
ALBERT LUDWIGS UNIVERSITÄT FREIBURG

TECHNISCHE FAKULTÄT

PicoC-Compiler

Übersetzung einer Untermenge von C in den Befehlssatz der RETI-CPU

BACHELORARBEIT

Abgabedatum: 28th April 2022

Author:
Jürgen Mattheis

Gutachter:
Prof. Dr. Scholl

Betreuung:
M.Sc. Seufert

Eine Bachelorarbeit am Lehrstuhl für
Betriebssysteme

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Inhaltsverzeichnis

1	Ergebnisse und Ausblick	8
1.1	Compiler	8
1.1.1	Überblick über Funktionen	8
1.1.2	Vergleich mit GCC	8
1.1.3	Showmode	8
1.2	Qualitätssicherung	8
1.3	Erweiterungsideen	8

Abbildungsverzeichnis

1.1	Cross-Compiler als Bootstrap Compiler	9
1.2	Iteratives Bootstrapping	11

Codeverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

1.1	Self-compiling Compiler	8
1.2	Minimaler Compiler	10
1.3	Bootstrap Compiler	10
1.4	Bootstrapping	11

Grammatikverzeichnis

1 Ergebnisse und Ausblick

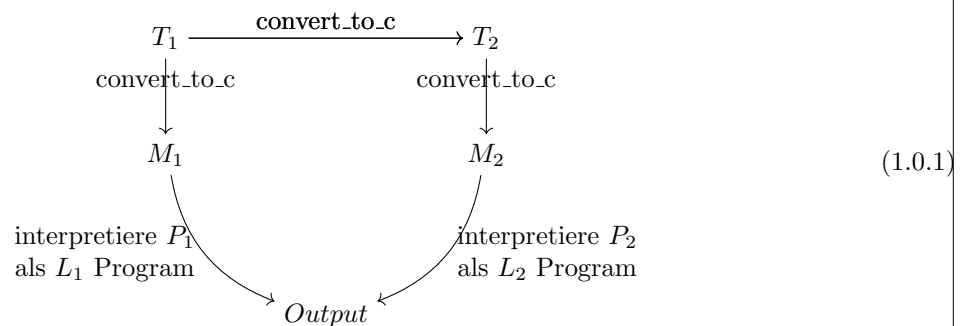
1.1 Compiler

1.1.1 Überblick über Funktionen

1.1.2 Vergleich mit GCC

1.1.3 Showmode

1.2 Qualitätssicherung



1.3 Erweiterungsideen

Wenn eines Tages eine **RETI-CPU** auf einem **FPGA** implementiert werden sollte, sodass ein **provisorisches Betriebssystem** darauf laufen könnte, dann wäre der nächste Schritt einen **Self-Compiling Compiler** $C_{RETI_PicoC}^{PicoC}$ (Definition 1.1) zu schreiben. Dadurch kann die **Unabhängigkeit** von der Programmiersprache L_{python} , in der der momentane Compiler C_{PicoC} für L_{PicoC} implementiert ist und die Unabhängigkeit von einer **anderen Maschine**, die bisher immer für das Cross-Compiling notwendig war erreicht werden.

Definition 1.1: Self-compiling Compiler

Compiler C_w^w , der in der Sprache L_w **geschrieben** ist, die er **selbst** kompiliert. Also ein Compiler, der sich **selbst** kompilieren kann.^a

^aEarley und Sturgis, „A formalism for translator interactions“.

Will man nun für eine Maschine M_{RETI} , auf der bisher keine anderen Programmiersprachen mittels **Bootstrapping** (Definition 1.4) zum laufen gebracht wurden, den gerade beschriebenen **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$ implementieren und hat bereits den gesamten **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$ in der

Sprache L_{PicoC} geschrieben, so stösst man auf ein Problem, dass auf das **Henne-Ei-Problem**¹ reduziert werden kann. Man bräuchte, um den **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$ auf der Maschine M_{RETI} zu kompilieren bereits einen kompilierten **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$, der mit der Maschinensprache B_{RETI} läuft. Es liegt eine **zirkulare Abhängigkeit** vor, die man nur auflösen kann, indem eine **externe Entität** zur Hilfe nimmt.

