JOHANNES KEPLER
UNIVERSITY LINZ | JKU

Was ist ein „Computerprogramm“?

Michael Sonntag

Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM)

Johannes Kepler Universität Linz, Österreich

sonntag@fim.uni-linz.ac.at

Problemaufriss

- Ist nicht klar, was ein Computerprogramm ist?
  - In C, Java, … geschrieben und wird kompiliert oder interpretiert!
- Nicht unbedingt, denn was ist mit:
  - Shell Scripts? Funktionale/Logische Programmiersprachen?
  - Programme mit schweren Fehlern (Syntax, nicht Semantik!)?
  - Unvollständige Programme bzw. Klassenbibliotheken?
  - Schnittstellen?
  - Datenbankstrukturen?
    - Motivation für die Untersuchung aufgrund konkreter Anfrage (Buchhaltungs-SW)!
  - Konfigurationsdateien?
  - VHDL: Nur-Simulation oder Synthesefähig?

Noch vergleichsweise einfach!

Definition für „Computerprogramm“ (1)

- Gesetz/Richtlinien: Leider nein! Technische Quellen: Diverse!
  - Problemlösungsverfahren in für Computer verständlicher Form
  - Algorithmus einschließlich Datenobjekten in für Computer verständlicher Form einer Programmiersprache. Ist auf einem Computer ausführbar.
  - Verfahren, Daten durch Algorithmen zu manipulieren für gewünschtes Ergebnis. Formulierung in einer versteh- und abarbeitbaren Sprache.
  - Syntaktische Einheit aus Anweisungen und Vereinbarungen zur Lösung einer Aufgabe
- Beruhen alle auf „Programmiersprache“ → Auch hier gibt es Definitionen:
  - Hilfsmittel zur Abbildung der Daten- und Kontrollstrukturen
  - Künstliche Sprache zur präzisen Formulierung von Algorithmen
  - Eine Folge von auf einem Computer ausführbaren Anweisungen

Definition für „Computerprogramm“ (2)

- Gemeinsames Element: Programmiersprache
- Aber wie wird eine solche definiert? Erforderliche Aspekte sind:
  - Formalisierung: Keine menschliche Interpretation nötig
  - Ablauffähigkeit: Wenn nicht → Entwurfsmaterial
    - Problem: Unmittelbarkeit, Syntaxfehler
  - Universalität: Alle Algorithmen sollen (sinnvoll!) umsetzbar sein
    - Statischer Teil: Strukturierung von Daten (Aber: Datentypen erforderlich?)
    - Dynamischer Teil: Sequenz, Verzweigung, (pot. unendliche) Schleifen
  - Ähnlichkeit zu „Turing-Vollständigkeit“
- Kein bestimmtes „Modell“ erforderlich (→ Nicht „Befehl“, „Anweisung“, …!)
  - Imperative Sprache → Schleife ≈ Funktionale Sprache → Rekursion
    - Beides ermöglicht „Widerholungen“ mit statisch endlicher Länge, die dynamisch potentiell unendlich lange dauern!

Irrelevante Aspekte

- Weite Verbreitung: Auch Spezial-Programmiersprachen eines einzelnen Unternehmens erzeugen Programme
  - Exakte Definition erforderlich: Auch andere müssten potentiell können!
- Art der Programmiersprache: Konkrete technische Umsetzung ist unerheblich
- Tatsächliche Ausnutzung aller Aspekte: Ein konkretes Programm bleibt ein Computerprogramm, auch wenn es keine Verzweigung/Schleife/… enthält
  - Schutz kann dann durch das erforderliche (Minimal-)Niveau entfallen
- Konkrete Funktion: Korrekt, legal, sinnvoll, kommerziell erfolgreich → Egal!
- Korrektheit: Auch Programme mit Syntaxfehlern sind in der jeweiligen Programmiersprache abgefasst
  - Aber: Nicht ablauffähig → Nur Entwurfsmaterial!

Laufzeitumgebung und Universalität

- Weitere Programme erforderlich: Fast jede SW benötigt ein Betriebssystem
  - Interpreter, Bibliotheken, Laufzeitumgebungen könnten ein Problem sein
- Aber: Laufzeitumgebungen sind einheitlich für eine Vielzahl an Programmen!
  - Daher kein Problem der Universalität: Ein Betriebssystem → Unzählige Programme
- Was ist bei Klassenbibliotheken? Diese benötigen eine „Laufzeitumgebung“ (=Hauptprogramm) dass speziell für sie geschrieben wird
  - Nur genau für diese eine Bibliothek geeignet
  - Dann kein Computerprogramm mehr, sondern nur mehr Entwurfsmaterial für ein solches → Gleicher Schutz
  - Alternative: Bei vielen Programmiersprachen ist „Introspektion“ möglich; dies erlaubt es, **ein** Hauptprogramm für **beliebige** Bibliotheken zu schreiben. Aber: Konkrete Funktionalität ermöglicht dies nicht, lediglich einen technischen „Aufruf“!

