(Bild: Chat Karen Studio – Shutterstock)

Sonderdruck

Formal verifiziert:

# Der perfekte Compiler

**Genauso wie Software-Programme sind auch Compiler fehleranfällig und können aus korrektem Quellcode fehlerhaften Maschinencode machen. Nicht so der CompCert-Compiler, dessen korrekte Funktion mathematisch bewiesen ist und der beim Übersetzen auch noch den Code optimiert.**

Von Jörg Barrho, Sandrine Blazy, Christian Ferdinand, Daniel Kästner, Xavier Leroy, Marc Schlickling, Michael Schmidt, Bernhard Schommer und Ulrich Wünsche

Ein Compiler übersetzt ein in einer höheren Programmiersprache geschriebenes Quellprogramm in ausführbaren Maschinencode des Zielprozessors. Dieser Übersetzungsprozess, insbesondere die Code-Erzeugungs- und Code-Optimierungsphase, ist hochkomplex und dadurch fehleranfällig. Zahlreiche unabhängige Studien dokumentieren Fehler in allen untersuchten Compilern. Bei einer Fehlcompilierung erzeugt der Compiler ohne Warnung fehlerhaften Maschinencode. In sicherheitskritischen Systemen ist dies ein ernstes Problem, das zu fehlerhaftem Programmverhalten führen und insbesondere Speicherverletzungen und Programmabstürze verursachen kann. Zudem werden viele Verifikationsaktivitäten auf Modell- oder Quellcode-Ebene durchgeführt. Bei einer möglichen Fehlcompilierung haben diese Nachweise jedoch keine Aussagekraft für den vom Compiler generierten ausführbaren Code. Viele Sicherheitsnormen erfordern daher zusätzliche aufwendige Maßnahmen, um zu zeigen, dass die Anforderungen, die schon auf höheren Ebenen nachgewiesen wurden, auch auf Ebene des ausführbaren Objektcodes gelten.

Der CompCert-Compiler ist seit 2015 kommerziell verfügbar. CompCert wurde durch maschinenunterstützte mathematische Nachweise als korrekt bewiesen, was insbesondere das Auftreten von Fehlcompilierungen ausschließt. Der Beweis zeigt, dass der vom Compiler produzierte Maschinencode dasselbe Verhalten aufweist, wie es die Semantik des C-Quellprogramms vorgibt. CompCert ist der erste kommerziell erhältliche formal verifizierte Compiler und bietet einen bislang unerreichten Vertrauensgrad in die Korrektheit des Übersetzungsprozesses. Quellcode und Beweise sind frei zugänglich und ermöglichen eine unabhängige Prüfung. Akademische Lizenzen sind kostenfrei, zur kommerziellen Nutzung ist eine von AbsInt erhältliche kommerzielle Lizenz erforderlich.

Kosten zum Aufspüren und Umgehen – oder Beheben – von Compilerfehlern und Auslieferung von Patches an die Endkunden des eingebetteten Systems können vermieden werden. Der Testaufwand zum Nachweis, dass bereits auf höheren Ebenen überprüfte Eigenschaften auch auf Maschinencode-Ebene gelten, kann reduziert werden, da dies automatisch aus dem Korrektheitsbeweis von CompCert folgt. In Bereichen, wo bislang aus Sicherheitsgründen Compiler-Optimierungen ganz oder teilweise abgeschaltet wurden, können erstmals Compiler-Optimierungen genutzt werden.

## So geht ein Compiler vor

Die Eingabe eines Compilers besteht aus einer Menge von C Source- und Headerdateien. CompCert fokussiert auf den eigentlichen Übersetzungsvorgang und enthält weder Präprozessor, noch Assembler und Linker. Er muss daher in Kombination mit einer konventionellen Compiler-Toolchain eingesetzt werden. CompCert liest die extern präprozessierten C-Dateien ein, führt

eine Reihe von Codeerzeugungs- und -optimierungsphasen aus und erzeugt Assemblerdateien mit Debug-Informationen. Anschließend erzeugen der externe Assembler und Linker den ausführbaren Maschinencode. Zur Erhöhung des Vertrauens in die Korrektheit von Assemblieren und Linken liefert CompCert ein Werkzeug zur Translation Validation (genannt Valex) mit, das eine Äquivalenzprüfung zwischen CompCert-Assemblercode und dem ausführbaren Code durchführt (**Bild 1**).

