieren, kann die Zwischensprache bereits generische Optimierungen enthalten, von denen alle Backends profitieren.