

Ein SIL-3-konformes Werkzeug zur formalen Verifikation von C-Programmen

Dennis Walter

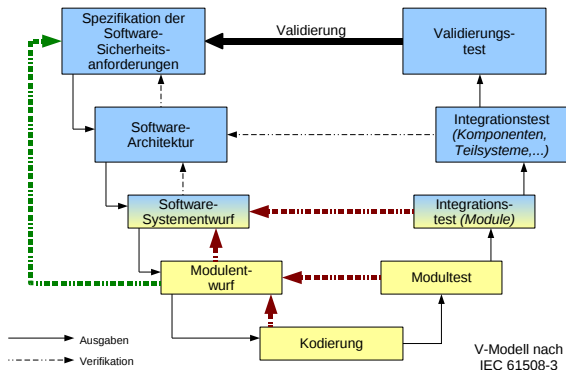
Deutsches Forschungszentrum für Künstliche Intelligenz, Bremen

Universität Bremen, Cartesium, 13.10.09



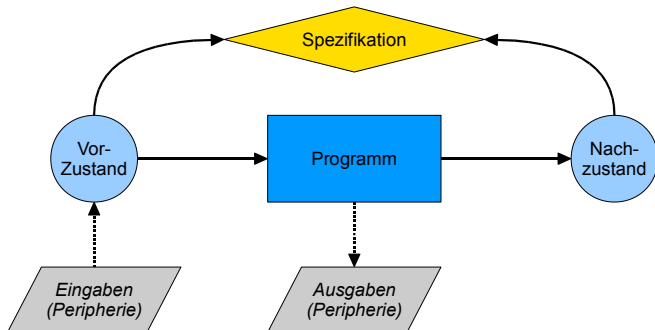
- Modell: Klassische analytische Geometrie
- Viele geometrische Operationen im Code
 - Geeignete Spezifikationsmethode: detaillierte funktionale Spezifikation
- Korrektheit der Berechnungen essentieller Bestandteil der SW-Integrität
 - Weder Spezifikation noch Code “offensichtlich” richtig
 - Rigoroser mathematischer Ansatz geboten
 - Schritt Detailspezifikation \Rightarrow Code eröffnet Fehlermöglichkeiten
- Formale Korrektheitsaussage möglich
 - Gute Formalisierbarkeit des Problems
 - Aussagen mit modernen interaktiven Werkzeugen beweisbar

- Entwurf einer verständlichen, formalen Sprache zur **Spezifikation** von (MISRA-)C-Code
 - Augenmerk auf “geometrischen” Code
- Entwicklung von Werkzeugunterstützung für **formale Verifikation**
 - Keine reinen “Papier & Bleistift”-Beweise
 - Nicht auf stichproben-basiertes Testen verlassen
- Nachweis der Anwendbarkeit für SW-Systeme nach **IEC 61508 SIL-3**
- Formalisierung im Theorembeweiser Isabelle
 - Anwendungsdomäne — Programme — Spezifikationen
- Verifikation des SAMS-Codes

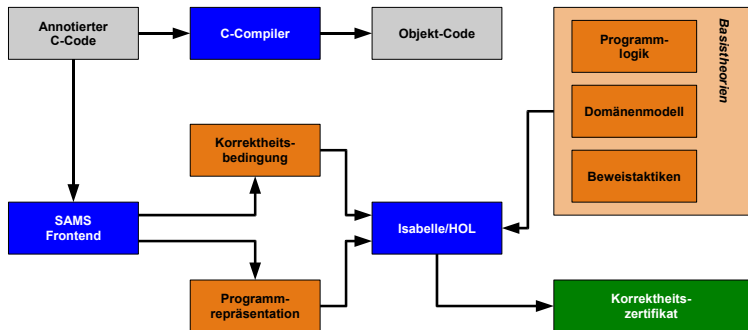


- Folgt **Design by Contract**-Prinzip
 - C-Funktionen mit Vor-/Nachbedingungen versehen
 - Vorbedingung: Bei Aufruf der Funktion sicherstellen
 - Nachbedingung: Erfüllt nach Ausführung der Funktion
- **Funktionale Spezifikation**: Aussagen über Programmmzustände
 - Direkter Bezug zum Code
 - $x < y, \{a. 'x \leq a \leq 'y\} \cap M = \{\}$
- Formale Sprache für Spezifikationen **integriert Domänenmodell und Programmdaten**
- Hilfskonzepte: Definitionen von Prädikaten, Präprozessor-Konstanten, ...