CompCert besteht aus 20 Übersetzungsschritten, die den C-Quellcode schrittweise in Maschinencode übersetzen. Sie können in vier Phasen gegliedert werden:

- Die erste Phase beginnt mit dem Aufruf eines externen Off-the-Shelf-Präprozessors (z.B. GCC). Der präprozessierte Code wird vom formal verifizierten CompCert-Parser in einen abstrakten Syntaxbaum (AST) transformiert.
- Die zweite Phase legt die Auswertungsreihenfolge fest, vereinfacht die Codestruktur und macht implizite C-Operationen explizit.
- Die dritte Phase umfasst alle Code-Optimierungen. Hierzu zählen Registerallokation, Funktionsinlining, Codeselektion, Konstantenfaltung, Entfernung gemeinsamer Teilausdrücke und Redundanzeliminierung. Die Ausgabe des Backend-Compilers ist ein abstrakter Syntaxbaum in generischer Assemblersprache.
- Die letzte Phase liest den abstrakten Assembler-Syntaxbaum ein, erzeugt die konkrete Assembler-Syntax, fügt DWARF2-Debug-Information hinzu und ruft den externen Assembler und Linker auf, um den ausführbaren binären Maschinencode zu generieren.

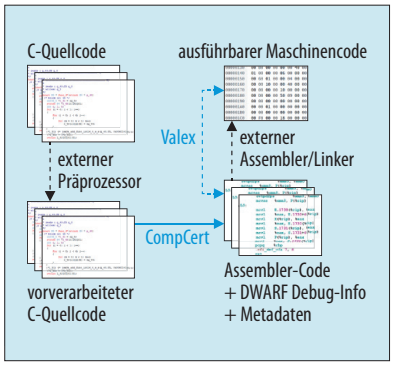


**Bild 1. Der CompCert-Compiler greift auf externen Präprozessor, Linker und Assembler zurück und liefert mit Valex ein Werkzeug zur Translation Validation, um die Äquivalenzprüfung zwischen CompCert-Assemblercode und ausführbarem Code durchzuführen.** (Bilder: AbsInt)

Der Translation Validator Valex weist nach, dass abstrakter Assemblercode und erzeugter Binärcode äquivalent sind.

## Vertrauen ist gut, Kontrolle ist besser

Für alle CompCert-Übersetzungsschritte ist formal bewiesen, dass keine Fehlcompilierung auftreten kann. Formal wird die Erhaltung der Semantik zwischen der Eingabe und der Ausgabe jeden Übersetzungsschrittes bewiesen. Hierzu werden formale Semantiken für C, jede verwendete Zwischensprache und den abstrakten Assembler definiert, die alle möglichen Verhaltensweisen des Programmes beschreiben. Die Verhaltensbeschreibung beinhaltet die Terminierung (normales Programmende, Abbruch durch Laufzeitfehler oder Endlosausführung) sowie einen Trace aller beobachtbaren Eingabe-/Ausgabeoperationen des Programms.

Der Beweis der Semantikbewahrung ist aufgrund der Vielzahl von Übersetzungsschritten sehr komplex und wurde daher maschinengestützt unter Verwendung des Theorembeweisers Coq durchgeführt.

Im Fall von Laufzeitfehlern durch undefiniertes oder unspezifiziertes Verhaltens gemäß C-Standard – Beispiele hierfür sind Division durch 0, Pufferüberläufe oder Out-of-Bounds Array-Zugriffe – ist die Semantik des Quellprogramms nicht wohldefiniert. Daher kann der Korrektheitsbeweis von CompCert durch einen Nachweis ergänzt werden, dass keine solchen Laufzeitfehler im Quellprogramm enthalten sind, z.B. durch Einsatz des sicheren statischen Analysators Astrée.

## Einsatz im Kernkraftwerk

Im Folgenden berichten wir über praktische Erfahrungen bei der Ersetzung eines traditionellen Compilers durch CompCert, die bei MTU Friedrichshafen im Entwicklungsprozess eines hochsicherheitskritischen Steuerungssystems gewonnen wurden. Es handelt sich um die Steuerung von Dieselmotoren in Notstromaggregaten für Kernkraftwerke. Die von MTU entwickelte digitale Engine Control Unit (ECU) führt ausschließlich Sicherheitsfunktionen aus. Sie hält den vom Betreiber angeforderten sicheren Zustand aufrecht: Entweder steht die Maschine still, oder sie läuft mit der angeforderten Geschwindigkeit.

Der Anwendungsteil der ECU-Software besteht aus handgeschriebenem C-Code und vom Scade-Codegenerator erzeugtem C-Code. Software und Entwicklungsprozess sind konform zu den internationalen Standards IEC 60880 und IEC 61508:2010, Part 3 (SCL3 für Software). Der TÜV Rheinland hat die Notstromaggregate und den Entwicklungsprozess und damit auch die Werkzeugkette dementsprechend zertifiziert.