Da man den gesamten **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$ nicht selbst komplett in der Maschinensprache B_{RETI} schreiben will, wäre eine Möglichkeit, dass man den **Cross-Compiler** C_{PicoC}^{Python} , den man bereits in der Programmiersprache L_{Python} implementiert hat, der in diesem Fall einen **Bootstrapping Compiler** (Definition 1.3) darstellt, auf einer anderen Maschine M_{other} dafür nutzt, damit dieser den **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$ für die Maschine M_{RETI} kompiliert bzw. **bootstrapped** und man den kompilierten **RETI-Maschiendencode** dann einfach von der Maschine M_{other} auf die Maschine M_{RETI} kopiert.²

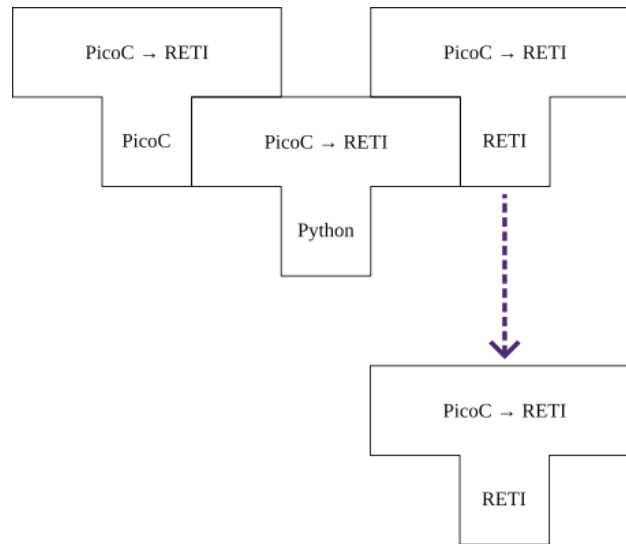


Abbildung 1.1: Cross-Compiler als Bootstrap Compiler

Einen ersten **minimalen Compiler** $C_{2-w.min}$ für eine Maschine M_2 und Wunschsprache L_w kann man entweder mittels eines **externen Bootstrap Compilers** C_w^o kompilieren^a oder man schreibt ihn direkt in der **Maschinensprache** B_2 bzw. wenn ein **Assembler** vorhanden ist, in der **Assemblesprache** A_2 .

Die letzte Option wäre allerdings nur beim allerersten Compiler C_{first} für eine allererste **abstraktere Programmiersprache** L_{first} mit Schleifen, Verzweigungen usw. notwendig gewesen. Ansonsten hätte man immer eine Kette, die beim allerersten Compiler C_{first} anfängt fortführen können, in der ein Compiler einen anderen Compiler kompiliert bzw. einen ersten minimalen Compiler kompiliert und dieser minimale Compiler dann eine umfangreichere Version von sich kompiliert usw.

^aIn diesem Fall, dem **Cross-Compiler** C_{PicoC}^{Python} .

¹Beschreibt die Situation, wenn ein System sich selbst als **Abhängigkeit** hat, damit es überhaupt einen **Anfang** für dieses System geben kann. Dafür steht das Problem mit der **Henne** und dem **Ei** sinnbildlich, da hier die Frage ist, wie das ganze seinen Anfang genommen hat, da beides **zirkular** voneinander abhängt.

²Im Fall, dass auf der Maschine M_{RETI} die Programmiersprache L_{Python} bereits mittels **Bootstrapping** zum Laufen gebracht wurde, könnte der **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$ auch mithilfe des **Cross-Compilers** C_{PicoC}^{Python} als **externe Entität** und der Programmiersprache L_{Python} auf der Maschine M_{RETI} selbst kompiliert werden.

Definition 1.2: Minimaler Compiler

Compiler $C_{w,min}$, der nur die **notwendigsten Funktionalitäten** einer Wunschsprache L_w , wie **Schleifen, Verzweigungen** kompiliert, die für die Implementierung eines **Self-compiling Compilers** C_w^w oder einer **ersten Version** $C_{w_i}^w$ des Self-compiling Compilers C_w^w wichtig sind.^{a,b}

^aDen **PicoC-Compiler** könnte man auch als einen **minimalen Compiler** ansehen.