Anwendungsbeispiele: HTML (zum Aufwärmen!)

- HTML ist keine Programmiersprache
  - Statische Aspekte (Datenorganisation) ist vorhanden, aber dynamische Aspekte fehlen: Weder sind Berechnungen möglich, noch können Schleifen formuliert werden
  - Achtung: Eingebettetes JavaScript ist sehr wohl eine Programmiersprache!
- Und der Browser als „Laufzeitumgebung“?
  - Ändert nichts daran, dass eine Webseite (ohne Benutzereingabe!) rein passiv bleibt
- Und was ist mit CSS?
  - Erlaubt zumindest Verzweigungen (Bedingungen); HTML5+CSS3 ist (ev) Turing-vollständig, dh es könnte damit potentiell ein beliebiger Computer simuliert werden!
- Aber: Allgemeiner Konsens, dass HTML keine Programmiersprache ist
- Ergebnis: Kein Computerprogramm (was rechtlich schon länger klar ist!)
  - Und: LaTeX, PostScript, … (beide klar Turing-vollständig!); PDF (nicht T.-v.) → ???

Anwendungsbeispiele: Konfigurationsdateien

- Ähnlich wie Webseiten: Sowohl Konzept wie auch konkrete Erstellung kann viel Arbeit sein und wichtige Daten enthalten; Exakte Syntax und Semantik
- Werden vom Computer als Anweisungen verstanden: Was & Wie zu tun ist
- Aber:
  - Nur statische Strukturierung
  - Keine dynamischen Elemente
  - Wiederholung, selbst wenn unbegrenzt: Keine unendlichen Schleifen
  - Spezifizieren nur, **ob** etwas zu tun ist, aber nicht **was** → Das steht im Programm!
    - Einzelfälle; keine allgemeinen Regeln wie bei logischen/funktionalen Sprachen
- Daher: Kein Computerprogramm
  - Achtung: Business-Rules etc. können enthalten sein; analog zu HTML+JavaScript können diese „Computerprogramme“ sein → Kein Einfluss auf Konfigurationsdatei!

Anwendungsbeispiele: Datenbankdefinitionen

- Eine gute Datenbankstruktur zu entwerfen ist schwierig und aufwändig, auch wenn es dafür Entwurfs-Regeln gibt, zB Normalisierung
- Was ist alles darin enthalten:
  - Struktur der Daten; inkl. Datentypen
  - Trigger: Wenn best. Bedingungen eintreten werden gewisse Aktionen ausgeführt
  - Abfragen (zB in SQL) können trivial bis extrem komplex (Unterabfragen, Bedingungen, Gruppierung, Sortierung etc.) sein
    - Abfragen können auch direkt in der Datenbank gespeichert werden („Views“)
  - Programme („Stored procedures“) können gespeichert sein und beziehen sich auf die konkreten Strukturen
    - Allerdings wird zwischen SQL und SQL-PL (oder anderem, zB Java) strikt unterschieden (Structured Query Language / SQL Procedural Language)!

Anwendungsbeispiele: Datenbankdefinitionen

- Berücksichtigt man die Definition von vorher:
  - Formalisierung: Ist vorhanden. Die Definition ist klar und exakt möglich/erforderlich.
  - Ablauffähigkeit: Ohne Datenbank-SW passiert nichts. Diese ist aber universell, da zB SQL (nach Standard!) in einer beliebigen DB-SW läuft, bzw. eine DB-SW beliebige Datenbanken betreiben kann. Die DB-SW ähnelt daher einem Betriebssystem.
    - Daher ist die Datenbank aber auch **nicht** ein **integraler Teil der DB-SW**!
  - Universalität: Statischer Teil (Datenstrukturierung) ist da, dynamischer Teil fehlt
    - Abfragen und Programme beruhen zwar auf der Struktur, gehören aber nicht zu ihr dazu; Unterscheidung zwischen der DB-Struktur und ev. enthaltenen SQL-basierten Programmiersprachen
      - Diese Programmiersprachen funktionieren auch ohne Tabellen!
    - Ähnlich: Ersatz der DB durch eine Datei → Das darauf zugreifende Programm ist ein Computerprogramm, die Datei oder ihre interne Struktur jedoch nicht