Der C-Code wird in einer Teilmenge von C99 erzeugt, bei der Robustheit und die Vermeidung fehleranfälliger Konstrukte im Vordergrund stehen. Als Ausgangspunkt wurde der MISRA C:2004-Standard verwendet und um eigene In-House-Codierrichtlinien erweitert. Zur automatischen Prüfung der Codierrichtlinien wird das Werkzeug AbsInt RuleChecker eingesetzt. Er wurde um die zusätzlichen MTU-spezifischen Regeln erweitert und erreicht insgesamt eine Abdeckung von 85 % des MTU-Regelsatzes. Die verbleibenden 15 % der Regeln erfordern menschliche Interaktion; sie zielen z. B. auf die Auswahl verständlicher Bezeichner-Namen und das Einfügen hilfreicher Kommentare.

Das Compilieren des Quellcodes für Sicherheitsfunktionen im Produktionseinsatz ist eine kritische Aktivität: Jedes hierfür eingesetzte Werkzeug muss gemäß einer Reihe von Kriterien qualifiziert werden. MTU setzt ausschließlich Tools für Safety-Anwendungen ein, die in einem strukturierten Entwicklungsprozess entstehen. Es muss ausreichende Nachweise geben, dass das Tool verlässlich arbeitet, und die Entwickler müssen mit dem Tool zufrieden sein. Die Qualifizierungsstrategie von MTU ist in **Bild 2** veranschaulicht.

## Früher: kontinuierliche Fehlerbehebung

In der Vergangenheit wurde bei MTU ein traditioneller kommerzieller C-Compiler genutzt, der verbreitet im Einsatz war. Aufgrund des sporadischen Auftretens neuer Fehler verursachte der Einsatz dieses Compilers kontinuierlichen Aufwand. Jeder gemeldete Compilerfehler muss untersucht werden, und gegebenenfalls sind Code-Ände-

rungen und Anpassungen der Checklisten für Code Reviews nötig. Läuft der Herstellersupport für einen solchen Proven-in-use-Compiler aus, gibt es zwei Alternativen: Der Hersteller prüft, ob Compilerfehler, die in aktuellen Versionen gefunden werden, auch schon in der alten Version existierten – was mit erheblichen Kosten verbunden sein kann. Alternativ kann der Anwender entscheiden, eine neuere Version des Compilers mit einer kommerziellen Validierungssuite zu validieren. Auch dies ist mit erheblichem Aufwand und externen Kosten verbunden. Keine dieser Lösungen ist zufriedenstellend.

## Integration von CompCert in den Entwicklungsprozess

Beim Wechsel auf CompCert mussten lediglich kleine Anpassungen des Build-Prozesses an die Compiler-Optionen von CompCert und Änderungen der Steuerungsdatei für den Linker bezüglich der Allokation von Speichersegmenten vorgenommen werden.

Wie zuvor beschrieben, wird nach C99-Standard undefiniertes Verhalten nicht vom formalen Korrektheitsbeweis von CompCert abgedeckt. Daher wird der sichere Laufzeitfehler-Analysator Astrée eingesetzt, um die Abwesenheit undefinierten Verhaltens und von Laufzeitfehlern nachzuweisen. Astrée basiert auf abstrakter Interpretation, einer formalen Methode zur statischen Programmanalyse, die es ermöglicht, zu beweisen, dass kein Laufzeitfehler übersehen wird. Beispiele der von Astrée abgedeckten Fehler sind Division durch Null, Out-of-Bounds Array-Zugriffe, Pufferüberläufe, hängende Zeiger, Datenwettläufe und Deadlocks.

## Reduzierter Testaufwand

Werden Software-Komponenten so spezifiziert, dass sie auch in Grenzbereichen vollständig definiertes und nicht-widersprüchliches Verhalten aufweisen und sind sie möglichst generisch implementiert, werden abstrakte Modul- oder Software-Tests möglich. Mit generischem Verhalten ist gemeint, dass es von Prozessoreigenschaften wie Endianness und Hardware-Registerverteilung unabhängig ist. Codierrichtlinien und Architektureinschränkungen können die Einhaltung solcher Regeln sicherstellen. Entsprechen Software-Artefakte diesen Anforderungen, können sie unabhängig von der Hardware und einer bestimmten Compiler-Toolchain getestet werden.