^bThiemann, „Compilerbau“.

Definition 1.3: Bootstrap Compiler

Compiler C_w^o , der es ermöglicht einen **Self-compiling Compiler** C_w^w zu **bootstrappen**, indem der Self-compiling Compiler C_w^w mit dem **Bootstrap Compiler** C_w^o **kompiliert** wird^a. Der Bootstrapping Compiler stellt die **externe Entität** dar, die es ermöglicht die **zirkulare Abhängigkeit**, dass initial ein **Self-compiling Compiler** C_w^w bereits kompiliert vorliegen müsste, um sich selbst kompilieren zu können, zu brechen.^b

^aDabei kann es sich um einen **lokal** auf der Maschine selbst laufenden Compiler oder auch um einen **Cross-Compiler** handeln.

^bThiemann, „Compilerbau“.

Aufbauend auf dem **Self-compiling Compiler** $C_{RETI_PicoC}^{PicoC}$, der einen **minimalen Compiler** (Definition 1.2) für eine Teilmenge der **Programmiersprache** C bzw. L_C darstellt, könnte man auch noch weitere Teile der Programmiersprache C bzw. L_C für die Maschine M_{RETI} mittels **Bootstrapping** implementieren.³

Das bewerkstelligt man, indem man **iterativ** auf der Zielmaschine M_{RETI} selbst, aufbauend auf diesem **minimalen Compiler** $C_{RETI_PicoC}^{PicoC}$, wie in Subdefinition 1.4.1 den minimalen Compiler schrittweise zu einem immer vollständigeren **C-Compiler** C_C weiterentwickelt.

³Natürlich könnte man aber auch einfach den **Cross-Compiler** C_{PicoC}^{Python} um weitere Funktionalitäten von L_C erweitern, hat dann aber weiterhin eine **Abhängigkeit** von der Programmiersprache L_{Python} .

Definition 1.4: Bootstrapping

Wenn man einen **Self-compiling Compiler** C_w^w einer Wunschsprache L_w auf einer **Zielmaschine** M zum laufen bringt^{a,b,c,d}. Dabei ist die Art von **Bootstrapping** in 1.4.1 nochmal gesondert hervorzuheben:

1.4.1: Wenn man die **aktuelle Version** eines **Self-compiling Compilers** $C_{w_i}^{w_i}$ der Wunschsprache L_{w_i} mithilfe von **früheren Versionen** seiner selbst kompiliert. Man schreibt also z.B. die aktuelle Version des Self-compiling Compilers in der Sprache $L_{w_{i-1}}$, welche von der früheren Version des Compilers, dem Self-compiling Compiler $C_{w_{i-1}}^{w_{i-1}}$ kompiliert wird und schafft es so **iterativ** immer umfangreichere Compiler zu bauen.^{e,f,g}

^aZ.B. mithilfe eines **Bootstrap Compilers**.

^bDer Begriff hat seinen Ursprung in der englischen **Redewendung** „pulling yourself up by your own bootstraps“, was im deutschen ungefähr der aus den **Lügendgeschichten des Freiherrn von Münchhausen** bekannten Redewendung „sich am eigenen Schopf aus dem Sumpf ziehen“ entspricht.

^cHat man einmal einen solchen **Self-compiling Compiler** C_w^w auf der Maschine M zum laufen gebracht, so kann man den Compiler auf der Maschine M weiterentwickeln, ohne von externen Entitäten, wie einer bestimmten Sprache L_o , in der der Compiler oder eine frühere Version des Compilers ursprünglich geschrieben war abhängig zu sein.

^dEinen Compiler in der Sprache zu schreiben, die er selbst kompiliert und diesen Compiler dann sich selbst kompilieren zu lassen, kann eine gute **Probe aufs Exempel** darstellen, dass der Compiler auch wirklich funktioniert.

