Motivation

Informatik-Produkte wie Hard- und Software, im allgemeinen *Programme*, bestimmen immer häufiger die Abläufe des alltäglichen Lebens. Dies sind oft sicherheitskritische Anwendungen, wie Electronic Banking, vollautomatische Steuerungen von Komponenten in Flugzeugen, etc.

In den 60er und 70er Jahren, als die Informatik sich langsam zu einer eigenständigen Wissenschaft entwickelte, war die Hauptanforderung an sie noch, Methoden bereitzustellen, mit denen Programme erzeugt und ausgeführt werden können. Dieses Problem ist mittlerweile zur Genüge gelöst. Es gibt unzählige Programmiersprachen, Entwicklungsumgebungen, Compiler, Compiler-Compiler, etc., und der Intel-Prozessor der jeweils nächsten Generation wird auch als Selbstverständlichkeit angesehen.

Mit diesen weit entwickelten Methoden wird es zunehmend schwieriger zu verstehen, was ein gegebenes Programm wirklich macht. Damit wächst nicht nur die Gefahr die Missbrauchs, sondern es wird auch schwieriger, Programme herzustellen – vergleichbar zur Herstellung von Medikamenten, deren Nebenwirkungen man abstellen, aber wenigstens doch einschränken möchte. So ist die größte Herausforderung an die Informatik heutzutage das Herstellen korrekter, sicherer, verläßlicher und vertrauenswürdiger Hard- und Software.

Dies wird z.B. von der Software-Technik durch Methoden unterstützt, die das Herstellen von Software klar strukturieren, um so unerwünschte Nebeneffekte zu vermeiden. Außerdem ist ausgiebiges Testen ein Muss im Software-Design-Prozess. Testen hat jedoch den Nachteil, dass erstens, nur die Anwesenheit, aber nicht die Abwesenheit eines Fehlers dadurch belegt werden kann. Andererseits ist es sehr schwierig, Fehler durch Testen zu finden, die nur sehr selten auftreten und evtl. kontextabhängig sind.

Die theoretische Informatik fing an, ihren Beitrag zum Design verlässlicher Programme in der Form von *Semantik* zu leisten. Die denotationelle oder operationale Semantik eines Programms beschreibt sein Verhalten vollständig, aber leider ist sie nur wirklich benutzbar auf sehr kleinen Programmen.

Für größere Programme bietet sich *Abstraktion* an. Dabei wird ein Programm auf sein essentielles Verhalten reduziert. Dies hat zwar den Nachteil, dass Korrektheit nur in Bezug auf das abstrahierte Programm gezeigt werden kann (wenn überhaupt) und Aussagen über dessen Korrektheit bei falscher Abstraktion keine Rückschlüsse auf das ursprüngliche Programm zulassen. Aber es hat einerseits den Vorteil, dass dadurch die Korrektheit einigermaßen komplexer Programme gezeigt werden kann. Andererseits ermöglicht es auch die Verifikation von Programmen während des oder vor dem Software-Design-Prozess statt nach dem fertigen Abschluss der Entwicklung. In Anwendungen der Ingenieurswissenschaften hat sich dieses Vorgehen längst bewährt und ist zum Standard geworden.

Dies wirft natürlich sofort die Frage nach der Definition von Korrektheit auf. Dies kann offensichtlich keine Aussage der Form "niemand außer mir hebt Geld von meinem Konto ab" oder "die Rakete überlebt den Start" sein, genausowenig wie ein Computer einfach so alle Information über den Mord an J.F. Kennedy aus dem Internet zusammenstellen kann. Stattdessen stellen wir Methoden zur Verfügung, dies es ermöglichen, Korrektheitsaussagen in konkreten Fällen effektiv und unmissverständlich zu konstruieren. Als Systeme, in denen diese Aussagen formalisiert – d.h. spezifiziert – werden, betrachten wir *temporale Logiken*.

Diese existierten in der Form *modaler Logiken* bereits lange, bevor die Informatik geboren wurde, insbesondere in der Philosophie. Durch Pnueli's Arbeit an der *Linear Time Temporal Logic* LTL [Pnu77] Ende der 70er Jahre, und auch Arbeiten von Pratt, Fischer und Ladner an *Propositional Dynamic Logic* [FL79, Pra78] aus dieser Zeit haben solche Logiken Einzug in die Informatik gefunden.

Kurze Zeit später wurde der Begriff *Model Checking*, der mittlerweile fest mit temporalen Logiken verknüpft wurde, geprägt: Dies bezeichnet das wohldefinierte Problem, zu einer Formel einer Logik und einer ihrer Interpretationen festzustellen, ob diese ein Modell für die Formel ist. Anfang der 80er Jahre haben Queille, Sifakis [QS82] und Clarke, Emerson, Sistla [CES83] dann erkannt, dass sich das Problem der *Verifikation* von Programmen bzgl. gewisser Korrektheitseigenschaften auf das Model Checking Problem reduzieren läßt. Mittlerweile wird der Begriff Model Checking bereits als solches verstanden.

In dieser Vorlesung werden die bekanntesten temporalen Logiken vorgestellt und aus theoretischer Sicht beleuchtet. Uns interessieren insbesondere die Komplexitäten ihrer Entscheidungsprobleme, ihre Ausdrucksstärken und ihre Beziehungen zueinander. Die Vorlesung soll ihre Hörer im Umgang mit temporalen Logiken vertraut machen, um als Informatiker im Beruf einen leichteren Einstieg in das Gebiet der Verifikation von Programmen zu haben.