CompCert ist verfügbar für ARM, x86 und PowerPC-Architekturen. Aufgrund der bewiesenen Korrektheit des Übersetzungsvorgangs gelten Eigenschaften, die auf einer Plattform erfüllt sind, auch auf den anderen. Der von CompCert generierte ausführbare Code wurde durch Valex geprüft. Der generierte Code wurde in die Zielhardware integriert und in einer simulierten synthetischen Umgebung getestet, was die Voraussetzung zur Verwendung der Software auf dem echten Motor darstellt. Falls Simulatortest und Motortest erfolgreich sind, bieten sie zusammen eine Validierungsabdeckung aller Aspekte der funktionalen Systemanforderungen.

Alle Buildprozesse wurden erfolgreich abgeschlossen, alle funktionalen Tests waren erfolgreich. Insbesondere haben diese Tests – wie erwartet – auch keinen Hinweis auf Compilerfehler gezeigt.

## Performance des generierten Codes

Zur Bewertung der Leistungsfähigkeit des von CompCert generierten Codes wurden die Codegröße, der maximale Stackverbrauch und die maximale Ausführungszeit (Worst-Case Execution Time, WCET) untersucht.

Beim traditionellen Compiler waren alle Optimierungen abgeschaltet, um die Nachvollziehbarkeit des Compilierens zu wahren und die funktionalen Risiken zu vermeiden, die durch den Compiler selbst in der Code-Optimierungsphase eingeführt werden. Aufgrund des formalen Korrektheitsbeweises von

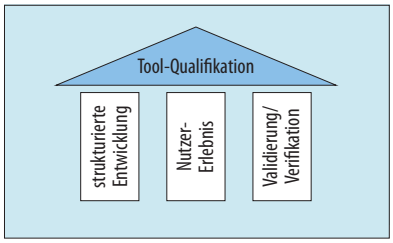


**Bild 2. Nach diesen Kriterien setzt MTU Friedrichshafen Software-Werkzeuge ein.**

CompCert konnte entschieden werden, diese Einschränkung aufzuheben. Insbesondere ist bei CompCert die Registerallokation aktiviert, um Speicherzugriffe zu reduzieren.

Im Vergleich zum zuvor eingesetzten konventionellen Compiler wurde die Größe des Codesegments reduziert. Die Größe des Datensegmentes ist in beiden Fällen fast identisch.

Durch Einsatz des statischen Analysators aiT werden sichere obere Schranken der maximalen Ausführungszeit (Worst-Case Execution Time, WCET) jeder synchronen Funktion und jedes Interrupthandlers bestimmt. Daraus

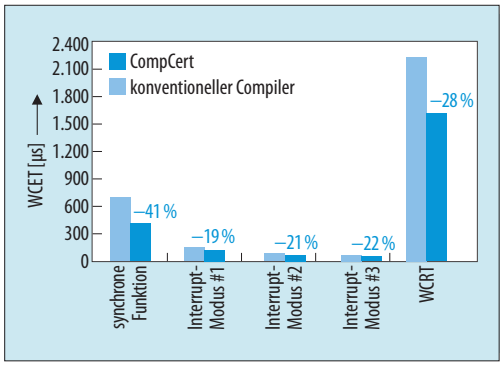


**Bild 3. Beim Vergleich der Worst Case Execution Time (WCET) wurden beim konventionellen Compiler die Optimierungen abgeschaltet, um funktionale Risiken zu vermeiden.**

wird die maximale Antwortzeit (Worst-Case Response Time, WCRT) eines Berechnungszyklus ermittelt. Die Ergebnisse für die MTU-Applikation sind in **Bild 3** gezeigt.

Die berechneten WCET-Schranken zeigen eine um 28 % niedrigere Prozessor-Auslastung als mit dem konventionellen Compiler. Der Hauptgrund hierfür ist die verbesserte Speicherperformance. Zudem hat sich der maximale Stackverbrauch um etwa 40 % reduziert.

## Tool-Qualifizierung

Wie zuvor beschrieben, basiert die Qualifizierungsstrategie von MTU auf drei Säulen, nämlich dem Nachweis eines strukturierten Entwicklungsprozesses, der positiven Benutzererfahrung, und dem Nachweis zuverlässiger Ausführung durch Validierung. Diese Strategie wurde auch auf CompCert angewendet.

Im Gegensatz zu der bisher üblichen, sehr aufwendigen Validierung der Übereinstimmung von Quelldatei und vollständig gelinkter ausführbarer Binärdatei wird die Qualifizierung der Compiler-Toolchain in drei Phasen aufgeteilt: formale Verifikation der Compilierung, traditionelle Testsuite-Validierung und Translation Validation (**Bild 4**).