^eEs ist hierbei theoretisch nicht notwendig den **letzten** Self-compiling Compiler $C_{w_{i-1}}^{w_{i-1}}$ für das Kompilieren des **neuen** Self-compiling Compilers $C_{w_i}^{w_i}$ zu verwenden, wenn z.B. der **Self-compiling Compiler** $C_{w_{i-3}}^{w_{i-3}}$ auch bereits alle Funktionalitäten, die beim Schreiben des **Self-compiling Compilers** C_w^w verwendet werden kompilieren kann.

^fDer Begriff ist sinnverwandt mit dem **Booten** eines Computers, wo die wichtigste Software, der **Kernel** zuerst in den Speicher geladen wird und darauf aufbauend von diesem dann das Betriebssysteme, welches bei Bedarf dann **Systemsoftware**, Software, die das Ausführen von Anwendungssoftware ermöglicht oder unterstützt, wie z.B. Treiber, und **Anwendungssoftware**, Software, deren Anwendung darin besteht, dass sie dem Benutzer unmittelbar eine Dienstleistung zur Verfügung stellt, lädt.

^gEarley und Sturgis, „A formalism for translator interactions“.

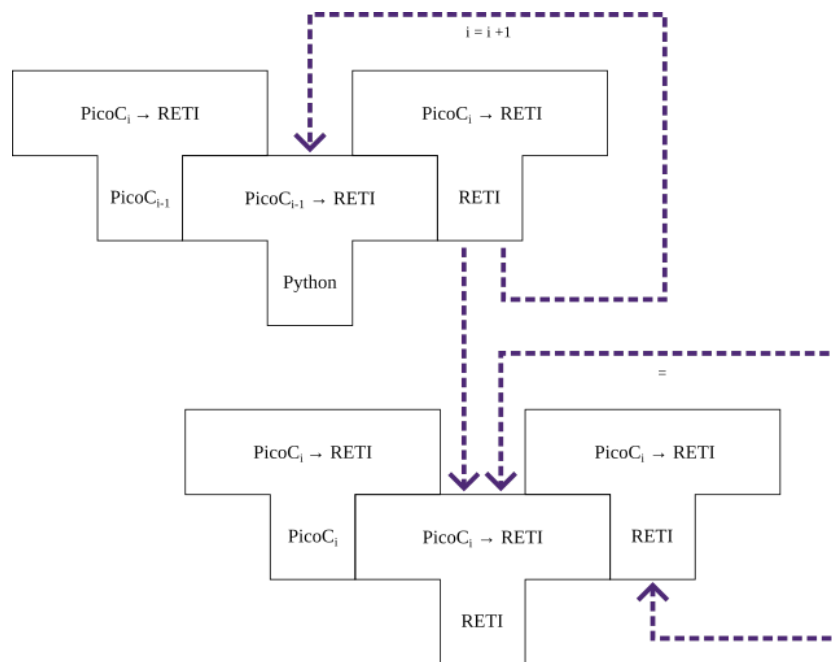


Abbildung 1.2: Iteratives Bootstrapping

Auch wenn ein **Self-compiling Compiler** $C_{w_i}^{w_i}$ in der Subdefinition 1.4.1 selbst in einer früheren Version $L_{w_{i-1}}$ der Programmiersprache L_{w_i} geschrieben wird, wird dieser nicht mit $C_{w_{i-1}}^{w_{i-1}}$ bezeichnet, sondern

mit $C_{w_i}^{w_i}$, da es bei **Self-compiling Compilern** darum geht, dass diese zwar in der Subdefinition 1.4.1 eine frühere Version $C_{w_i-1}^{w_i-1}$ nutzen, um sich selbst kompilieren zu lassen, aber sie auch in der Lage sind sich selber zu kompilieren.

Literatur

Artikel

- Earley, J. und Howard E. Sturgis. „A formalism for translator interactions“. In: *CACM* (1970). DOI: [10.1145/355598.362740](https://doi.org/10.1145/355598.362740).

Vorlesungen

- Thiemann, Peter. „Compilerbau“. Vorlesung. Vorlesung. Universität Freiburg, 2021. URL: <http://proglang.informatik.uni-freiburg.de/teaching/compilerbau/2021ws/> (besucht am 09.07.2022).