- Spezifikation ist **Relation von Vor- und Nachzuständen**
- Spezifikationen erlauben Aussagen wie:
 - `foo()` liefert sichere Schätzung des Bremsweges gemäß SS Bremsmodell
 - Nothalt wird ausgelöst bei Überschreitung maximaler Geschwindigkeit
 - `sin()` ändert den Zustand von Variablen nicht
- Keine Aussagen möglich über:
 - Laufzeit von Programmen
 - Ein-/Ausgabeverhalten mit Peripheriegeräten
 - Hardware-bedingtes Fehlverhalten



... mehr zu Spezifikationen im nächsten Vortrag



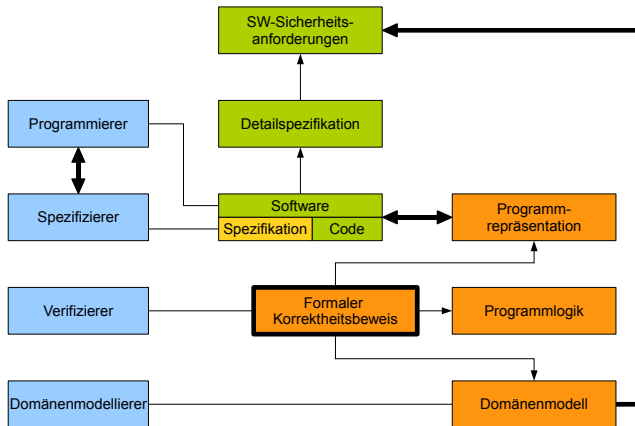
Isabelle-Theorie



Werkzeug



Beweisskript &
PDF-Dokument



Übersetzer

- prüft C-Code nach ISO 9899:1990
- prüft Syntax/Typen von Spezifikationen
- generiert Isabelle-Theorien
 - Programmrepräsentation (mit Spezifikationen)
 - Korrektheitsbedingung (eine pro Funktion)

MISRA-Checker

- Stellt Einhaltung der MISRA-Regeln sicher
- Ist konfigurierbar: Teilmenge prüfen, Zeilen ignorieren

Umfang der Formalisierung (in Isabelle)

- Modell des **Programmzustands**
 $\Sigma : BLoc \rightarrow Type \times (nat \rightarrow Val)$
- Programme und Spezifikationen als **Datentypen**
datatype *stmt* = *AssignStmt lv expr*
 | *WhileStmt expr stmt* | ...
- Formalisierung der **Semantik** von MISRA-C-Programmen
 $\llbracket stmt \rrbracket : Env \rightarrow \Sigma \rightarrow (\Sigma + exn)$
 $\llbracket expr \rrbracket : Env \rightarrow \Sigma \rightarrow (\Sigma \times Val + exn)$
- **Programmlogik** (Beweisregeln) zur Verifikation
- Beweistaktiken zur **Automation**

- Realisiert formale Verifikation im Theorembeweiser Isabelle
 - “Handwerkszeug” des Verifizierers
- Zugrundeliegende Verifikationsmethode: Hoare-Logik
 - Quellcode-Spezifikationen übersetzt in Prädikate der Logik
- Ein Korrektheitsbeweis pro Programmfunktion
- Mischung aus interaktivem und automatischem Beweis
 - **apply** $t_1 \cdots$ **apply** t_{100} **done** ggü. **by** auto

- Nachweis der Gültigkeit des Hoare-Tripels $\Gamma, \Delta \vdash_s [P] c [Q]$
- **lemma** “*function-correctness sams-transformiere global-env*”

- Nachweis der Gültigkeit des Hoare-Tripels $\Gamma, \Lambda \vdash_s [P] c [Q]$
- **lemma** “*function-correctness sams-transformiere global-env*”
 - Funktionale Korrektheit
$$\forall \sigma \sigma' v. P \sigma \longrightarrow \llbracket c \rrbracket \Gamma \sigma = \llbracket (\sigma', v) \rrbracket \longrightarrow Q \sigma'$$