### Compilierung

Die formalen CompCert-Beweise zeigen, dass keine Fehlcompilierung auftreten kann und liefern einen hohen Vertrauensgrad in das Frontend und Backend von CompCert. Insbesondere umfasst dies alle Compiler-Optimierungen, die durch traditionelle Validierungsmethoden besonders schwer zu qualifizieren sind. Der formale Beweis deckt einige Elemente der Parsing-Phase nicht ab und er deckt nicht das Präprozessieren, Assemblieren und Linken ab, wozu externe Tools verwendet werden. Daher wird der formale Beweis durch eine Validierungssuite ergänzt.

### Präprozessor

Zum Präprozessieren des Quellcodes wird eine weit verbreitete Version von GCC eingesetzt. Diese Version wurde durch eine Präprozessor-Testsuite validiert, für die manuell nachgewiesen wurde, dass sie den von den MTU-Codierrichtlinien vorgegebenen Sprachumfang voll abdeckt. Um sicherzustellen, dass die Quelldateien die Codierrichtlinien tatsächlich einhalten, wird der AbsInt RuleChecker eingesetzt.

### Assemblieren und Linken

Cross-Assemblieren und Cross-Linking wird ebenfalls mit GCC durchgeführt. Zur Ergänzung des Proven-In-Use-Arguments und der impliziten Abdeckung durch die Validierungssuite wird das Translation Validation Tool Valex verwendet, das zusätzliches Vertrauen in die Korrektheit von Assembler und Linker herstellt.

Alle Tools, die im Qualifizierungsprozess von CompCert verwendet werden – Astrée, RuleChecker und Valex – wurden ebenfalls gemäß oben dargestellter Strategie qualifiziert. Durch die mitgelieferten Qualification Support Kits und Qualification Software Life Cycle Data Reports zum Nachweis eines strukturierten Entwicklungsprozesses wird die Tool-Qualifizierung erheblich erleichtert.

Durch Aufteilung der Qualifizierung von CompCert in Einzelschritte und Anwendung strikter Codierrichtlinien während der gesamten Entwicklung kann die Komplexität der Compiler-Qualifizierung deutlich reduziert werden. Dies ist ein wichtiger Pluspunkt des Einsatzes von CompCert für hochsicherheitskritische Industrieanwendungen.

## Feuertaufe bestanden

CompCert ist ein formal verifizierter optimierender C-Compiler. Der erzeugte Maschinencode verhält sich genau, wie die Semantik des C-Programmes vorgibt. CompCert lässt sich mit wenig Aufwand in existierende Build-Prozesse integrieren, ermöglicht ein noch nie dagewesenes Vertrauen in die Korrektheit des generierten Codes, eine deutliche Reduktion des Qualifizierungsaufwands und eine spürbar verbesserte Systemperformance. Diese Vorteile konnten bei MTU Friedrichshafen im Rahmen der Ersetzung eines konventionellen Compilers durch CompCert für eine hochsicherheitskritische Steuerungs-Software für Notstromaggregate im realen Industrieeinsatz belegt werden. *jk*

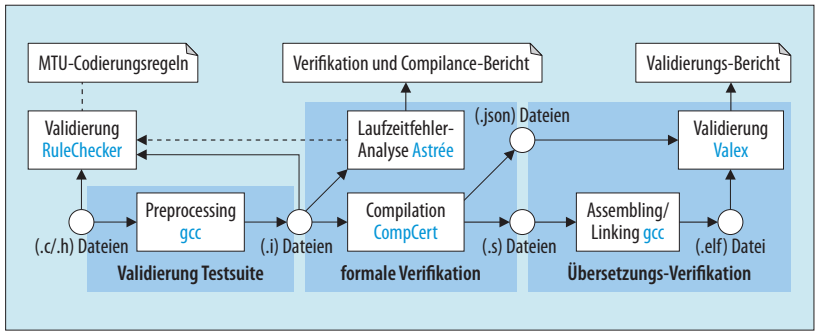


**Bild 4. Im Gegensatz zu der bisher üblichen, sehr aufwendigen Validierung der Übereinstimmung von Quelldatei und vollständig gelinkter ausführbarer Binärdatei wird die Qualifizierung der Compiler-Toolchain in drei Phasen aufgeteilt.**

**Dr. Daniel Kästner**
studierte Informatik und BWL an der Universität des Saarlandes und promovierte über Codeoptimierung für eingebettete Prozessoren.
Dr. Kästner ist einer der Mitgründer der AbsInt Angewandte Informatik GmbH und seit 2003 Leiter der technischen Entwicklung. Er ist Autor einer Vielzahl wissenschaftlicher Publikationen aus den Bereichen Funktionale Sicherheit, Cybersecurity, Programmanalyse, Laufzeitfehleranalyse, Codeerzeugung und -optimierung.
kaestner@absint.com