Anwendungsbeispiele: Schnittstellendefinitionen

- Schnittstellen sind in einer Programmiersprache definiert, oft sogar nur ein „Extrakt“ daraus (physisch integriert)
  - Ähnlich zu Konstanten (zB Fakten) und sonstigen Daten (zB Text): Physisch enthalten, aber nicht Teil des „Computerprogramms“ → Ggf separater Schutz!
  - Ähnlich wie Datenstrukturen uU (!) stark mit dem Algorithmus und der konkreten Implementierung verwoben (ähnlich: „interne“ vs „externe“ Schnittstellen)
- Dynamische Elemente fehlen jedoch:
  - Es wird nur beschrieben, was aufgerufen werden **kann** (ev auch Reihenfolge), aber nicht wie es nun konkret aufgerufen werden **soll** oder **woraus es besteht**
    - Unterschied zu logischen/funktionalen Sprachen: Kein „Rezept“, da jedes Element für sich alleine steht (keine Abbildung Eingabe → Ausgabe)
- Daher kein Schutz als Computerprogramm, aber ev (!) als Teil eines solchen

Anwendungsbeispiele: Klassenbibliotheken

- Hier ist zu unterscheiden:
  - „Schnittstelle“ der Klassenbibliothek: Siehe Schnittstellendefinitionen
    - Beinhaltet: Konzept, Aufteilung in Klassen, Struktur der Klassen, Hierarchie, …
  - „Implementation“ der Klassenbibliothek: Hier untersucht!
- Eine Bibliothek ist alleine nicht ausführungsfähig
  - Siehe oben: Daher kein Computerprogramm, sondern nur Entwicklungsmaterial für eines, ähnlich wie ein Programm das erst zur Hälfte fertig ist
  - Ob auch alle Teile (oder überhaupt etwas) verwendet wird, spielt keine Rolle
    - Andere Meinungen sind mM falsch: Ein Programmteil/-entwurf kann auch dann geschützt sein, wenn er nicht für ein ganz konkretes Programm geschaffen wird
    - Denn der Einsatzzweck eines Programmes ist für den Schutz irrelevant, daher ist auch unbedeutend, wenn es (jetzt noch) keinen solchen gibt

Anwendungsbeispiele: VHDL

- Very High Speed Integrated Circuit Hardware Description Language
  - Wird uA verwendet, um die “Interna” von Mikrochips zu beschreiben
  - Hauptsächlich für Digital-Systeme verwendet; standardisiert
  - „Hardwarebeschreibungssprache“, aber auch als Programmiersprache bezeichnet
    - Eigentlich falsch: Nicht die **Hardware** wird beschrieben, sondern ein **Verhalten**!
  - Stark formalisiert und standardisiert (=ablauffähig); sowohl statische wie auch dynamische Aspekte → Kann vollständige CPU (potentiell kompletten Computer!) beschreiben und mit entsprechender Software auch emulieren → Absolut universell
- Ergebnis: Schutz als Computerprogramm
  - Untermenge: „Synthesefähig“ → Hardware kann automatisch generiert werden
    - „Freiwillige Selbstbeschränkung“, daher keine Änderung
- Nicht umfasst: Daraus generierte Hardware/Pläne dafür; nicht mehr formell beschrieben oder emulationsfähig (entspricht der Ausgabe eines Programms)!

Ausblick

- Fokus auf „Programmiersprache“ und die Elemente
  - Formalisierung,
  - Ablauffähigkeit und
  - Universalität (Statisch + Dynamisch vollständig; sinnvoll einsetzbar)
    - Weiterhin etwas problematisch: Wann ist sie „sinnvoll“ einsetzbar?
      - Ev: Würde ein Programmierer sie zumindest in Sonderfällen verwenden?
- erlaubt einfachere Beurteilung, was ein Computerprogramm ist und was nicht
- Vorteile:
  - Abstraktion von Massen-Programmiersprachen und kein alleiniger Fokus auf imperative („Befehl“, „Anweisung“,…) Sprachen
  - Leichter zu handhabende Definition, da die Elemente einfacher und nachvollziehbar überprüft werden können sowie unabhängig von einem konkreten Programm sind

Vielen Dank für Ihre Aufmerksamkeit!

Michael Sonntag

Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM)

Johannes Kepler Universität Linz, Österreich

sonntag@fim.uni-linz.ac.at