- Nachweis der Gültigkeit des Hoare-Tripels $\Gamma, \Lambda \vdash_s [P] c [Q]$
- **lemma** “*function-correctness sams-transformiere global-env*”
 - Funktionale Korrektheit
$$\forall \sigma \sigma' v. P \sigma \longrightarrow \llbracket c \rrbracket \Gamma \sigma = \llbracket (\sigma', v) \rrbracket \longrightarrow Q \sigma'$$
 - Programmtermination
$$\forall \sigma. P \sigma \longrightarrow \exists \sigma' v. \llbracket c \rrbracket \Gamma \sigma = \llbracket (\sigma', v) \rrbracket$$

- Nachweis der Gültigkeit des Hoare-Tripels $\Gamma, \Lambda \vdash_s [P] c [Q]$
- **lemma** “*function-correctness sams-transformiere global-env*”
 - Funktionale Korrektheit
$$\forall \sigma \sigma' v. P \sigma \longrightarrow \llbracket c \rrbracket \Gamma \sigma = \llbracket (\sigma', v) \rrbracket \longrightarrow Q \sigma'$$
 - Programmtermination
$$\forall \sigma. P \sigma \longrightarrow \exists \sigma' v. \llbracket c \rrbracket \Gamma \sigma = \llbracket (\sigma', v) \rrbracket$$
 - Programmsicherheit: Feldindizes, “wilde” Zeiger, Nulldivision

- Nachweis der Gültigkeit des Hoare-Tripels $\Gamma, \Lambda \vdash_s [P] c [Q]$
- **lemma** “*function-correctness sams-transformiere global-env*”
 - Funktionale Korrektheit
$$\forall \sigma \sigma' v. P \sigma \longrightarrow \llbracket c \rrbracket \Gamma \sigma = \llbracket (\sigma', v) \rrbracket \longrightarrow Q \sigma'$$
 - Programmtermination
$$\forall \sigma. P \sigma \longrightarrow \exists \sigma' v. \llbracket c \rrbracket \Gamma \sigma = \llbracket (\sigma', v) \rrbracket$$
 - Programmsicherheit: Feldindizes, “wilde” Zeiger, Nulldivision
 - Begrenztheit von Zustandsänderungen:
Nur Speicherstellen in Λ wurden modifiziert

Verifikation in SAMS

- Beispiel `hilfsfunktionen.c`
 - 16 Funktionen, zw. 4 u. 104 Zeilen, durchschn. 18 Zeilen
- Beispiel `schutzfeld_berechnen.c`
 - 6 Funktionen, zw. 26 u. 127 Zeilen, durchschn. 53 Zeilen
- Insgesamt ~ 50 Funktionen
- Volumen: 200kB (`.c/.h` Dateien)
- Längste Spezifikation (`schutzfeld_berechnen()`): 52 Zeilen
- Verifikationsdauer pro Funktion: zw. 30 Minuten und mehreren Wochen

- A.4.1c Formale Methoden
- A.4.2 Rechnergestützte Entwurfsmethoden
- A.4.5 Entwurfs- und Codierungs-Richtlinien (deckt B.1 ab)
- A.4.6 Strukturierte Programmierung
- A.9.1 Formaler Beweis
- A.9.3 Statische Analyse
- B.8.{1,3,4,5,8} u.a.: Grenzwertanalyse, Steuerflussanalyse, Symbolische Ausführung

- 1 Zertifizierung des SAMS-Codes mithilfe der SAMS-Verifikationsumgebung
 - SIL-3 erreicht für Code und Verifikationsumgebung
 - Formale Beweise von echtem, realistischem Programm
- 2 SAMS-Verifikationsumgebung als eigenständiges Projektergebnis
 - Allgemeines Domänenmodell und Programmlogik für weitere Projekte einsetzbar
- 3 Reichweite der Verifikation
 - Funktionale Korrektheit
 - Arbeit auf Modulebene, mit formalem Bezug zur Si.-Anf.-Spezifikation